

A Model-Based Service Customization Framework for Consumer Variability Management in Service-Oriented Architectures

by

**Sandile Wilmoth Dlamini
(200702501)**

A dissertation submitted in fulfillment of the requirements for the degree of

Master of Science in Computer Science

Faculty of Science and Agriculture

Department of Computer Science

University of Zululand

KwaDlangezwa

RSA

Supervisor: Prof. M.O Adigun

Co-supervisor: Mr. P. Tarwireyi

2014

DECLARATION

I, Sandile Wilmoth Dlamini, hereby declare that except where due acknowledgement has been made, this dissertation describes my own research work; and the work has not been submitted previously, in whole or in part, to another tertiary institution for any other degree or professional qualification.

Signature: _____

Date: _____

DEDICATION

*This dissertation is dedicated
to my family*

ACKNOWLEDGEMENTS

The successful completion of this research has been made possible through the contributions of many people for the past two and half years. Their contributions came in different forms and were very important to the accomplishment of my study. Thus, I would like to express my appreciation for their help and support, and show my gratitude.

First, I would like to appreciate and thank God the Almighty, for giving me the strength to rise out of a large number of difficult circumstances to finish the journey I have started; and for teaching me how to develop patience with persistence, and how to be strong but compassionate.

Many thanks go to my supervisors, Professor Matthew O. Adigun and Mr. Paul Tarwireyi, for giving me advice, support and guidance without pushing too hard; and for being so patient and understanding all the way. This work would never have been possible if it were not for the freedom they gave me to choose an important and controversial research topic that interested me.

I am grateful to the members of the Centre of Excellence for Mobile e-Services, particularly, Mr. Edgar Jembere for fruitful discussions and feedback on early drafts of this dissertation, although he had a busy schedule himself.

I would also like to thank Telkom SA SOC Limited for their financial support during the course of this research.

Finally, I would like to express my appreciation and thanks my family and friends, for being there when I needed them; and for putting up with my obsessive and endless rants about all the various problems encountered throughout.

ABSTRACT

In today's service-oriented business environments, the standard Publish-Find-Bind model as embodied by the Service-Oriented Architecture (SOA) paradigm presents a new strong challenge in the consumption and applicability of services to its consumers. This is because services in SOA-based environments are not built and published for predefined consumers; rather they are advertised for potentially many unknown consumers. Thus, they could be (re) used by various anonymous consumers with varying requirements and business needs. Hence, to increase service applicability and efficiency in the consumption of services, as well as to stay relevant in today's global market economy, service providers are expected to provide services covering such a wide variety of demands. However, they are still faced and have to deal with a number of problems which need to be balanced.

Consequently, this research work addresses the problem of how to deliver customizable software services, as a way to address and/or increase the applicability and efficiency in the consumption of software services. In particular, this research proposed a service customization framework called *FreeCust*, which exploits the feature modeling concepts or techniques from the Software Product Line Engineering (SPLE) discipline.

The FreeCust framework as suggested in this research was constructed, validated, and evaluated through practical use case scenarios, proof-of-concept prototype implementations, experiments, and a comparative (static) analysis. This was to show its utility, technical feasibility, functional correctness, and business benefits. The evaluation and validation results demonstrated that the FreeCust approach has the potential or is appropriate for minimizing the complexities involved in consumers' service customization processes and increasing service applicability.

TABLE OF CONTENTS

DECLARATION	i
DEDICATION	ii
ACKNOWLEDGEMENTS	iii
ABSTRACT	iv
TABLE OF CONTENTS	v
LIST OF FIGURES	ix
LIST OF TABLES	xi
LIST OF ACRONYMS	xii
LIST OF PUBLICATIONS	xiv
Chapter 1	1
INTRODUCTION	1
1.1. Research Context Overview	1
1.2. Problem Domain	3
1.3. Statement of the Problem.....	6
1.4. Rationale of the Study.....	7
1.5. Research Goal and Objectives	8
1.4.1. Research Goal	8
1.4.2. Research Objectives.....	8
1.6. Research Methodology	8
1.5.1. Literature Survey	9
1.5.2. Framework Development.....	10
1.5.3. Proof-of-Concept	10
1.7. Scope and Delimitation.....	11

1.8. Definition of Key Terms	12
1.9. Structure of the Dissertation	12
Chapter 2	14
BACKGROUND AND LITERATURE REVIEW	14
2.1. Introduction.....	14
2.2. Service-Oriented Architecture	15
2.2.1. Basic Component Types	16
2.2.2. Basic Design Principles	17
2.2.3. Implementation Technologies.....	19
2.3. The GUISET Project.....	22
2.4. SOA and GUISET Challenges.....	25
2.5. Software Product Line Engineering.....	26
2.5.1. Software Product Line	27
2.5.2. Model-Driven Development in SPLE.....	30
2.5.3. Variability Management	32
2.5.4. Feature Modeling.....	34
2.6. Overview of Related Work	37
2.7. Summary	40
Chapter 3	42
DESIGN OF THE MODEL-BASED SERVICE CUSTOMIZATION FRAMEWORK	42
3.1. Introduction.....	42
3.2. Design Criteria	43
3.3. The FreeCust Approach	44
3.3.1. Framework Architecture	44
3.3.2. Extended Feature Model	49

3.4. Example Application Scenarios	52
3.4.1. Live Match Score Application Scenario	52
3.4.2. Purchase Order Application Scenario	54
3.5. Summary	56
Chapter 4	57
IMPLEMENTATION AND EVALUATION	57
4.1. Introduction.....	57
4.2. Implementation Overview	57
4.2.1. FreeCust in UML	58
4.2.2. FreeCust Underlying Technologies, Tools, and System Packages	61
4.2.2.1. The GUI-flavored Mechanism.....	63
4.2.2.2. The Feature Management Engine	64
4.2.3. FreeCust Proof-of-Concept Implementation.....	66
4.2.3.1. The Development and Discovery of Software Service Feature Models	66
4.2.3.2. The Validation of Service Customization Requests	70
4.3. Experimental Evaluations	72
4.3.1. Usability Experiment	72
4.3.1.1. Experimental Design.....	73
4.3.1.2. Process, Results and Analysis.....	74
4.3.2. Request Validation Experiment	79
4.3.2.1. Experimental Design.....	80
4.3.2.2. Process, Results and Analysis.....	80
4.4. Experimental Evaluations Result Discussion	81
4.5. Comparative Analysis	82
4.6. Summary	85

Chapter 5	87
CONCLUSION AND FUTURE WORK	87
5.1. Research Summary	87
5.2. Conclusions.....	89
5.3. Review of the Research Methodology and Contributions	91
5.4. Limitations and Future Work.....	94
BIBLIOGRAPHY	96
APPENDICES	112
Appendix A: Consent Form.....	112
Appendix B: Background Questionnaire (Pre-Experiment)	114
Appendix C: Usability Questionnaire (Post-Experiment)	115
Appendix D: Detailed Usability Evaluation Instructions or Task List	116
Appendix E: XMethods Demo Service Terms and Conditions	117
Appendix F: Live Match Score Service Feature Model and Consumer’s Feature Configurations Request in XML.....	118

LIST OF FIGURES

Figure 1.1: SOA Architectural Model	2
Figure 1.2: Overview of the Research Methodology	9
Figure 1.3: Dissertation Structure, with Dependencies between Chapters	13
Figure 2.1: GUISET Architecture	24
Figure 2.2: SPL Engineering Overview	28
Figure 2.3: A Sample Feature Model Diagram	35
Figure 3.1: FreeCust Framework Architecture	45
Figure 3.2: Interaction Diagram of the Framework Components	48
Figure 3.3: Non-Functional Property (NFP) Model	49
Figure 3.4: Cardinality-Based Feature Model with Non-Functional Properties	50
Figure 3.5: A Sample Feature Model for the Live Match Score Service	54
Figure 3.6: A Simplified Purchase Order SaaS Feature Model	55
Figure 4.1: FreeCust Use Case Diagram	59
Figure 4.2: FreeCust Activity Diagram	60
Figure 4.3: High-Level Layered Technological View of FreeCust Implementation	61
Figure 4.4: The Dependencies among the Core Packages that implemented the GUI-flavored Mechanism	64
Figure 4.5: The Dependencies among the Core Feature Management Engine Packages	64
Figure 4.6: Provider's Live Match Score Service Feature Model in FreeCust	67
Figure 4.7: XMethods Service Directory	68
Figure 4.8: Automatic Selection of an Interdependent Feature (bet365)	69
Figure 4.9: bet365 Customization Request	70

Figure 4.10: Key Java Snippet for Validating Service Customization Requests.....	71
Figure 4.11: Participants Demographic Results (n=10).....	75
Figure 4.12: Participants Task Success Levels (n=10)	77
Figure 4.13: Participants Cognitive Load Using a 7-point Semantic Differential Scale (n=10) ..	78
Figure 4.14: Ease of Use Results Using a 7-point Likert scale (n=10)	78
Figure 4.15: Overall Satisfaction Using a 7-point Likert scale (n=10).....	79
Figure 4.16: Service Customization Request Validation	81

LIST OF TABLES

Table 4.1: Summary of Key Platforms and Frameworks Used In the Realization of FreeCust ...	62
Table 4.2: Usability Evaluation Tasks	75
Table 4.3: Comparative Analysis of Existing Service Customization Approaches.....	85

LIST OF ACRONYMS

CBE	Component-Based Engineering
CBFM	Cardinality-Based Feature Modelling
CORBA	Common Object Request Broker Architecture
CSP	Constraint Satisfaction Problem
CSUQ	Computer System Usability Questionnaire
DSL	Domain-Specific Languages
ebXML	electronic business XML
EMF	Eclipse Modelling Framework
FAMA	FeAture Model Analyzer
FODA	Feature-Oriented Domain Analysis
FOPLE	Feature-Oriented Product Line Engineering
FORM	Feature-Oriented Reuse Method
FreeCust	Feature Model-based Service Customization
GSD	Generative Software Development
GUI	Graphical User Interface
GUISET	Grid-based Utility Infrastructure for SMMEs Enabling Technology
HTTP	Hypertext Transfer Protocol
ICT	Information and Communication Technology
ISO	International Organization for Standardization
JMS	Java Messaging Service
MDD	Model-Driven Development

MDE	Model-Driven Engineering
NASA-TLX	NASA Task Load Index
NFP	Non-Functional Property
OOD	Object-Oriented Databases
SaaS	Software-as-a-Service
SAT	Boolean Satisfiability
SMMEs	Small, Medium and Micro Enterprises
SMTP	Simple Mail Transfer Protocol
SOA	Service-Oriented Architecture
SOAP	Simple Object Access Protocol
SPL	Software Product Line
SPLE	Software Product Line Engineering
SSDL	SOAP Service Description Language
UBR	UDDI Business Registry
UML	Universal Modeling Language
URI	Uniform Resource Identifier
W3C	World Wide Web Consortium
WSDL	Web Services Description Language
WSIL	Web Service Inspection Language
WSOL	Web Service Offerings Language
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

LIST OF PUBLICATIONS

Parts of the work presented in this dissertation have been published in the following conference and journal paper(s):

- **S. W. Dlamini**, P. Tarwireyi, and M. O. Adigun, “On the Delivery of Consumer Variability-Aware Services in Service-Oriented Architectural Environments,” In: *Proceedings of the 16th Annual Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, Spier Wine Estate, Stellenbosch, Western Cape, South Africa, 2013.
- **S. W. Dlamini**, P. Tarwireyi, and M. O. Adigun, “Maximizing Web Service Applicability and Consumption through Customization with Feature Modelling,” In: *Proceedings of the 15th Annual Conference on World Wide Web Applications*, CPUT, Cape Town, Western Cape, South Africa, 2013.
- **S. W. Dlamini**, P. Tarwireyi, and M. O. Adigun, “A Model-Driven Approach for Managing Variability in Service-Oriented Environments,” In: *International Journal of Information Technology & Computer Science*, vol. 8(1), pp. 91 – 97, 2013.

INTRODUCTION

1.1. Research Context Overview

The University of Zululand's Centre of Excellence for Mobile e-Services is currently working on a project called Grid-based Utility Infrastructure for SMMEs Enabling Technologies (GUISET). This project was proposed by Adigun *et al.* (2006), as an infrastructure aimed at addressing the problems of software and hardware acquisition experienced by resource constrained enterprises such as Small, Medium, and Micro Enterprises (SMMEs) in Africa. In essence, GUISET was motivated and informed by the following main reasons: (i) the inability of SMMEs to afford the cost of owning Information and Communication Technology (ICT) infrastructures and services, and (ii) the lack of ICT experts within SMMEs.

One of the key success factors of GUISET is the efficient delivery of ICT solutions that meet consumer demands. Accordingly, to enable SMMEs to have access to ICT services on-demand, in support of their business processes, GUISET leverages on service-orientation and implements the Service-Oriented Architecture (SOA) paradigm. Thus, the challenges towards realizing SOA are GUISET challenges as well.

SOA is an architectural paradigm that has recently become one of the preferred choices for designing and developing systems that are characterized by coarse-grained services and service consumers (Lewis, Smith, & Kontogiannis, 2010). One of the key motivating factors behind the adoption of this paradigm is the capacity to deliver flexible ICT solutions, which have the ability to respond quickly and prove cost-effective to changing business or consumer requirements.

According to Papazoglou and Heuvel (2007), the other key characteristic for the popularity of SOA is that it addresses the requirements of loosely-coupled, standard-based and protocol independent distributed computing, by mapping enterprise information systems to the overall business process flow.

In SOA, software components are encapsulated as services. A service is a self-contained, loosely-coupled and reusable business element or software artifact that can be advertised, discovered and used in order to perform certain business operations or combined with other services to produce value-added business applications (Papazoglou & Georgakopoulos, 2003). For the provisioning and consumption of services, SOA embodies the model of Publish-Find-Bind (see Figure 1.1), in which service providers offer their application functionalities as a service (SaaS), by publishing and/or advertising them over a network (such as the Internet). Service consumers then search and discover these services on-demand, based on their business needs and requirements.

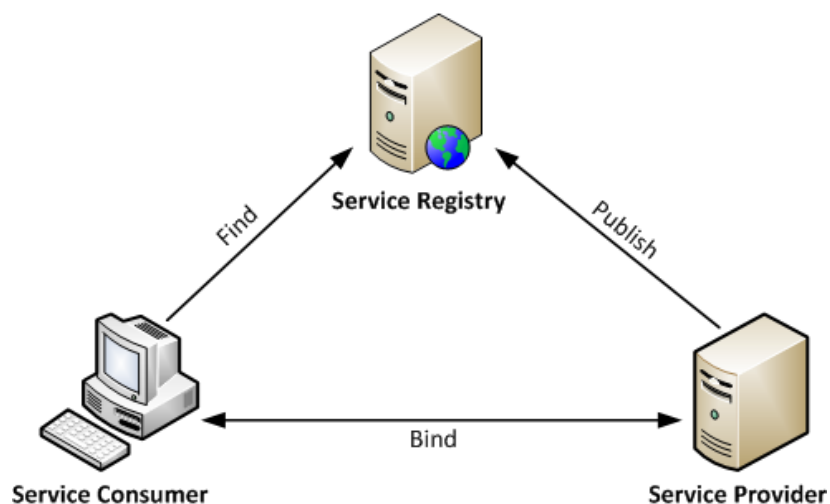


Figure 1.1: SOA Architectural Model (Papazoglou, 2003)

1.2. Problem Domain

Despite the major benefits and success of SOA in terms of its adoption, it is still facing some challenges. SOA presents a strong challenge in the consumption and applicability of services to service consumers. This is because services in SOA models are built with little or no knowledge about service consumers, that is, they are not developed and published for predefined consumers; rather they are advertised for potentially many unknown consumers. Hence, software services could be used by a number of anonymous consumers with varying *feature*¹ requirements and business needs. In addition, to drive business competitiveness in today's service-driven business environment, most business enterprises usually prefer services that are tailored to their business needs. Therefore, to increase service applicability and efficiency in service consumption, service providers are expected and/or required to provide services covering such a wide variety of demands. However, they are still faced and have to deal with a number of problems which need to be balanced.

First, in order for service providers to draw a significant number of service consumers, they need to manage and take into account the varying requirements of their potential consumers, by providing highly adaptable and flexible services. Second, to minimize development cost and achieve economies of scale, they need to make sure that different variants of their services preserve enough commonalities. According to Papazoglou and Heuvel (2007), service providers rely on industry best practices to address requirements variability. However, they cannot cover or handle all possible specific service consumers' requirements and business scenarios. Hence, the management of consumer requirements variability is still one of the challenges facing SOA.

¹ A feature is a distinguishing characteristic of a software item (IEEE Standard Glossary of Software Engineering Terminology, 1990).

One of the well-known approaches for dealing with consumers' requirements variability in SOA is service versioning. According to Sarang *et al.* (2007), versioning in SOA is very important because of re-usability. Versioning ensures that multiple implementations of the same thing can co-exist. In SOA, service versioning equates to the co-existence of multiple variants of the original service, which enables consumers satisfied with the original service to continue using it unchanged, while ensuring that a new variant of a service is created to meet the needs of consumers with different requirements (Woolf, 2007).

Another approach for managing requirements variability is through service customization (Frei, 2006), whereby service consumers are permitted to perform customizations in order to generate specific service variants meeting their feature requirements and business needs. In its basic form, service customization is described as a process whereby service consumers perform a set of operations (e.g., modifying service interface description documents) to adapt a service to their application scenarios. According to Stollberg and Muth (2010), this is a nontrivial task that requires both technical knowledge as well as business expertise and hence, most business enterprises usually appoint service experts to carry out this task. Moreover, there exist three major challenges to providing customizable services in SOA, and these are: (i) reduction of service customization complexities, (ii) validation of service customization requests, and (iii) the dynamic deployment of customized services.

The first challenge emanates from the fact that modifying a service description document such as the Web Services Description Language (WSDL) document is a daunting process, especially since services support a plethora of possible customization options with a massive number of dependencies scattered among those options. In addition, these dependencies are normally

expressed in natural language(s). For these reasons, it becomes a very complex task for service consumers to perform service customizations. Because of the previously explained complexities, the process of customizing a service can be a very error-prone process (Stollberg & Muth, 2009). Thus, consumers' customizations need to be automatically validated, to make sure that they do not breach any properties (i.e., functional and non-functional) described by service providers. The last challenge results from the fact that in SOA, service consumers usually do not have similar requirements and business needs. Thus, different customization requests can be generated by service consumers, and the implementations of each specific service variant will not be the same. Hence, to ensure that development time and computing resources are not wasted, a mechanism to avoid redundant running of service variants should be employed.

To this end, and based on the literature, approaches for addressing requirements variability in SOA can be classified into two categories: (i) design-time (service versioning) and (ii) run-time (service customization) approaches. In the first approach, variant services satisfying specific consumers' requirements are developed and deployed as individual services, meaning that service consumers are the ones responsible for searching, binding and invoking appropriate services. The second approach allows service consumers to perform run-time customizations in order to generate specific services satisfying their needs. In examining these two approaches, if there exist a small number of service consumers requiring variant services, the first approach is considered simpler and requires less effort from service consumers. However, if the number is very large (which is the case in today's service-driven business environments), the second approach becomes more appropriate, because developing and deploying individual service variants for each service consumer or for meeting all possible specific consumers' requirements will result in development cost and computing resources being wasted. The development effort

will also be wasted because the commonality of the core requirements will not be exploited to support reuse efficiently.

1.3. Statement of the Problem

From the foregoing discussion, it is clear that in today's service-oriented business environments, the different business sizes (i.e., large and small), and the ever-changing requirements of service consumers pose a great challenge in the applicability and consumption of services. Thus, to enhance service applicability and improve the consumption of services in such environments, mechanisms for handling and managing consumer-introduced variability (i.e., the variability caused by the commonalities and differences in service consumers' requirements) are becoming more fundamental. Hence, this work seeks to contribute by proposing a model-based service customization framework. Various solutions do exist that attempt to address requirements variability through enabling service customization in SOA, however, these solutions suffer from a number of drawbacks (as discussed later in Chapter 2). One of the major drawbacks in these solutions is that these solutions do not take into consideration the need to address the first challenge in providing an efficient service customization solution. That is, they do not consider reducing the complexities faced by service consumers when they are performing customizations. Thus, a crucial research question that this work was formulated to investigate is:

1. How can we ensure that complexities involved in consumers' service customization processes are minimized to a potential *efficient level*²?

² A level at which non-ICT experts (such as SMMs) without good background knowledge in Web service related technologies are able to customize software services.

The sub-questions that emanated from the central research question include:

- a. How can a mechanism that enables graphical and/or visual representation of service customization options be implemented?
- b. Can possible interdependencies among service customization options be automatically selected?

1.4. Rationale of the Study

As mentioned earlier, SOA has emerged as a cross-disciplinary and predominant architectural paradigm for implementing large distributed systems that are characterized by business services and service consumers (Hassanzadeh, Namdarian, & Elahi, 2011; Lewis, Smith, & Kontogiannis, 2010). However, with the continuing advancement and adoption of this (SOA) paradigm, and the evolution of service-oriented systems, as well as the frequent change of consumers' demands in service-driven business environments, the need for flexible business solutions that respond to requirements and business context changes is becoming more fundamental. Hence, there is a need for tools, methods, and mechanisms for supporting and managing variability, and improve reuse in service-based environments. According to Cohen and Krut (2010), the development of such tools and mechanisms is still on-going and solutions are still being sort. Thus, the work in this dissertation aims at contributing to the ongoing research towards finding and developing solutions for managing variability in service-oriented environments. In addition, this work also aims at contributing to GUISET. This work will ensure that the GUISET infrastructure can, in an *ad-hoc* manner and cost-effectively provide services that are highly adaptable and appropriate to its consumers.

1.5. Research Goal and Objectives

1.4.1. Research Goal

The goal of this work was to develop a service customization mechanism with less customization complexity, for managing consumer-introduced variability in SOA-based environments.

1.4.2. Research Objectives

The above goal was formulated as an equivalent of some lower-level objectives, which were:

- (i) To investigate how service properties (functional and non-functional) should be represented in order to support model-based service customization.
- (ii) To investigate existing mechanisms and techniques used to facilitate customization processes and for communicating customization options, which could enable or assist service providers and service consumers in service customization processes.
- (iii) To design and develop a model-based service customization framework.
- (iv) To implement as a proof-of-concept prototype the framework developed in (iii) for descriptive and experimental evaluations.

1.6. Research Methodology

The goal and objectives of this research were accomplished by means of following the design science research methodology (see Figure 1.2): (i) establishment of state of the art, (ii) solution design, and (iii) solution validation and evaluation, as described in software engineering and computer science (Elio *et al.*, 2008; Hevner *et al.*, 2004).

The aforementioned research steps consisted of three major activities, namely literature survey, framework development, and proof-of-concept. A brief overview of each of these activities is respectively given in the following sub-sections: 1.5.1 to 1.5.3.

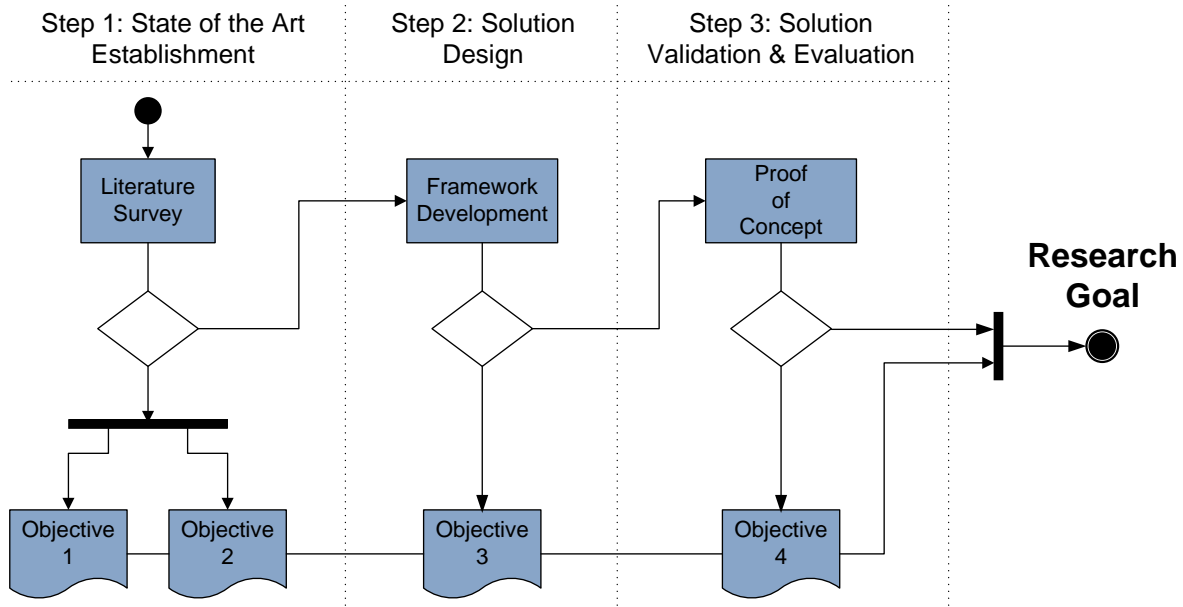


Figure 1.2: Overview of the Research Methodology

1.5.1. Literature Survey

As compared to conventional service provisioning and consumption, service customization puts into effect extra communications between service providers and service consumers. In other words, there are additional operations that need to be in place for the delivery and consumption of customizable services (e.g., propagation of customization request from the service consumer's side to the service provider). Hence, an extensive literature survey on some of the recent trends in the area of service computing, with special focus on how service customization is supported was conducted. A comprehensive review of other software engineering disciplines such as SPLE was also carried out, with the purpose of finding out how their methodology and paradigm

enable a flexible and adaptive system. Thereafter, an investigation on how services' functional and non-functional properties should be represented and communicated in order to support model-driven service customization was conducted. The information and knowledge gained from these investigations helped in identifying an appropriate mechanism for describing and communicating service variability information between service providers and consumers.

1.5.2. Framework Development

Based on the knowledge that was gained from the literature on what has already been achieved in addressing the issue(s) of variability in SOA and the recent trends of service computing, software product line and model-driven engineering, as well as the design requirements that were gleaned from reviewing and analyzing existing service customization solutions, a set of design principles that a complete service customization solution should support were drawn. A design of **FreeCust** - a **Feature Model-based Service Customization** framework was then developed according to the constructed design criteria.

1.5.3. Proof-of-Concept

This part of the research involved implementing and evaluating the developed service customization framework as a proof-of-concept prototype. In particular, the implementation was carried out using Eclipse, the Eclipse Modelling Framework (EMF), and some other relevant open-source frameworks and tools in the fields of service computing and software product line engineering. To demonstrate the applicability and utility of the proposed service customization solution, real-life descriptive application scenario(s) were constructed. A number of controlled experimental evaluations were also conducted to determine and evaluate the reliability and efficiency of the FreeCust framework.

1.7. Scope and Delimitation

The focus of this dissertation was to develop a service customization mechanism for managing consumer-introduced variability, i.e., the variability caused by the commonalities and differences in the requirements of service consumers in SOA-based environments. This mechanism needed to have less customization complexity. This, along with the duration of the research project, as well as the considerations of the environment in which FreeCust would function, meant that the major concern was to find appropriate tools and mechanisms to the specific research questions and/or challenges addressed in this dissertation. Thus, the FreeCust prototype developed in this dissertation is intended to demonstrate the proof-of-concept implementation (with special focus on the simplification of the service customization process), rather than the actual realization of a fully-fledged unified service creation environment for delivering customizable software services.

In brief, the process of customizing a software service involves both the service provider and consumer. Hence, this process is generally seen and broken down into two major parts. That is: (i) the part whereby service consumers comprehend customization options and perform service customizations, and (ii) the part where service providers derive and deploy customized software service variants and manage those variants.

The FreeCust proof-of-concept implementation and evaluations focus only on the first part. The realization of the second part is beyond the scope of this dissertation. However, occasional reference is made to the concepts related to this part (in the subsequent Chapters), for clarifying the core issues related to the proposed FreeCust solution.

1.8. Definition of Key Terms

Due to the fact that some important terms could have varying definitions in the literature and might cause confusion, this section provides definitions of such terms. The goal is to establish a consistent level of understanding for terms used throughout this dissertation.

Functionality: captures an intuitive notion of the amount or quantity of functional and non-functional properties contained in a delivered product or in a description of how the product is supposed to be (Fenton & Pfleeger, 1998).

Feature: a feature is defined as a logical unit of behaviour of a software system or product that satisfies functional and non-functional requirements. Features are a set of functionality by which different software or service products can be created and distinguished (Bosch, 2000).

Software Product Line (SPL): is a family of software-intensive products that share a common set of features, developed from a base set of core assets to satisfy a particular market segment, while allowing a specific margin for differentiation to satisfy different and specific customers' needs (Clements & Northrop, 2001; Pohl, Bockle, & Linden, 2005).

1.9. Structure of the Dissertation

The remainder of this dissertation is structured as follows:

- Chapter 2 gives background information on the fundamental concepts that are crucial for this dissertation and, a preliminary review of relevant research work, which establish a foundation for the contents presented in the rest of this dissertation.

- Based on the understanding of the literature reported in Chapter 2, a conceptual design of the FreeCust framework, as suggested in this dissertation is described in Chapter 3.
- Chapter 4 discusses the proof-of-concept implementation and reports on the experiments conducted to evaluate the FreeCust approach (presented in Chapter 3). It also compares FreeCust with other existing service customization solutions.
- Chapter 5 concludes the dissertation, outlines the contribution of this research work, and discusses research issues needing further investigation.

Figure 1.3 provides an illustration of the overall dissertation outline and how the different chapters relate to each other.

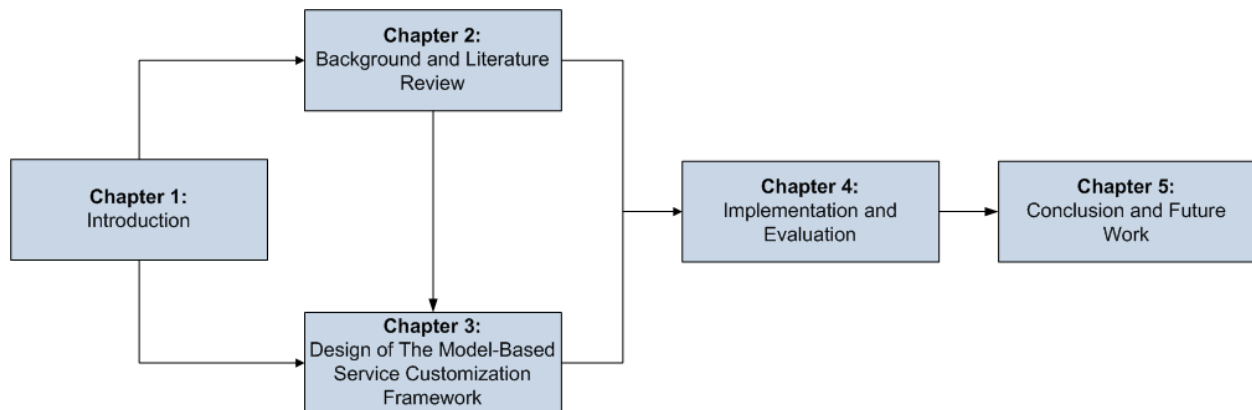


Figure 1.3: Dissertation Structure, with Dependencies between Chapters

BACKGROUND AND LITERATURE REVIEW

2.1. Introduction

In today's global economy, business enterprises either small or large are striving for growth and competitiveness in order to be at the cutting-edge. Moreover, to increase agility and flexibility of their business processes, business enterprises face the need to quickly react to changes in the market and business environments as such. The need to respond quickly and cost-effectively to ever-changing market conditions and business environments led many businesses to service-oriented architectures, where services are the fundamental building blocks that enable the reuse of business functionalities within and across business enterprises (Papazoglou *et al.*, 2007). As reuse becomes more and more the driving factor for developing cost-effective, agile, and high quality business solutions, services in service-oriented architectures could be (re) used by a large number of anonymous service consumers. Although these consumers share the same vision with regard to the offered services, their business needs, usage scenarios and applications are usually different from one consumer to another or change from time to time.

As mentioned in the previous chapter, the aim of this research was to develop a model-based service customization framework for managing consumer-introduced variability in SOA-based environments. This chapter, therefore, provides background information on the fundamentals and basic domain concepts used in this research and discusses existing relevant research work. Section 2.2 gives an overview of the SOA paradigm. It describes SOA underlying components, design guidelines, as well as implementation techniques. The idea of the GUISET undertaking is

provided in Section 2.3. Section 2.4 discusses the challenges of SOA that are similarly faced by the GUISET infrastructure, particularly, the issue of variability management. In view of the fact that variability management is the core subject in the field of SPLE, Section 2.5 is dedicated to this engineering discipline. Section 2.6 discusses existing related work, while the chapter is summarized in Section 2.7.

2.2. Service-Oriented Architecture

Over the past years, Service-Oriented Architecture (SOA) has attracted more and more interest from both organizations and the research community. This comes from the fact that SOA promises and/or enables the delivery of flexible ICT solutions to satisfy business goals. These goals include, easy and flexible integration with legacy systems, streamlined business processes, innovative solutions to customers, reduced costs, and agile adaptation and reaction to opportunities and competitive threats (Bianco, Kotermanski, & Merson, 2007; Papazoglou *et al.*, 2008; Trkman, Kovacic, & Popovic, 2011).

SOA is an architectural style that provides guidelines for building infrastructures or distributed systems which support rapid development and delivery of software components in a form of interoperable and loosely-coupled business aligned services (Hunaity, 2008; Lewis, Smith, & Kontogiannis, 2010). A service is a self-contained, reusable, and loosely-coupled unit of functionality or software artifact that can be advertised, discovered and invoked in order to perform a certain business operation or combined with other services to produce value-added applications (Papazoglou & Georgakopoulos, 2003). According to Shaw and Garlan (1996) as cited in Gohar (2010), an architectural style defines a vocabulary of component and connector types and constraints on how they can be combined or work together. For SOA, the basic

component types are: service providers, service registry, and service consumers (as discussed shortly in Section 2.2.1). Connector types include synchronous and asynchronous calls using Representational State Transfer (REST) protocol, bare Hypertext Transfer Protocol (HTTP), SOAP, and other messaging paradigms and/or infrastructures (Bianco, Kotermanski, & Merson, 2007). The Publish-Find-Bind interaction model as illustrated in Figure 1.1 shows how the components of SOA are constrained in their interactions.

2.2.1. Basic Component Types

The fundamental component behind the notion of service-orientation is, of course, the service itself. However, SOA is an architecture not only about software services; it is a relationship of three types of participants, as they are described below.

Service Providers are organizations or companies that offer their software functionalities as a service (SaaS) to the public, through advertising them over a network (such as the Internet). In principle, service providers create and publish service descriptions on a service registry to enable consumers to search and discover them (cf. Figure 1.1). A service description is a document or a file that contains the necessary information about the service. This document enables service consumers to understand the functionality of the offered services.

One of the other basic and important components of SOA is a mechanism for services to be discovered by potential service consumers. For this reason, the **Service Registry** (often also called a service broker) serves as a middleman between service providers and service consumers. The goal is to offer the possibility to provide access to service descriptions and to discover services. In fact, service registries allow providers to advertise their services by publishing their

service descriptions, and enable potential consumers to discover and retrieve them based on their demands.

Service Consumers are organizations or entities that look for a software service to execute a required business function. They are usually considered as a client, either an end-user, another service, or some other type of software module that might need a service (Lu & Yu, 2007; Papazoglou, 2003).

2.2.2. Basic Design Principles

According to Valipour *et al.* (2009), software architectures have some specific characteristics and principles that need to be followed in order to fully utilize their capabilities. This section takes a closer look at some of the special characteristics of SOA. These characteristics can also be found in other concepts of software engineering, but, the combination and descriptions are very specific to SOA and this dissertation.

- *Loose-Coupling*: is the most fundamental characteristic of SOA. Basically, coupling refers to the degree or level of dependencies among software system components (Krafzig, Banke, & Slama, 2005), and describes the way in which different components are connected. Coupling can be categorized as either loosely-coupled or tightly-coupled. A loosely-coupled software system is one in which each of its components has, or makes use of, little or no knowledge of the definitions of the other separate components (Pautasso & Wilde, 2009). SOA emphasizes developing loosely-coupled services, which means, there should be minimal dependencies between service consumers and providers. This encourages the independent design and evolution of a service's logic and provides interoperability with service consumers.

- *Discoverability*: is a prerequisite for using a service. Thus, service providers should allow their service descriptions to be easily discovered and understood by humans and other services that may be able to make use of their logic. In practice, the service and its capabilities should be discovered by either the software architect or an automatic mechanism. Discoverability is very important to leverage the true potentials of SOA, as it also promotes reuse of services. If the ability to find adequate services that offer the required functionality is not specified or not convenient for use, the stakeholders of SOA would rather build redundant services than to use existing ones.
- *Reusability*: is the principle that requires that a software service is self-contained in a way that it can be used for more than one business case. That is, it offers its capabilities such that they can be utilized for multiple usage scenarios to support requirements in different contexts. To exploit fully the benefits and potentials of SOA, effective methodologies to support systematic, agile, and cost-effective reuse during the development of software services are very important (Papazoglou & Heuvel, 2006).
- *Granularity*: is a relative measure to define how detailed a required piece of functionality must be in order to address the need at hand (Bloomberg, 2007). Subsequently, service granularity defines the scope of functionality exposed by a service (Papazoglou & Heuvel, 2006). Services can be fine-grained or coarse-grained. Fine-grained services address small units of functionality or exchange small amounts of data. Whereas, coarse-grained services encapsulate larger chunks of business capabilities within a single abstracted interface, reducing the number of service calls necessary to accomplish a business task (H. Ying, Wu, & Liu, 2010).

In SOA, the general idea in designing services is that services need to have business value (Krafzig, Banke, & Slama, 2005). This leads to coarse-grained services that are closely aligned to business functions. However, the drawback in designing coarse-grained services lies in the fact that they might return excessive quantities of data, be difficult to reuse, or prove difficult to change to meet new requirements (Kulkarni & Dwivedi, 2008). Consumers that only need parts of the offered capabilities might not be interested in using the provided service. As a result, service providers must determine and craft the appropriate optimum balance or level of service granularity with respect to all the principles listed above, to ensure that services are reusable and accommodate new consumers' requirements.

To be precise, services must be designed for appropriate granularities that offer greater flexibility to service consumers. Granularity should also make it easy for consumers to assemble a service to execute specific business scenarios. According to Kulkarni and Dwivedi (2008), well-designed or implemented services should be fine-grained enough to be reusable, but coarse-grained enough to make business sense.

2.2.3. Implementation Technologies

As described earlier, SOA is an architectural model, not a technology. Meaning, it cannot be bought and used, but can be realized and implemented using several technologies such as Web services, CORBA (Common Object Request Broker Architecture), and REST (Papazoglou & Heuvel, 2007; Pastore, 2008). However, the Web services method is currently the most popular technology for realizing SOA. Web services are a specific kind of software services that use Web technologies and protocols such as HTTP and XML, to expose their features to the Internet, and are usually implemented using open standards like SOAP (Simple Object Access Protocol),

WSDL (Web Services Description Language), and UDDI (Universal Description, Discovery, and Integration). This section, therefore, provides a brief overview of these concepts and other related ones.

SOAP: is an XML-based protocol for communication over standard transportation protocols, such as HTTP, Java Messaging Service (JMS), Simple Mail Transfer Protocol (SMTP), and the eXtensible Messaging and Presence Protocol (XMPP). SOAP is lightweight and geared for the exchange of information in a distributed environment. The use of XML technologies to define a messaging framework allows it to be platform and programming language-independent (Gudgin *et al.*, 2007). Web services take advantage of this fact in the exchange of messages with other Web services.

Web Services Description Language (WSDL): introduces a common grammar to describe Web services in an XML format (World Wide Web Consortium [W3C], 2007). It describes the Web service interface, which includes the service's location on the Web (mostly in the form of a URI.), as well as the functionality and interaction specifications of the Web service (Cavanaugh, 2006; Papazoglou & Heuvel, 2007). Web service consumers make use of the WSDL description to find and utilize Web services. Alternatives to WSDL are, the SOAP Service Description Language (SSDL) (Parastatidis *et al.*, 2005) and Web Service Offerings Language (WSOL) (Tosic, Patel, & Pagurek, 2002). Both SSDL and WSOL are also XML-based languages for describing Web services, however, SSDL only focuses on Web services that make use of SOAP, while WSOL enables and facilitates formal specification of functional and non-functional constraints, context monitoring, and relationships with other service offerings (Tosic, Lutfiyya, & Tang, 2006). According to Tosic, Patel, and Pagurek (2002), describing Web services in

WSOL in addition to WSDL, provides a method to select a more appropriate Web service for a particular situation. For more details about WSOL, interested readers are referred to Tomic (2004).

Universal Description, Discovery, and Integration (UDDI): is a technique which provides the infrastructure required to publish and discover Web services (Clement *et al.*, 2004). It enables organizations (service providers) to advertise their businesses by publishing their Web service descriptions in a public or private repository and also makes it possible for service consumers to search and inspect appropriate Web services. At the start of Web services, big vendors such as Microsoft, SAP, and IBM maintained a registry called the UDDI Business Registry (UBR) as a single repository for publishing Web service information (Al-Masri & Mahmoud, 2008a). But, this attempt has been neglected, because the employed categorization scheme, as well as the UDDI support for publishing and discovering services turned out to be insufficient (Stollberg, 2008). One of the reasons of the UDDI registry being insufficient, as stated by Al-Masri and Mahmoud (2008b), is the inability to support representation of Web services' non-functional properties. Hence, it does not provide a suitable Web service discovery method that is able to satisfy consumers' demands (Hunaity, 2008).

There exist various techniques that provide an alternative to UDDI, for example, ebXML (electronic business XML) and WSIL (Web Service Inspection Language). However, these registries are not as well-known as the UDDI registry. The ebXML standard is a centralized Web service registry which allows businesses to collaborate and conduct business activities (Dustdar & Treiber, 2005). This registry (ebXML) is broader than UDDI as it is able to store more data, such as business process models and collaboration-protocol profiles (Dustdar &

Treiber, 2005). The WSIL provides a distributed metadata model for Web service information. WSIL supports aggregating different types of Web service descriptions in a single file (Dustdar & Treiber, 2005). This means, there are no restrictions on the type of Web service information published for a Web service.

In summary, it should be noted that although Web services technology is the most prominent implementation platform for SOA, it is just one demonstration of SOA and does not necessarily fully adhere to SOA design principles (Erl, 2005; Huhns & Singh, 2005).

2.3. The GUISET Project

In Africa, as in most emerging and developing economies, the significance of SMMEs in stimulating economic growth and generating employment that results in local development is absolute. However, SMMEs, especially those in rural areas are faced by a number of business challenges, which hinder them from easily expanding their businesses, maintaining business competitiveness, engaging with their trading partners, and responding quickly to rapidly changing business markets. These problems include the inability to afford adequate ICT infrastructures, the lack of supply-chain volumes, limited understanding of ICT technologies, and little or no knowledge to inform decision making (Adigun *et al.*, 2006; Dai, 2010; Zadeh, Mukhtar, & Sahran, 2012).

In an attempt to overcome the aforementioned business problems, an open dynamic service-oriented and grid environment called Grid-based Utility Infrastructure for SMMEs Enabling Technologies (GUISET) was proposed (Adigun *et al.*, 2006). Technically, the concept of the GUISET infrastructure is based on the idea that there is the need for a technology or platform that would enable SMMEs to have access to ICT services on-demand, without owning the

infrastructure on which the services are running. GUISET leverages on service-orientation to enable services to be discovered and invoked in order to support the business processes of SMMEs. In this infrastructure, software services are shared among consumers with different needs, thus, as mentioned previously, services in SOA-based environments like GUISET could be (re) used by a number of anonymous consumers with varying feature requirements and business needs. Hence, the need to support and manage service changes and variability in such environments is a real necessity, with a view to increasing service applicability and efficiency in service consumption and ensuring that the GUISET infrastructure continues to provide highly flexible and valuable services to its targeted consumers.

GUISET is a three-tier layered architecture (see Figure 2.1), consisting of the Multi-modal Interfaces Layer, Middleware Layer, and Grid Infrastructure Layer. The Multi-modal Interfaces Layer deals with rendering information to GUISET clients and provides support for universal accessibility to all GUISET services. In essence, this layer establishes the necessary interactions with the external entities (e.g., SMMEs) and the Middleware Layer. The Middleware Layer is a utility broker, which manages the sharing of resources and enables dynamic service discovery, selection, and deployment. This layer underlines the link between the Multi-modal Interfaces Layer and Grid Infrastructure Layer, which is the layer at which GUISET services and resources reside.

As mentioned previously, GUISET leverages on the service-oriented archetype to offer software functionalities as a service (SaaS) to its targeted consumers. In the process of providing such software artifacts, GUISET acts as two different variants: (i) as a single service provider and (ii) as an intermediary provider. In the former variant, having both the resources and knowledge to

offer the required functionalities, GUISET provides software services by assembling its own resources (i.e., native GUISET services). The latter involves linking and having knowledge of the resources of third-party service providers which would meet the required service capabilities. The consumers (e.g., SMMEs) subscribe to the GUISET infrastructure and look for software services (service-on demand), and only require Web technologies to access and use the offered services.

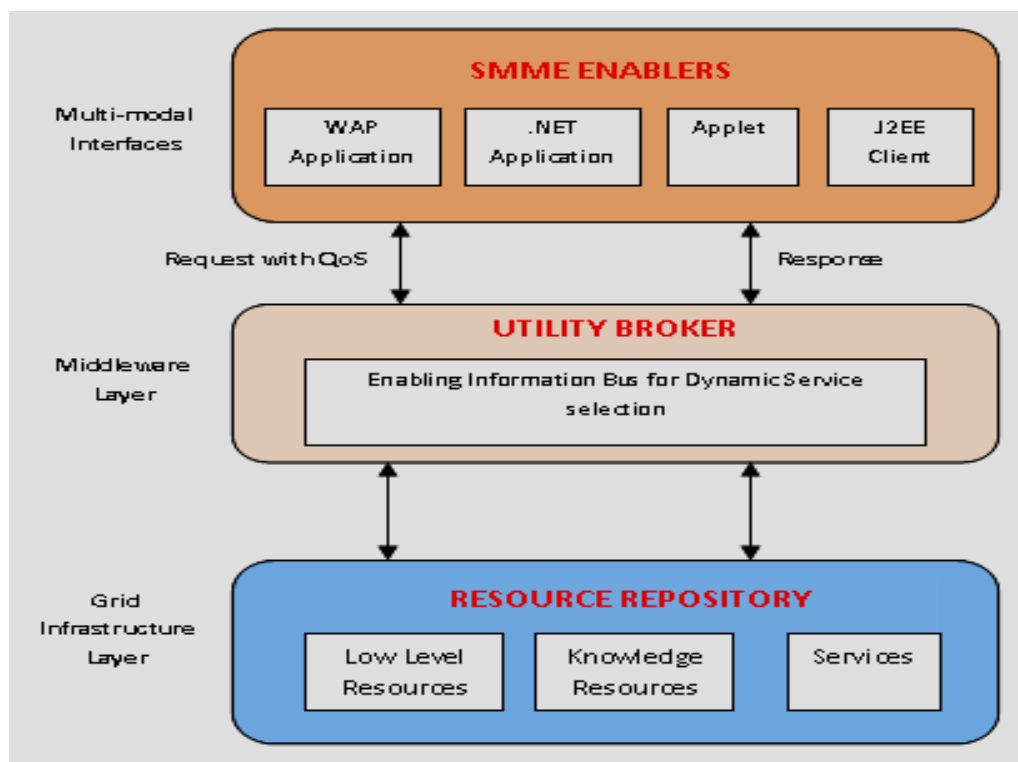


Figure 2.1: GUISET Architecture (Adigun *et al.*, 2006)

Delivering high quality services has always been the ultimate goal for GUISET. However, there are several challenges that hinder GUISET in achieving such, and these challenges need to be addressed for the GUISET infrastructure to be as successful as possible. Some of the GUISET

challenges are posed by the technology and computing paradigms that GUISET leverages on, and they are discussed in the next section.

2.4. SOA and GUISET Challenges

Although SOA is not new, it still poses a number of challenges to organizations and the business community as a whole. This section discusses some of these challenges, which include security, SOA governance, and variability management.

Security: as mentioned earlier, SOA is built on open standards such as XML, WSDL, SOAP, UDDI, and these standards do not have concrete security of their own. This, together with the heterogeneous, dynamic, and business-oriented nature of SOA, make services and SOA systems vulnerable to different attacks and hence make addressing security a challenging issue. Thus, appropriate security developments and mechanisms are very important to the success of SOA systems (Arsac, Laube, & Plate, 2013; Kanneganti & Chodavarapu, 2008).

SOA governance: is the process of ensuring and validating that assets and artifacts within the SOA architecture are acting as expected and maintaining a certain level of quality (Manes, 2005). Governance encompasses a broad area, including the management of all elements in SOA-based systems, such as software services, the service infrastructure, and artifacts across their life-cycle. As services are one of the most important reusable assets in SOA systems, lack of SOA and service governance would therefore lead to a number of flaws (e.g., having services that cannot be reused easily, because they were not designed with reuse in mind). Moore (2006) states that proper governance minimizes the risk of mismatched services as well as redundant development efforts. Hence, governance is also important to the success of SOA systems (Tibco, 2011).

Variability management: variability in software systems refers to the notion that the elements constituting the software architecture may vary (Aiello, Bulanov, & Groefsema, 2010; Williams & Carver, 2010), due to a range of factors including diverse customer needs, business strategies, and technical constraints. In SOA systems, variability can occur or manifest itself in various ways and at different levels of abstraction. For example, it manifests itself in the varying functional and non-functional requirements of service consumers, changing architectures, and changing execution environments. Requirements change because the business needs, usage scenarios, and applications of service consumers are often different from one consumer to another or change from time to time. This type of variability is called consumer-introduced variability (Kannan & Proenca, 2008).

Variability can also manifest itself in the execution environments of SOA-based systems, because of the available variations in service middleware environments, operating systems, and implementation languages. According to Cohen and Krut (2010) and Sun *et al.* (2010), variability in SOA is driven by the continuing advancement in Internet technologies, market situations, new marketing strategies, and other factors related to business. Therefore, it is very important to systematically identify and manage variability in order to realize variability-aware service-oriented systems. As mentioned in Chapter 1, the work in this dissertation focuses mainly on managing consumer-introduced variability, with the view to increase service applicability and efficiency in service consumption.

2.5. Software Product Line Engineering

In the previous section, a number of key SOA challenges were identified and described. The challenge and/or focal point of the research presented in this dissertation was then pointed-out

(i.e., variability management). This section takes a deeper look at Software Product Line Engineering (SPLE), where variability management is an extensively researched subject. SPLE is an efficient engineering paradigm for supporting and managing variability (Kim, Lee, & Jang, 2011). The rest of this section is structured as follows. Section 2.5.1 discusses the notion of Software Product Line (SPL), while Section 2.5.2 illustrates the possible functions of Model-Driven Engineering (MDE) in SPL engineering. The concepts related to variability management are presented in Section 2.5.3. Section 2.5.4 discusses feature modeling as a successful and vital technique for identifying and representing variability.

2.5.1. Software Product Line

In contrast to the traditional focus of software engineering paradigms, on developing individual software products (i.e., one software *application*³ at a time), Software Product Line (SPL) focuses on the development of multiple related software products (Bosch, 2009; Pohl, Bockle, & Linden, 2005). SPL enables the systematic reuse of software assets such as features during the process of developing software applications, which aims at decreasing development and maintenance costs and time, as well as increased quality and productivity. The main goal of this engineering paradigm is to address issues related to designing and developing software for reuse, variability management and/or mass-customization (Kang, Lee, & Donohoe, 2002; Pohl Bockle, & Linden, 2005).

³ The terms “application” and “product” refer to the outcome of a product line or customization process, and are used interchangeably in this dissertation.

SPL focuses on the means of efficiently producing and maintaining multiple related software products, exploiting what they have in common and managing what varies among them. Hence, the concept of variability is a central idea of systematic reuse in SPL engineering (Pohl, Bockle, & Linden, 2005). Variability enables software applications to be efficiently extended, changed, configured or customized for use in a particular context (Svahnberg, van Gurp, & Bosch, 2005). The identification and management of variability are the key concerns with identifying reusable assets (Kang, Lee, & Donohoe, 2002), and representing variability to achieve further software reuse, adaptation, and customization. In SPL engineering, related variants of software products are developed in a systematic and coordinated way, providing tailor-made solutions for different customers. Instead of independently developing each variant from scratch, commonalities are conceived only once (see Figure 2.2).

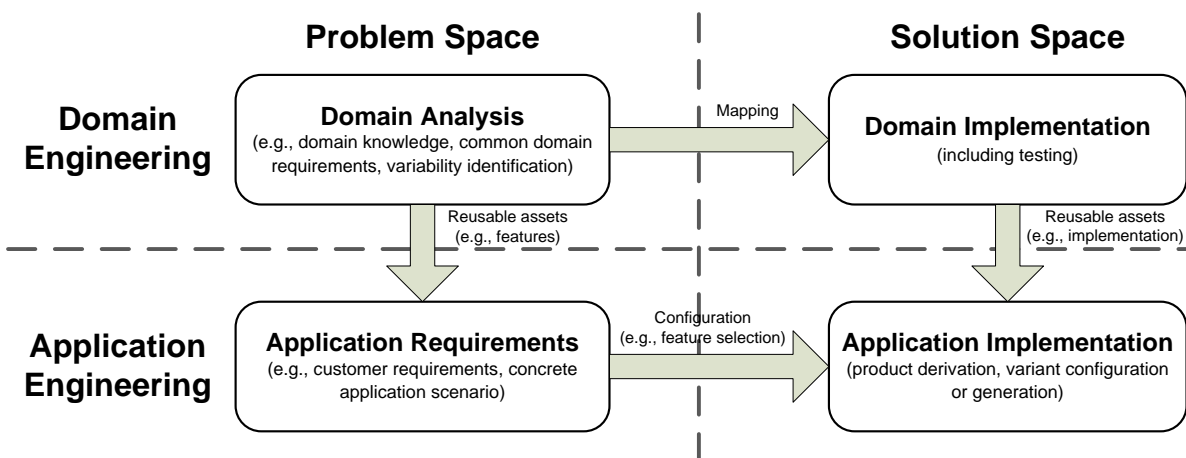


Figure 2.2: SPL Engineering Overview (adapted from (Pohl, Bockle, & Linden (2005))

The **domain engineering** phase concentrates on development *for reuse*. It focuses on analyzing the entire targeted market (i.e., business area) and its potential requirements. Specifically, the goal is to determine common and variable requirements between software products and identify reusable assets. The potential reusable assets are factored-out so that their re (use) is facilitated.

In principle, domain engineering starts with domain analysis, in which the commonalities and differences between potential product variants are determined and described, (e.g., in terms of features). Feature models, as presented later in Section 2.5.3, are known as an effective method for abstracting variability and configuration options in terms of features in the target market (Czarnecki *et al.*, 2012; Lee & Muthig, 2006). Subsequently, product developers or providers design and develop the SPL in such a way that different variants can be constructed from common and variable parts. **Application engineering** focuses on development *with reuse*. In particular, tailored applications are developed, configured or customized based on the common and reusable assets created in the domain engineering phase. This process of building specific applications using a base set of common and reusable assets is usually called product derivation (Deelstra, Sinnema, & Bosch, 2005). Depending on the form of implementation, there can be different automation levels of the application engineering process, from manual development effort to more advanced technology including automated variant configuration and generation.

The terms **problem space** and **solution space** (or problem analysis and solution design), as also shown in Figure 2.2, are normally employed to illustrate the distinction between the problem or system under study and its application domain (Genova, Valiente, & Marrero, 2009). In the SPL domain engineering phase, the commonalities and variabilities of potential products can be identified and modeled for both problem space and solution space (Kang & Lee, 2013). The problem space comprises domain-specific concepts that describe the requirements of software products and the intended behavior (i.e., the customers' objectives and/or requirements, and the situations in which the software products are used). Requirements are modeled to capture and express the high-level abstraction of the functionalities that need to be addressed by the product line. As stated earlier, the results of this process are usually documented in terms of features.

Solution space comprises implementation-oriented concepts describing how the customers' requirements are fulfilled and how the intended behavior is implemented. This might include designing and specifying features as a set of functional and non-functional properties at different level of abstraction. In SPLE, the concept of features or feature models may serve as a means of domain modeling, variability modeling and management, encapsulating functionality or product requirements, decision-making, and communication among application or system stakeholders (Sochos & Riebisch, 2004). The following section discusses possible roles of Model-Driven Engineering (MDE) in SPL engineering processes.

2.5.2. Model-Driven Development in SPLE

Abstraction and model: according to Kramer (2007), abstraction is “the mapping from one representation of a problem to another which preserves certain desirable properties and which reduces complexity”. In other words, the purpose of abstraction is to decrease the number of elements and focus on the important points rather than details. Approaches centered on the use of models (e.g., model-driven engineering, aspect-oriented modeling, generative programming) have been proposed in order to effectively represent, define, and use abstractions for any part of a software system.

Models have been used for many years in various research disciplines including mathematical sciences, biology, economics, house building, and geography (Bezivin, 2005). In the context of software engineering, several definitions of the notion of model have been given, for example, nine definitions are reported in Muller *et al.* (2012), but, in essence, a model is an abstraction of some aspect of a system under study. A system under study can be an actual existing system, or a system under development. A model is an abstraction since some details are hidden and/or

removed to simplify and focus attention (Selic, 2003). A model is an abstraction because general concepts can be created by abstracting common properties of instances or by extracting common features from specific examples. Models are created to serve particular purposes, for example, to present a human-relevant and understandable description of some aspects and properties of a system. Moreover, models can be processed by computer-based tools in order to derive other useful models or some artifacts (such as test cases, performance profiles, or documentation) composing a real-life software system (Czarnecki & Helsen, 2006). This process is called model transformation (France & Rumpe, 2007). Numerous approaches and techniques have been developed for supporting model transformations (e.g., see Czarnecki & Helsen (2006) for an overview of model transformation approaches).

Model-Driven Development (MDD) and other related paradigms are not primarily targeted at the development of SPLs. However, SPL engineering can benefit from models and transformations. Models such as feature models, as mentioned earlier, can be used both in the problem space and solution space to represent different aspects of an SPL and provide domain-specific abstractions. Model transformations can then be performed to generate lower-level models, and eventually program code, from higher-level models (Czarnecki & Helsen, 2006). In SPLE, MDD is gaining more attention as a provider of tools and techniques that can be used to manage the complexity of SPL development. For example, Generative Software Development (GSD) has been proposed to model and implement system families in such a way that a given system or product can be automatically generated from a specification or model written in one or more textual or graphical domain-specific languages (Czarnecki & Eisenecker, 2000).

2.5.3. Variability Management

Variability management is one of the fundamental concepts in SPLE as the main purpose of SPL is to support the development of related software product variants, by taking into account the commonalities and variabilities extracted from the targeted domain. Variability management is described as a set of activities for explicitly representing variability in software artifacts throughout the lifecycle, managing dependencies among different variabilities, and supporting configuration or instantiation of the variabilities (Schmid & John, 2004). This definition points out two important concepts related to variability management: (i) variability modeling and (ii) variability instantiation (J. Lee & Muthig, 2006).

Variability modeling: encompasses the explicit representation of variability and dependency description (Bachmann *et al.*, 2004). The main focus of variability representation is to identify and introduce variation points and variants in domain artifacts. A variation point is a location or a point where differences (variations) in software artifacts might exist and/or occur. Variation points are the source of variants existent in the targeted market or in a domain to be satisfied (Linden, Schmid, & Rommes, 2007; Pohl, Bockle, & Linden, 2005). Dependency description provides constraints on how variants are bound and connected to variation points, and aid in generating valid products (Sinnema *et al.*, 2006). For instance, the binding of variant X_1 to variation point X might require the binding of variant Y_1 to variation point Y, if there exists a relation “X *requires* Y”, and vice-versa: if there exists a relation “X *excludes* Y”.

Variability constraints or dependencies can be divided into two sets, structural interdependencies (Jarzabek, Yang, & Yoeun, 2006) and implicit interdependencies also referred to as cross-tree constraints (Fernandez-Amoros, Gil, & Somolinos, 2009; Lee & Kang, 2004). While the latter

are visible and of much interest to customers, as they provide guidance for variants or feature selection during application engineering, the former are obscure in nature and are mostly of much concern to providers only, to fully support and realize application variability. Pohl, Bockle, and Linden (2005) define the distinction between these two groups as external variability and internal variability respectively. There exist several popular techniques for identifying and representing variability, such as decision modeling (Dhungana, Grunbacher, & Rabiser, 2011), UML-based approaches (Oliveira *et al.*, 2005), goal modeling (Semmak, Laleau, & Gnaho, 2009), Domain-Specific Languages (DSL) (Voelter, 2009), and feature modeling (Shaker, 2010). However, to efficiently describe and represent variability in domain artifacts, modern approaches use feature modeling (discussed later in Section 2.5.4) as a crucial concept for variability representation. According to Shaker (2010), one of the advantages of using feature-based modeling concepts is that features can be easily understood by stakeholders from different backgrounds.

Variability instantiation: following the domain analysis process, and when variability is fully captured in domain artifacts (e.g., in feature models), variability instantiation involves deriving a software product based on customer decision-choices, using configuration tools such as model transformation tools (Perrouin *et al.*, 2008). The goal of variability instantiation is to produce a product that satisfies the requirements of a specific customer or a market segment (Svahnberg, Gorp, & Bosch, 2005). Variability in software systems can be instantiated or implemented at different stages, e.g., during the design of software products (Svahnberg, Gorp, & Bosch, 2005) or during the execution of a software system (Bosch & Capilla, 2012; Capilla & Bosch, 2011). In the context of SOA-based systems and the work presented in this dissertation, variability instantiation is only considered at execution or run time. This is due to the fact that software

services are usually developed for unknown consumers and/or because of the loosely-coupled nature of service providers and consumers. According to Sinnema *et al.* (2006), an effective and efficient means of instantiating variability is to describe variability at the highest level of abstraction, and propagate decision-choices toward lower levels of abstraction. This helps software developers to deal with fewer variation points and variants.

2.5.4. Feature Modeling

The notion of feature modeling was introduced by Kang *et al.* (1990) in the Feature-Oriented Domain Analysis (FODA) method and generated a lot of attention in SPL engineering. Feature modeling makes use of the concept of features as they are externally visible characteristics that can be used to differentiate one product from the next (Kang, Lee, & Donohoe, 2002). The term ‘feature’ is defined as a distinctive user-visible characteristic of a software product. Lee, K., Kang, and Lee, J. (2002) define feature modeling as the activity of identifying features of the family of software products in the targeted domain and organizing them into a model called a feature model. Feature models help in abstracting commonalities and differences in software products derived from the targeted market and/or stakeholders’ requirements. Antkiewicz and Czarnecki (2004) describe this process as domain scoping, and according to Fernandez-Amoros, Gil, and Somolinos (2009), a poorly scoped product domain may lead to inefficient development and maintenance costs (i.e., if the most relevant requirements as derivation points for features may not be realized or if some of the implemented ones may never be reused).

As the key elements for identifying and managing variability, feature models consist of both formal and graphical representation and encompass variability relations as well as constraints or dependencies defined over the product features (Lee & Muthig, 2006). There are different

notations in the literature that extend FODA for expressing the relations between features in a feature model, such as Feature-Oriented Reuse Method (FORM) (Kang *et al.*, 1998), Feature-Oriented Product Line Engineering (FOPLE) (Kang, Lee, & Donohoe, 2002), and Cardinality-Based Feature Modelling (CBFM) (Czarnecki, Helsen, & Eisenecker, 2005). However, the most comprehensive and widely used notation is the CBFM notation. In CBFM, a feature model (see Figure 2.3) is described as a tree-like or hierarchical structure, where features are represented in terms of consist-of relations as well as cardinalities annotated against the features of the feature model. A feature cardinality describes an interval $[m,n]$, where m and n represent the minimum and maximum number of sub-features to be selected within a single parent feature. For example, the cardinality $[1,1]$ in Figure 2.3 for the feature $F3_{A2}$ implies that this feature must be selected when its parent feature $F3_A$ is selected. Furthermore, the cardinality $[0,1]$ for the feature $F3_{A1}$ means that this feature might be selected if its parent feature is also selected.

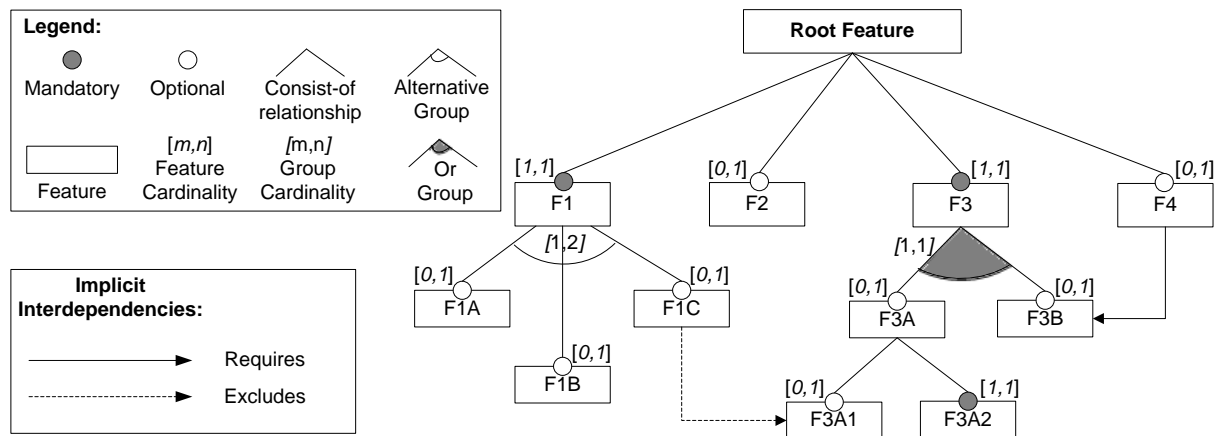


Figure 2.3: A Sample Feature Model Diagram

Stemming from the FODA method, features can be classified into four categories: mandatory features, optional features, alternative-group features, and or-group features. Mandatory features (i.e., common features) are the features that must appear in all product instances or customers'

decision-choices. These features are represented by means of grey circles in the sample feature model above, with cardinality $[1,1]$. Optional and variable features only appear in one or more customer product choices and are indicated with white circles, and with cardinality $[0,1]$. The alternative-group feature indicates that if the parent feature is selected, only x out of n features can be selected in the group. For example, the features $F1_A$ to $F1_C$ as illustrated in the feature model form an alternative-group features; and at most two $[1,2]$ of these features can be selected when the parent feature $F1$ is selected. Feature $F3$ represents the or-group feature with $F3_A$ and $F3_B$ as its sub-features. In this group, one and only one feature must be selected when the parent feature is selected. As was mentioned previously (cf. Section 2.5.2), implicit interdependencies (i.e., *includes* and *excludes*) can be introduced to describe dependencies that might exist among the features of a feature model. In the sample feature model (Figure 2.3), the selection of feature $F4$ requires the inclusion of feature $F3_B$. On the contrary, the selection of feature $F1_C$ imposes the exclusion of $F3_{A1}$.

Briefly, feature models are an efficient and effective technique for abstracting product variability drawn from the targeted business domain or stakeholders' requirements (Czarnecki *et al.*, 2012). They also provide or serve as a means for communicating product commonalities and differences between software application developers and customers, to assist software development (Trujillo, Batory, & Diaz, 2007), and promote customization (Cohen & Krut, 2010). As a consequence, they play a significant role in abstracting and managing variability as well as supporting software development (Heidenreich *et al.*, 2010; J. Lee & Muthig, 2006).

2.6. Overview of Related Work

The foregoing section (Section 2.5) gave a general literature on how variability is dealt with in SPLE, the efficient engineering paradigm for supporting and managing variability. This section drills down to the relevant existing works that are closely related and which inspired the work presented in this dissertation. These efforts are presented in two different parts. The first part focuses on existing service customization efforts, while the second part discusses the literature on service variability modeling.

Service Customization: as described in Chapter 1, is a process whereby service consumers perform a set of operations, such as modifying service description documents in order to adapt a software service to their specific usage or application scenarios. To enable efficient provisioning and delivery of customizable services, a number of key aspects need to be taken into account. First, since service customization allows and/or requires service consumers to execute a set of operations in order to derive specific service variants meeting their feature requirements and business needs, the service customization method needs to be simplified and consumer friendly. In particular, service capabilities (i.e., common and variable properties) should be communicated at the highest level of granularity or abstraction, to enable consumers to easily interpret and make business-sense decisions (cf. Section 2.2.2). Another important aspect to consider is the fact that service capabilities might have interdependencies among each other. That is, one service capability might require the existence of a certain service capability within the communicated and defined capability space (cf. Section 2.5.3). Thus, service customization requests need to be automatically validated to avoid invalid requests, and wastage of computing time and resources.

Liang *et al.* (2006) described an approach which leverages on existing web service standards for supporting service customization. Basically, these authors extended the WS-Policy framework for expressing customization policies, which are then used by service consumers to construct customization requests. According to the information contained in consumers' customization requests, a customized service is developed and deployed by service providers. Although this approach guarantees valid service customizations, it creates an unwanted burden for service consumers, because they have to perform a lot of tasks when they are preparing customization requests, that is, they need to think about customization policies as well as service interface descriptions. As a result, this approach does not satisfactorily fulfill the need for simplification of customization processes. Moreover, this approach assumes that service consumers have the knowledge of how and where to express and communicate customization information. Nguyen and Colman (2010) proposed an SPL-based approach to customize Web service interfaces. The authors use feature modeling to describe variability and variation points existing in the service interface (e.g., operations and message types). Stollberg and Muth (2009; 2010) and Barros *et al.* (2011) also adopted variability modeling concepts from SPLE in order to support service customization. However, these authors focused only on handling variability at technical level or low-level of abstraction, leaving-out variability at higher level of abstractions (i.e., requirements or feature level). Moreover, these authors' approach does not take into account adapting internal service implementations. Therefore, their approach permits running redundant service variants, which increases development time and consumption of computing resources.

Most importantly, since variability is only represented at low-levels of abstraction (e.g., in the aforementioned approaches), which includes a large number of variation points and variants, customizing a service in these approaches necessitates the understanding of technical details and

can be a very complex and error-prone task for service consumers. This can also create a lot of work for service providers, because of the amount of variation points to be handled.

Service variability modeling: the research works in this group focuses mainly on variability modeling techniques to identify and represent variability in service-oriented systems. Although this group does not describe mechanisms or techniques for enabling service customization, they are related to the work presented in this dissertation in terms of variability modeling methods (i.e., the methods for identifying and representing variation points and variants). Acher *et al.* (2010) proposed a feature-oriented method to model functional variability in service workflows. In their method, three feature models are constructed for each service. These feature models are used to represent functional variability at different levels (i.e., service interfaces, service inputs, and service outputs). Medeiros, Almeida, and Meira (2010), also motivated by the SPL concepts, describe a feature modeling method to represent functional and non-functional (QoS) variability in business processes. Chang and Kim (2007), and Kim and Doh (2008) also modeled variability at business process level (i.e., technical level) using SPL concepts.

In all of the research efforts reported in the first group, the researcher found that although the authors try to address the issue of consumer-introduced variability, they only focus on handling consumers varying functional requirements; they do not consider managing the non-functional requirements. However, in practice or in today's service-driven business environments, service consumers usually exhibit varying functional and non-functional requirements. For this reason, it is very important to ensure that their varying non-functional requirements are also considered, because non-functional properties such as security give consumers assurance and confidence that they are using the best software service. The reviewed efforts also do not consider managing the

life-cycle of service variants, thus, they do not reuse service capabilities and hence, reusability is not promoted in these works. In modern service-oriented business environments, the demanding requirements of service consumers increase the cost and time involved in building or generating variants of a service, thus, it is important to reuse service capabilities in order to support different usage scenarios. Moreover, in both groups (i.e., service customization and variability modeling) the authors represent variability at low-levels of abstraction, which might be very difficult for service consumers to interpret and understand. Hence, the techniques presented in the afore-discussed research works do not address the issue of simplifying and minimizing complexity in the service customization processes.

2.7. Summary

In this chapter, the necessary setting and preliminary review of related literature to understand the work presented in this dissertation was given. First, an overview of the SOA paradigm and its underlying components, as well as the Web services technology was presented. Then, the notion of GUISET was introduced. The key challenges facing SOA and GUISET, as one of the existing service-oriented infrastructures, were identified and described. These challenges include variability management, which is the main subject of this research and widely studied in SPLE. Hence, SPLE and its fundamental concepts were also introduced.

Finally, existing research efforts on service customization were presented and analyzed. Several research efforts relating to service variability modeling were also explored. Based on the analysis of all the relevant research efforts, a number of shortcomings were identified. For example, none of the works cover the aspect or concern of simplifying and reducing complexity in providing an effective and efficient service customization approach. The work in this dissertation argues that

this concern can be dealt with by capturing and describing variability at requirements (feature) level, and enabling or supporting service consumers to customize software services at that level. Another drawback is that all the reviewed efforts focus only on handling varying functional requirements, not taking into account that consumers usually do not only have varying functional requirements, their non-functional requirements also vary. Hence it is imperative that their non-functional requirements are also considered.

In brief, the main aim of this research was to create an approach that avoids the aforementioned shortcomings. The approach is outlined in the next chapter. It is aligned with several techniques and methods that are described in the preceding sections of this chapter, in particular, the feature modeling technique.

DESIGN OF THE MODEL-BASED SERVICE CUSTOMIZATION FRAMEWORK

3.1. Introduction

As mentioned in the previous chapters, Service-Oriented Architecture (SOA) enables the design and implementation of a Software-as-a-Service (SaaS) business model, where organizations such as Small, Medium, and Micro Enterprises (SMMEs) can acquire software services embodied and maintained by service providers, through the Internet, without having to purchase their own ICT infrastructures. Even though consumers (e.g., SMMEs) can have access to software services and utilize them in their business processes, in reality, they often have different business and varying requirements. Consequently, service providers are faced with various challenges to stay relevant in today's global economy. This, together with new market opportunities requires mechanisms for rapid development and delivery of software services that best meet consumers' demands. Hence, some providers have motivated the acceptance of customizable software-development methods to build and deploy customized services for their potential consumers.

Nevertheless, as also stated in Chapter 2, in order for service providers to systematically provide flexible and efficient software services, they need to consider both the technological knowledge as well as variable functional and non-functional requirements of (potential) service consumers. This chapter introduces the proposed **FreeCust (Feature Model-based Service Customization)** approach as a solution for managing consumer-introduced variability in SOA environments. The remainder of this chapter is structured as follows. Section 3.2 gives an overview of the design principles for an efficient service customization solution. Section 3.3 introduces the FreeCust

approach. It describes the architecture of FreeCust and its components, as well as the details of how non-functional properties can be incorporated and represented in a feature model. Section 3.4 takes FreeCust into the real-world by demonstrating its applicability using two application scenarios, while Section 3.5 concludes the chapter.

3.2. Design Criteria

Stollberg and Muth (2009) remarked that the process of customizing a service is a nontrivial task that necessitates technical knowledge as well as business expertise. As a result, most business organizations usually appoint experts to carry out this task. For this reason, and the fact that service providers would need to fully exploit economies of scale, lower service development costs and promote service reusability, a complete service customization solution needs to tackle and/or cover problems coming from both the service provider and consumer perspectives. That is: (i) how should service variability (i.e., customization options) be captured, represented, and communicated, and (ii) how to facilitate customizations at the service consumer's side. More to the point, in order to effectively design and develop an efficient service customization solution, the following key design principles gleaned from the literature, (which also served as the basis for the formulation of the proposed FreeCust approach), need to be considered:

- Because of the fact that software services usually have and/or support an excess number of possible customization options, with numerous of interdependencies scattered among those options, which in turn creates a burden and becomes a complex task for consumers to perform service customizations, a successful and efficient service customization solution should, therefore, consider simplifying and minimizing such *complexities*.

- Due to the excessive number of possible customization options and the great number of interdependencies across those options, as explained above, customizing a service can be a very error-prone process. Thus, a mechanism for *validating* consumers' customization requests should be provided by the service customization solution. This is to ensure that service consumers do not violate any properties described by providers, and to minimize wastage of computing time and resources.
- Since service consumers exhibit different or varying requirements, which (may) lead to different customization requests, different service implementations may also exist. Thus, to minimize development costs, and to ensure that computing resources and development efforts are not wasted, running redundant service variants should be avoided. Therefore, a mechanism for *deploying customized services dynamically* should be provided by the customization solution.

3.3. The FreeCust Approach

This section presents **FreeCust**, a **F**eature **M**odel-based **S**ervice **C**ustomization approach which was designed according to the abovementioned design criteria. The section starts by describing the FreeCust framework architecture and the interactions among its components. Subsequently, the details of how to represent and incorporate non-functional properties into software product models (i.e., feature models) are presented.

3.3.1. Framework Architecture

The feature model-based service customization framework (FreeCust) enables the provisioning and delivery of consumer variability-aware (customizable) software services. In this framework,

the service customization process employs the Model Driven Development (MDD)⁴ approach to automate almost all the parts and/or operations of the framework. Figure 3.1 illustrates the overall architecture of the framework. It consists of five major components: a service provider, service registry, service consumers, feature management engine, and variants repository. These components interact with one another to accomplish the service customization task and deliver consumer variability-aware software products. As mentioned in Section 3.2, one fundamental requirement for an efficient service customization solution is the ability to capture, represent, and communicate customization options, together with their interdependencies to service consumers.

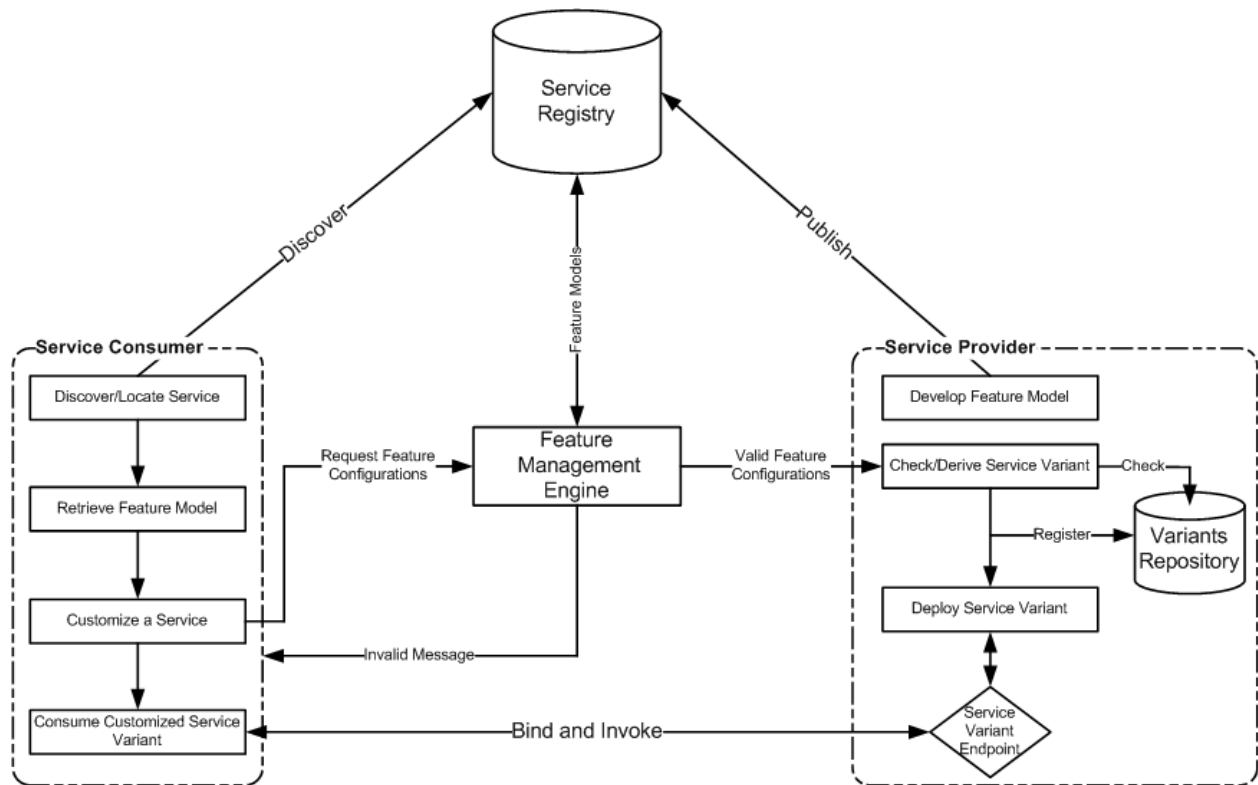


Figure 3.1: FreeCust Framework Architecture (Dlamini, Tarwireyi, & Adigun, 2013)

⁴ The goal of MDD is to increase automation, simplify the process of design, increase re-usability and productivity by maximizing compatibility between systems (France & Rumpe, 2007).

In addition, the service customization process should be consumer friendly, in such a way that consumers can easily interpret service variabilities to make business-sense decision choices. As mentioned in Chapter 2, this aspect can be addressed by capturing and describing variability at the highest level of abstraction (i.e., requirements or feature level), and supporting consumers to customize software services at that level. For these reasons, the FreeCust solution follows the MDD approach. MDD is considered as efficient and exhaustive when it comes to developing software systems, something which is also true for service-based systems (Srinivasan, Paolucci, & Sycara, 2006). Models (in MDD) are important for code generation, due to the different levels of abstraction (Nassar *et al.*, 2009), thus simplifying and minimizing complexity, as well as maximizing service development productivity.

The FreeCust approach, as illustrated in Figure 3.1, is particularly guided by feature modeling, an efficient technique for capturing and describing variability at any level of abstraction, using feature models. Shaker (2010) stated that one advantage of using feature modeling concepts is that features can be easily understood by stakeholders from different backgrounds. The steps and interactions involved in the FreeCust approach for the provisioning of customizable services are described below and illustrated in Figure 3.2.

First, the **service provider** creates and publishes a service description (e.g. a service interface) into the **service registry**. In the context of FreeCust, a service description artifact is considered as a feature model, which incorporates both the functional and non-functional properties of a software service. A feature model is used to guide and ease the customization process, and to capture the core and varying functional and non-functional requirements of service consumers. In addition, a feature model is considered as a description artifact because the service interface

that consumers will use would not be the same. This is due to the fact that service interfaces are allied to business objectives and requirements. Hence, they would be the final product of the service customization process.

The second step involves **service consumers** searching and/or looking for software services of interest in the service registry. Once a service of interest is discovered and located, a service consumer then retrieves a feature model of that particular software service and starts selecting or providing all the necessary functional and non-functional features required in the specialized service. This information is then communicated to the **feature management engine**, which utilizes automated analysis techniques of feature models (Benavides, Segura, & Ruiz-Cortes, 2010; Benavides, 2009), and is responsible for analyzing and checking whether the provided customization information, does not violate the properties described in provider's feature model. If the customization information is valid, it is forwarded to the service provider side, otherwise an invalid message is sent to the service consumer.

Before a service is generated, in case of a valid request, a service provider first checks on the **variants repository** to see if there exists any software service variant matching the consumer's customization information. If there is any and that variant is still running, the service's endpoint would be communicated to the service consumer and the consumer would then bind, invoke and start using the customized service in the application scenario(s) that the service is required. Otherwise, a different service variant would be created and before it is deployed it is registered in the variants repository. The main role of the variants repository is to store and keep track of the existing and deployed service variants.

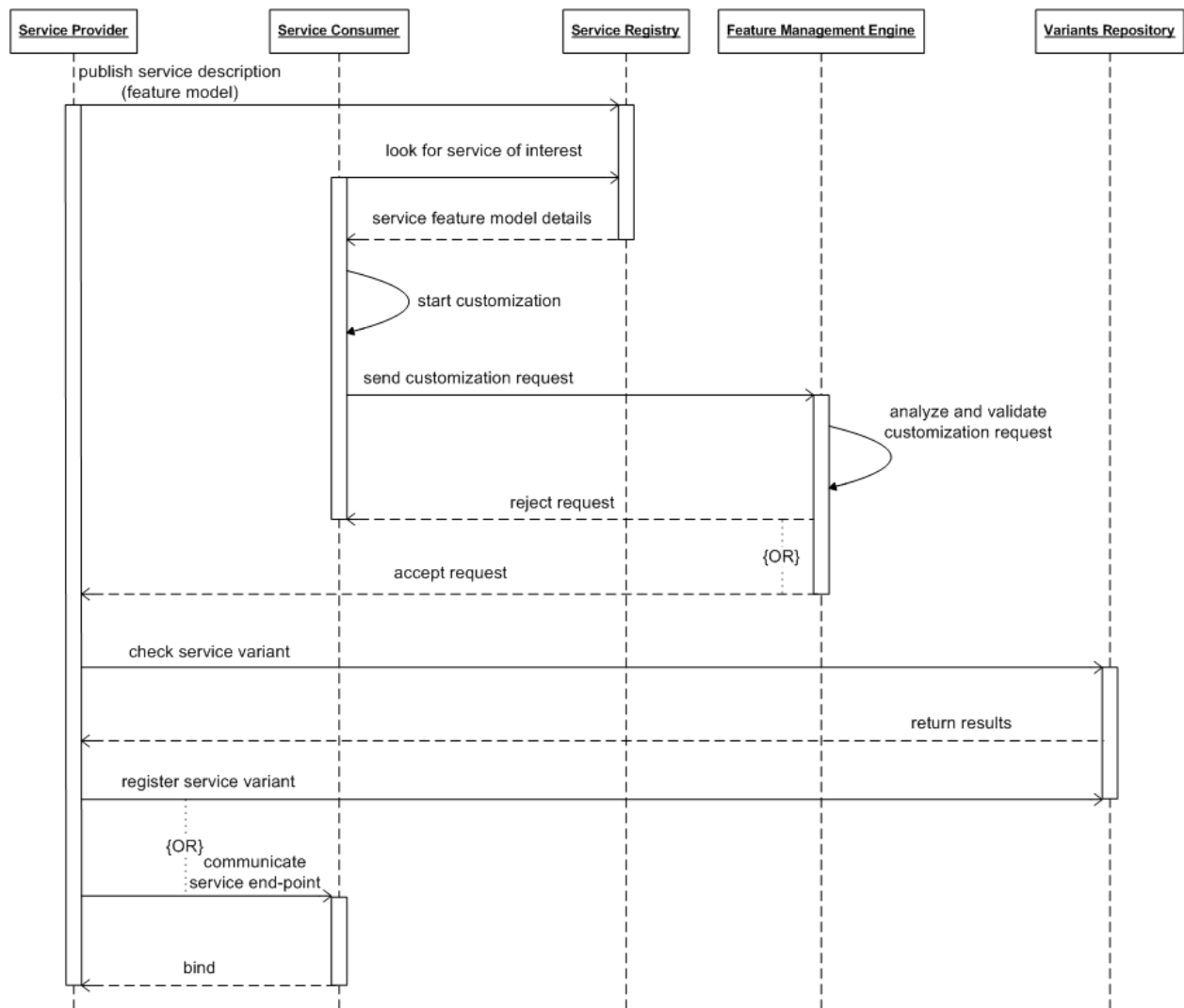


Figure 3.2: Interaction Diagram of the Framework Components (Dlamini, Tarwireyi, & Adigun , 2013)

From the foregoing discussion, it is clear that there is a strong relationship between a product family and a customizable software service. That is, they both represent a set of products that share a common set of properties and each product contains a set of distinct properties, which aid in differentiating one product from the next. Another important relationship is that a product family and a customizable service need to represent variability options to enable the delivery of products tailored to specific consumers' demands. Hence, a feature model was adapted in the

FreeCust approach to represent functional and non-function properties of a customizable service. The following section describes how non-function features can be included in a feature model.

3.3.2. Extended Feature Model

As mentioned in Chapter 2, software services should be developed to satisfy diverse functional and non-functional requirements of service consumers. So far, the main focus of existing service customization approaches is (or has been) on functional variability. Diversity of non-functional requirements has been neglected. However, in order to deliver customizable software services that best fit both functional and non-functional requirements of service consumers, it is very important to capture and represent non-functional properties along with functional properties in the product feature model.

Accordingly, this section introduces the solution proposed in this research for incorporating non-functional properties into software product models. It comprises a Non-Functional Property (NFP) model (see Figure 3.3) and extends the cardinality-based feature model with features containing non-functional properties (see Figure 3.4).

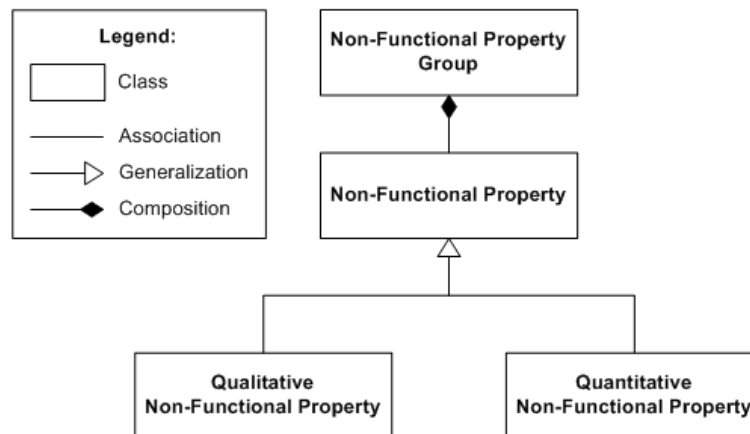


Figure 3.3: Non-Functional Property (NFP) Model

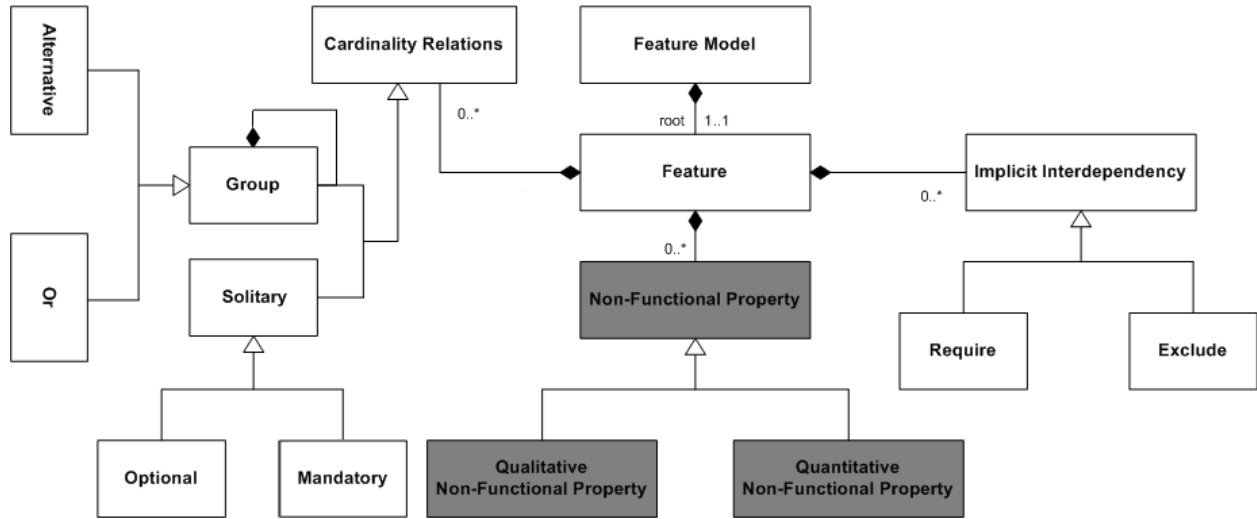


Figure 3.4: Cardinality-Based Feature Model with Non-Functional Properties

Based on the existing research in software engineering, software product line, and requirements engineering, handling non-functional requirements is very challenging (Glinz, 2007; González-Baixauli, Laguna, & Crespo, 2004; Mairiza, Zowghi, & Nurmuliani, 2010). This is because of the complex nature of non-functional properties. However, as mentioned before, non-functional properties are crucial to the success of an efficient service customization solution and/or delivery of effective software services.

In general, non-functional properties are broadly classified into two groups: (i) properties that a software product must exhibit and (ii) properties that describe the quality attributes of a software product (Mairiza, Zowghi, & Nurmuliani, 2010). The first group covers all the non-functional properties and characteristics of a software product, which includes aspects such as development and maintenance cost, availability, reliability, security, and so forth, whereas the second group is a subset of the first group. That is, it only covers a slight range of non-functional properties (i.e., quality of service attributes).

As inspired by the following scholars (Benavides, Trinidad, & Ruiz-Cortes, 2005 and Siegmund *et al.*, 2012), who have extended the notation of feature model with non-functional properties to represent the non-functional aspects of software products. The work in this dissertation has also proposed a method to incorporate non-functional properties in CBFM (Cardinality-Based Feature Modeling) notations (cf. Figures 3.3 and 3.4). The method focuses only on the second group of non-functional properties, i.e., the properties that describe the quality attributes of a software product. This is due to the fact that this group covers almost all the non-functional aspects that can be described and represented in such a way that consumers can define their preferences. For instance, some service consumers may call for high-level of security in their software product, while others, security might not be of that magnitude. As shown in Figure 3.3, the NFP solution further sub-divides the non-functional property group (i.e., quality attributes) into two groups: (i) qualitative non-functional properties and (ii) quantitative non-functional properties.

Quantitative non-functional properties are the non-functional aspects that are measurable and can be represented by a numerical value or interval. Cost, availability, throughput, response time are examples of quantitative non-functional properties. The other group of non-functional properties consists of properties that cannot be exactly measured. These include properties like security and reliability, and can be represented using linguistic labels (Andres, Garcia-Lapresta, & Martinez, 2010) such as high, medium, and low. In the context of the NFP model and FreeCust, where it is necessary for service consumers to specify their decision-choices (i.e., choosing functional and non-functional properties of their specialized software service), the offered and variable options defined over non-functional properties are represented using *linguistic quantifiers*⁵. For instance,

⁵ Linguistic quantifiers are operators that limit the variables of a proposition (Geurts, 2003).

the quantifiers mentioned earlier, i.e., high, medium, and low are used for the security feature. The fundamental reason behind the use of linguistic quantifiers is because of the complex nature of non-functional properties, and that linguistic quantification has been used and is extremely decisive in requirements and knowledge representation (Luisa, Mariangela, & Pierluigi, 2004; Ying, 2006). The following section demonstrates the utility and applicability of the notion of FreeCust and its underlying concepts.

3.4. Example Application Scenarios

In the preceding sections of this chapter, the FreeCust approach and its basic components were introduced. This section, therefore, seeks to illustrate the usefulness and relevance of FreeCust, using two real-life examples: a live match score application scenario inspired and adapted from Schoneveld and Philip (2007), and a purchase order application scenario, one of the widely used research-demonstrating and motivating scenarios in software product line and service computing research.

3.4.1. Live Match Score Application Scenario

The live match score service is a sample software as a service (SaaS) component, which enables consumers to display the most recent score of a particular sport match on their business websites. This service can be exploited by various consumers, such as sport clubs, sports federations, and gambling companies, to display the latest match score in their portals and/or websites. Given the fact that, in real-life, service consumers usually have different business goals and objectives, as well as dissimilar business sizes, their service requirements may not be identical. Take for example two consumers, e.g., a cricket club and a gambling company such as bet365 (one of the world's leading online gambling companies), requiring the live match score service for use in

two special cricket matches. Between these two matches, one match is a domestic league match, for example, Chevrolets Warriors against Queensland, and the club wants to display the real-time score and commentary on their website. The second match is an International Cricket Council (ICC) match, between Australia and South Africa, and bet365 needs to display the score and live commentary for its potential customers.

Certainly, a domestic league match is less important than an international match so during the course of these two matches, more applications and/or visitors will access the bet365 website, as compared to the club's website. For this reason, and the fact that the involved consumers have different business objectives, different feature requirements for the live match score service will exist. Case in point, bet365 may require that the service updates information at the frequency of 60 seconds, while the club may require information to be updated after 20 minutes. Bet365 may also require a 99% to 100% availability of the service, whereas for a club 70 to 80% availability might be sufficient. Availability in this case refers to the degree of availability of the software service relative to a maximum availability of 24 hours, seven days a week. From a security point of view, bet365 is more vulnerable to different kinds of attacks (such as distributed denial-of-service attacks); thus, security can be of great value to them. Finally, due to the fact that the number of visitors may increase rapidly during the international match, bet365 might also require the service to be highly scalable.

As mentioned before, services advertised to a large public could be used by various consumers, with varying feature requirements and business needs. Thus, to facilitate the process of building, deploying and consuming services in such situations, and to meet the specific requirements of

service consumers in a timely and cost-effective manner, it is very important to systematically identify and manage variability.

In applying the FreeCust method, a service provider can create and develop a feature model (as illustrated in Figure 3.5), encapsulating all the commonalities and variable features of the live match score service, and make it available for potential service consumers. From the live match score feature model, service consumers such as the application developers of the sport club and bet365 company, can generate their specialized live match score variant by choosing all the properties and/or features that they require their service to support.

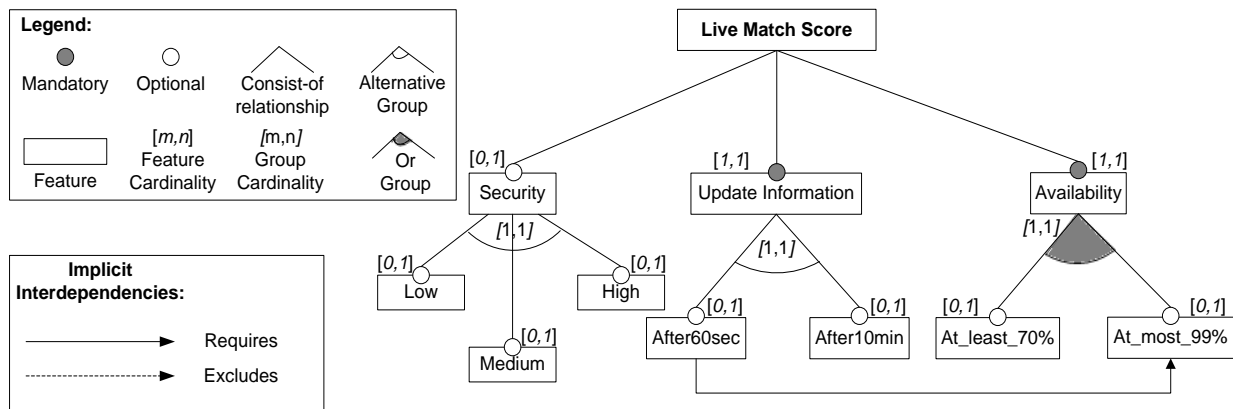


Figure 3.5: A Sample Feature Model for the Live Match Score Service

3.4.2. Purchase Order Application Scenario

To further demonstrate the concept of FreeCust, Figure 3.6 provides a simplified feature model, representing the commonalities and variabilities of the purchase order SaaS. This service can be used by both SMMEs and large organizations to support their on-line shopping and trading business. As was also stated in the previous section, different business sizes imply different business objectives, thus different consumer requirements for the purchase order service would

exist. For instance, depending on the business objectives and strategy of an SMME, one SMME might be focusing on marketing and selling art and craft products, while another is focused on providing bed and breakfast (Gumbo, Terzoli, & Thinyane, 2013; Sibiya *et al.*, 2008). The one that is centered on art and craft production might require a specialized purchase order service that supports both debit and credit card features (see Figure 3.6). While the same SMME may also require the shipment optional feature, the one focusing on bed and breakfast might not be interested in such a kind of feature.

As can also be seen in Figure 3.6, security is mandatory in all instances of the purchase order SaaS. This means, all purchase order service variants must and/or will support the security feature. And, depending on the choices made under the payment method feature, the credit card feature necessitates a higher level of security.

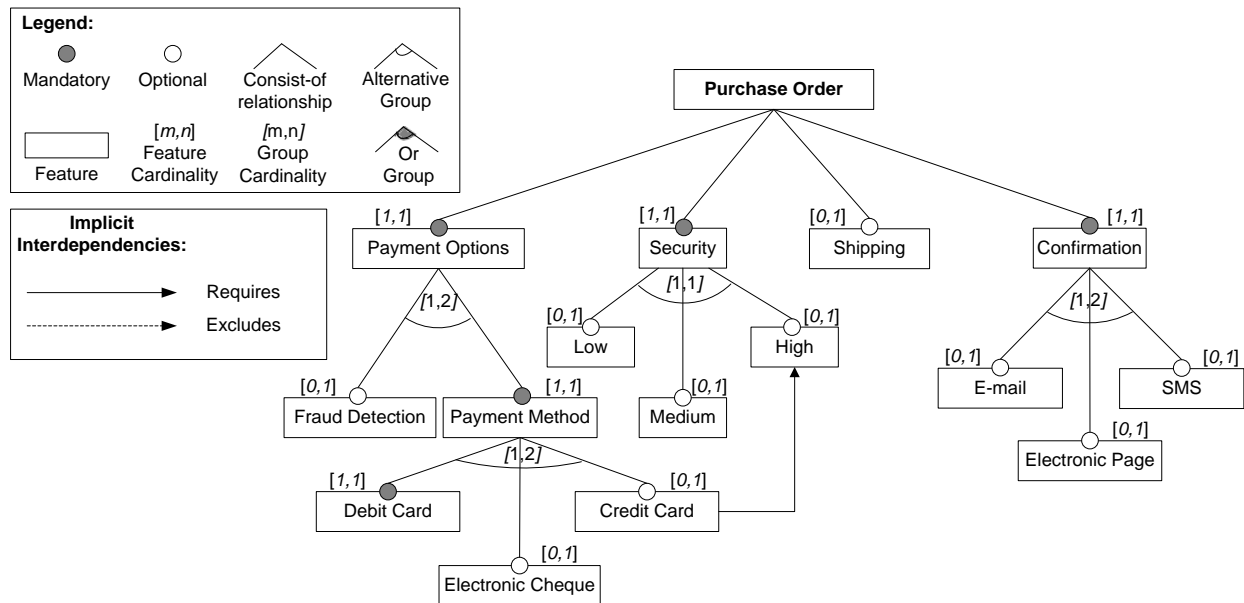


Figure 3.6: A Simplified Purchase Order SaaS Feature Model

3.5. Summary

As mentioned in Chapter 2, in order to effectively deliver customizable software services, that best meet consumers' demands, a service customization mechanism needs to be straightforward and consumer friendly. That is, service capabilities (i.e., the common and variable functional and non-functional properties) should be described and communicated at the highest level of granularity and/or abstraction, to enable service consumers to interpret easily and make business-sense decisions. The research in this dissertation argues that this concern can be dealt with by capturing and describing service variability at feature level, and enabling or supporting service consumers to tailor software services at that level. Hence, this chapter presented the **Feature Model-based Service Customization** approach (**FreeCust**), as put forward by the work in this dissertation. First, the design criteria or requirements that guided the construction of FreeCust were outlined. Then, the FreeCust framework architecture and the functions of its components, as well as a mechanism for incorporating non-functional properties in software product models were described. Finally, the chapter demonstrated the applicability and utility of the FreeCust approach using two real-life scenarios.

IMPLEMENTATION AND EVALUATION

4.1. Introduction

As this research was guided by the design science research methodology, in which one of the crucial research processes is the implementation and evaluation of the design artifact (Elio *et al.*, 2008; Hevner *et al.*, 2004), this chapter aims to provide the technical feasibility and evaluation of the suggested FreeCust design, as presented in the previous chapter. In particular, it focuses on the implementation details and evaluates FreeCust through several controlled experiments (i.e., usability and effectiveness tests). Subsequently, the benefits of FreeCust are represented via a criteria-based comparison against other existing service customization approaches (referred to as *static analysis* in the design evaluation stage of design science research (Hevner *et al.*, 2004)).

This chapter is structured as follows: Section 4.2 presents a high-level overview of the FreeCust prototype implementation and its details. Section 4.3 reports on the experiments conducted to evaluate the FreeCust approach. The results of the experimental evaluations are discussed in Section 4.4. Section 4.5 compares FreeCust with other existing approaches, while Section 4.6 ends the chapter by summarizing and highlighting the significance of FreeCust.

4.2. Implementation Overview

Before exploring the implementation details, this section starts by presenting Unified Modeling Language (UML) diagrams, that is, a use case diagram and an activity diagram. These diagrams are used to give a high-level prototype overview of the FreeCust framework.

4.2.1. FreeCust in UML

In Sections 3.2 and 3.4 of the previous chapter, the design requirements and/or considerations, as well as the applicability and utility of the FreeCust approach were presented respectively. This section, therefore, presents a use case diagram and activity diagram, as the mechanism to provide a high-level overview of the FreeCust framework prototype and, through which the framework prototype can be evaluated. Figure 4.1 represents the FreeCust prototype use case diagram. As shown in the diagram, the FreeCust prototype has two primary actors, i.e., the stakeholders that initiate or trigger a use case process, to achieve a specific goal (Bocchi, Fiadeiro, & Lopes, 2008; Oracle, 2007). These primary actors are the service provider and service consumer. The feature management engine is regarded as a supporting actor. This is because it offers additional functions to support the use cases of the primary actors in achieving their specific goals.

The Primary Actors:

The fundamental goals of the service provider and consumer actors are to respectively provide adaptable and flexible software services, in a rapid and cost-effective manner and to find and utilize an appropriate focused software service. These goals are achieved through the functions illustrated in Figures 4.1 and 4.2. That is, in order for the service provider to efficiently provide a flexible software service, the first action is to develop a software service description (i.e., the service feature model), which enables service consumers to easily interpret service customization options and fashion-out their required software service. Thereafter, publicly market the software service, by publishing its description in the service registry. For the service consumer (e.g., an application developer, a business analyst or a business owner), the initial step is to look for a software service of interest in the service registry and then retrieve the software service feature

model, and perform service customization operations (i.e., selecting and de-selecting the required and unwanted features), and then finally to send the customization request.

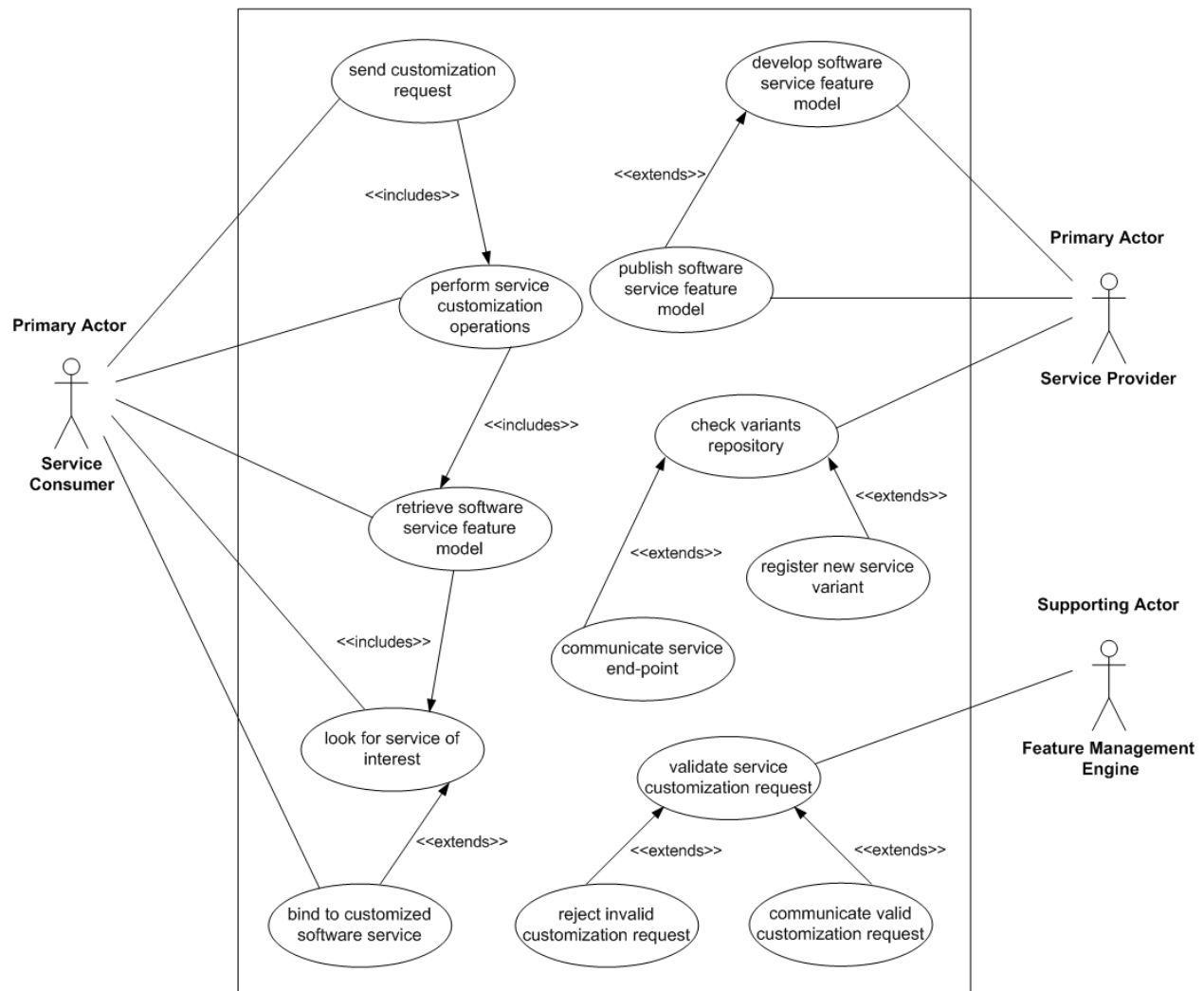


Figure 4.1: FreeCust Use Case Diagram

The Supporting Actors:

The feature management engine acts as a supporting actor. In particular, it employs automated analysis techniques of feature models (Benavides, Segura, & Ruiz-Cortes, 2010; Benavides, 2009) for verifying whether a customization request is valid or not. As shown in Figure 4.2, if a

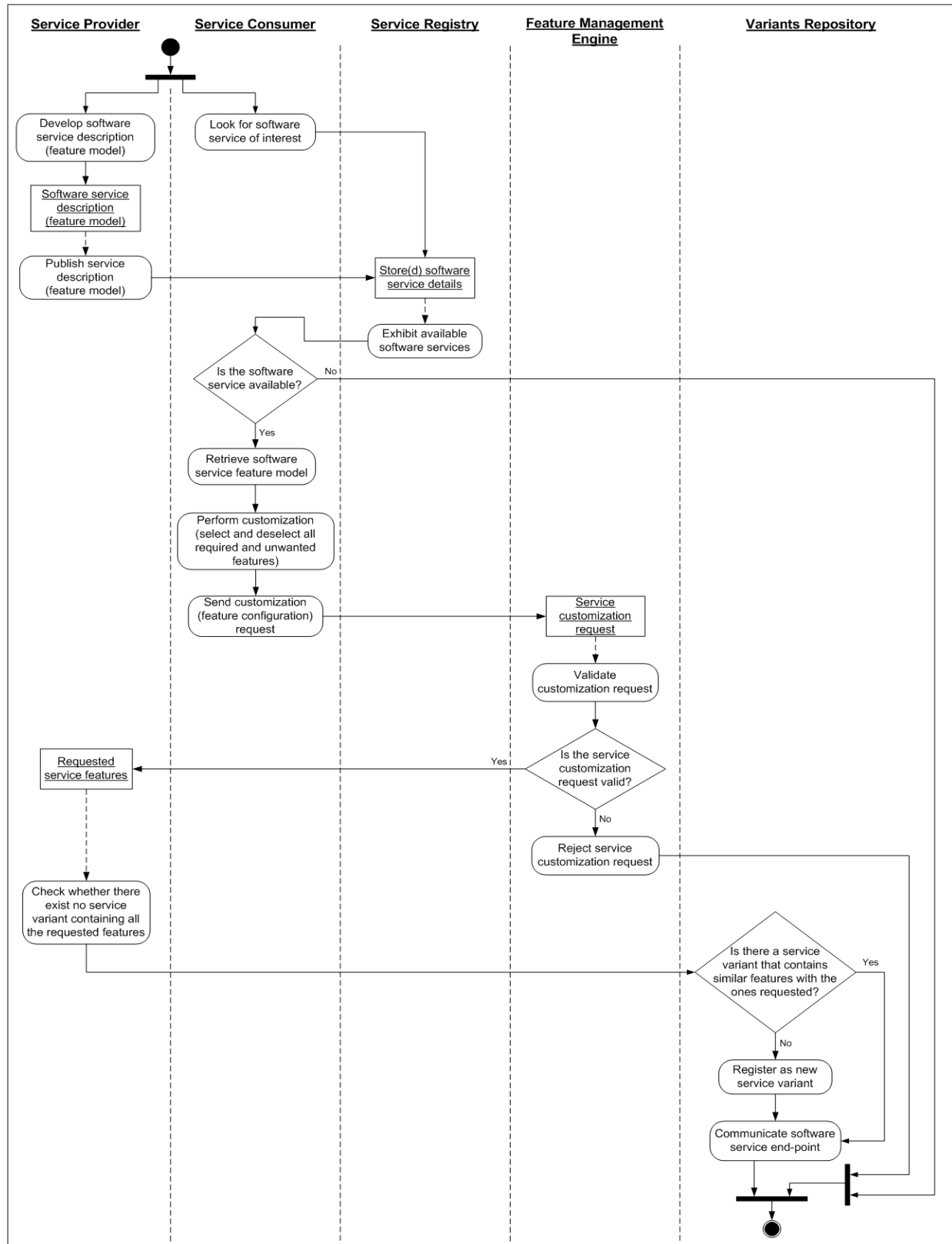


Figure 4.2: FreeCust Activity Diagram

customization request is valid, it is communicated to the service provider's side, otherwise the service customization process is stopped. In the event of a valid request, the service provider checks the variants repository, to find out if there exists no specialized service variant(s) containing similar features with the ones requested. As previously mentioned, this process ensures that running redundant services is avoided, and guarantees that computing time and development efforts are not wasted. The following sub-sections detail how FreeCust was realized, describing the various tools, platforms, and frameworks used in the implementation.

4.2.2. FreeCust Underlying Technologies, Tools, and System Packages

The FreeCust prototype implementation was carried out using Java technologies. Eclipse Juno IDE for Java EE Developers and the Eclipse Modeling Framework (EMF) were chosen as the core development platform. The reason behind this decision was because of the escalating level of support, community involvement, and custom acceptance of Web services and modeling tools in Eclipse (Blewitt, 2012; Skerrett, 2012). Figure 4.3 shows a layered technological view of FreeCust implementation, while Table 4.1 summarizes the tools, platforms, and frameworks exploited in the realization of FreeCust, as well as their roles.

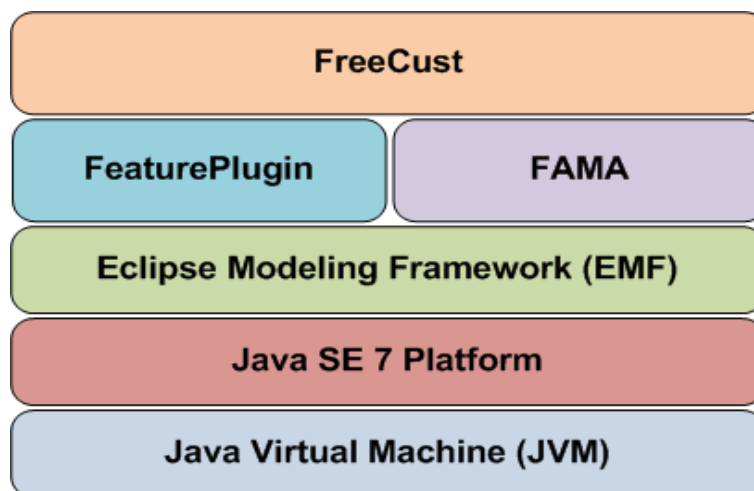


Figure 4.3: High-Level Layered Technological View of FreeCust Implementation

Table 4.1: Summary of Key Platforms and Frameworks Used In the Realization of FreeCust

Platform / Framework	Description	Role in FreeCust
Eclipse Juno	An IDE for Java EE Developers.	Main programming environment for components of FreeCust.
Java SE 7	Java Standard Edition version 1.7.	Java compiler.
EMF	Eclipse Modeling Framework.	Modeling platform used to generate the basic structure of software service capabilities using the generator for tree-oriented model editors.
featurePlugin	An EMF-based open source tool (written in Java programming language) for feature modeling.	Used to implement the graphical user interface flavored mechanism, which enables the development or creation of software service feature models, and facilitates all the operations of service customization.
XMethods	An open platform for advertising and registering software services.	Used as the registry to publish a customizable software service for demonstration and experimentation.
FAMA	FeAture Model Analyzer framework (written in Java programming language).	Used to implement the Feature Management Engine - the request validation component.

The sub-sections that follow (4.2.2.1 to 4.2.2.2) give a brief preview of the critical packages of the tools (i.e., featurePlugin⁶ and FAMA⁷ framework) that implemented the major functions or components of FreeCust, particularly, the Graphical User Interface (GUI) flavored mechanism, which enables and facilitate service customization, and the feature management engine.

⁶ <http://gsd.uwaterloo.ca/projects/fmp-plugin>

⁷ <http://www.isa.us.es/fama>

4.2.2.1. The GUI-flavored Mechanism

The GUI mechanism was implemented through an open-source tool called `featurePlugin`, which is an EMF-based tool for supporting requirements and/or feature modeling. The core packages that implemented this mechanism and its operations are described below and illustrated in Figure 4.4.

ca.uwaterloo.gp.fmp.presentation: This package has classes that implement and present to the service provider and consumer, a model wizard, with a set of commands or operations that can be performed and executed to develop a software service feature model and carry out service customizations, respectively. The classes contained in this package work together with several classes defined and implemented in **ca.uwaterloo.gp.fmp.imp**, **ca.uwaterloo.gp.fmp.system**, and **ca.uwaterloo.gp.fmp.system.drillDown** to create a model and configuration wizard, as well as to handle user interaction in viewing and selecting the desired and unwanted feature configurations.

The utility class called *fmp.system.MetaModel* and a method called *makeProject()* (contained in **ca.uwaterloo.gp.fmp.system**) are used and are responsible for the creation of the contents of an empty service feature model. The operations of creating and adding features (as well as their cross-tree constraints) into the software service feature model are implemented and handled by classes in **ca.uwaterloo.gp.fmp.provider.action** and **ca.uwaterloo.gp.fmp.constraints**. The **ca.uwaterloo.gp.fmp.constraints.ui** package contains classes that implement a window, which informs users (e.g., service consumers) about the existence of feature cross-tree constraints or feature interdependencies, as well as their information. The way in which the GUI-flavored mechanism is used is demonstrated in Section 4.2.3.

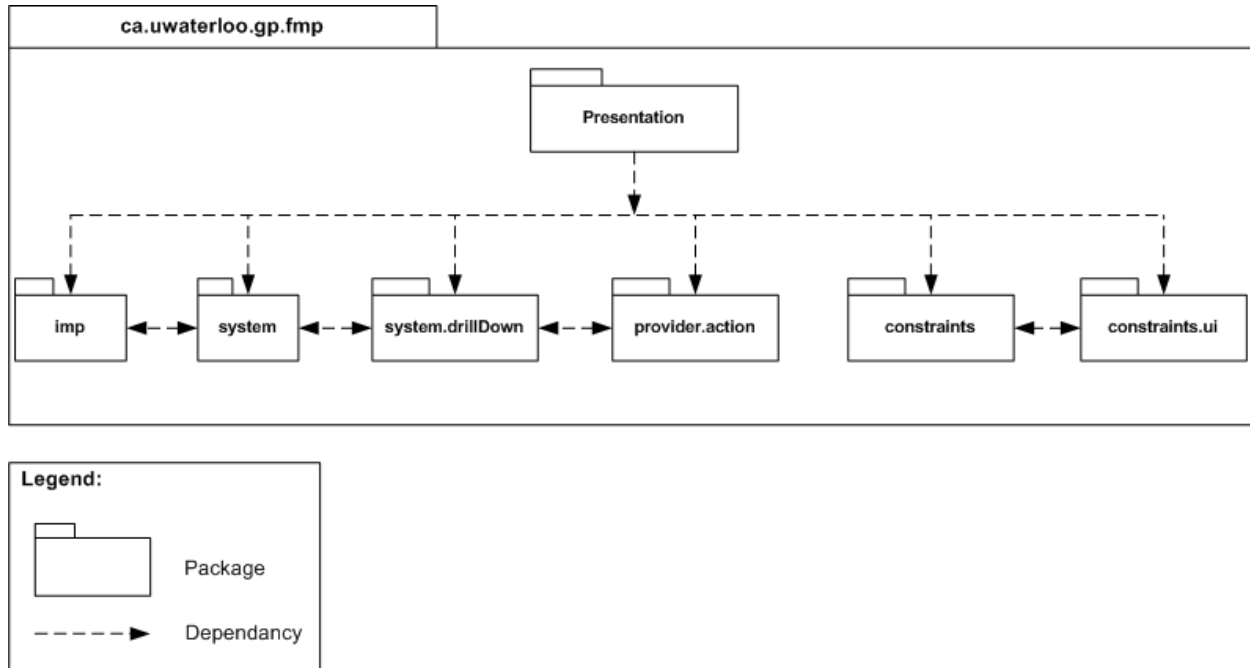


Figure 4.4: The Dependencies among the Core Packages that implemented the GUI-flavored Mechanism

4.2.2.2. The Feature Management Engine

The realization of the component *feature management engine* is based on the FAMA framework. Figure 4.5 illustrates the dependencies among the packages that implemented the main function of the feature management engine, that is, to check the validity of service customization requests.

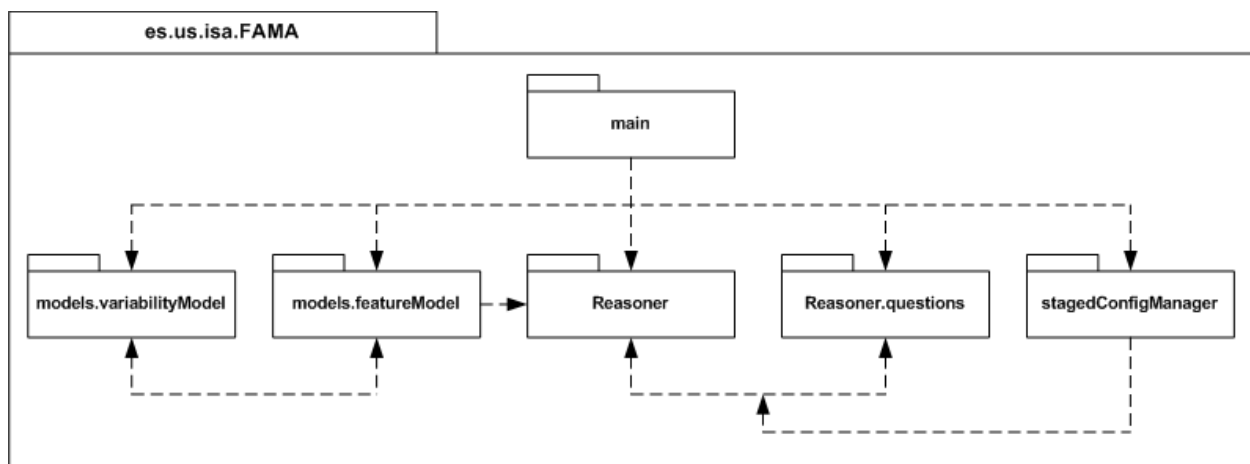


Figure 4.5: The Dependencies among the Core Feature Management Engine Packages

es.us.isa.FAMA.main: This package has a class called *FaMaMain* that starts the execution of the request validation process or the feature management engine. This class is responsible for the initialization process, which runs after receiving a service customization request. During the initialization process, the validation engine accepts and parses the service customization request, in an XML-based format, as well as the specifications of the service provider's feature model, to the process carried out by the classes contained in the following described packages.

es.us.isa.FAMA.models.variabilityModel: This package contains three classes for reading the consumer's requested feature configurations and provider's service feature model. These classes co-operate with several classes defined in the **es.us.isa.FAMA.models.featureModel** package (e.g., *Cardinality*, *Constraint*, and *GenericRelation* classes).

es.us.isa.FAMA.stagedConfigManager: This package has and/or implements a class called *Configuration*, which creates a configuration container with the set of selected and disabled features, from the consumer's feature configuration request.

es.us.isa.FAMA.Reasoner: This package mainly includes classes that implement the feature model reasoners. These reasoners are responsible for determining if the set of requested feature configurations are valid for a given (provider's) service feature model. In particular, the set of requested feature configurations represented by a service feature model are converted into a propositional formula defined over a set of Boolean variables, where each variable corresponds to a feature. From the propositional formula, various reasoning operations are then performed. There are helper classes defined in this package to support the reasoning process. These classes co-operate with classes contained in the **es.us.isa.FAMA.Reasoner.questions** package.

es.us.isa.FAMA.Reasoner.questions: This package contains classes that define various types of questions against the service provider’s feature model and the consumer’s requested feature configurations.

4.2.3. FreeCust Proof-of-Concept Implementation

The previous chapter elaborated the design rationale of the FreeCust approach. It covers the overall architecture of the FreeCust framework, the major constituent elements, and the critical technique for incorporating non-functional features in software product models. Moreover, Sections 4.2.1 and 4.2.2 gave a high-level prototype overview of the FreeCust framework, as well as the key development platform and technologies used to implement the components and functions of FreeCust. This section demonstrates the FreeCust proof-of-concept implementation. The proof-of-concept implementation necessitated a testing environment for testing the functions of FreeCust, in a service-oriented way. Hence, the live match score usage scenario introduced in Section 3.4.1 was reused, as highlighted in the sub-sections that follow.

4.2.3.1. The Development and Discovery of Software Service Feature Models

Figure 4.6 presents a screenshot of how providers prepare software services for customizations, that is, the creation of a software service’s feature model, describing all customization options and their implicit interdependencies (i.e., the *requires* and *excludes* cross-tree constraints). As mentioned in Section 4.2.2.1, this was realized using the open-source tool called featurePlugin. The screenshot illustrates how the service provider of the live match score software service can create a software service feature model and prepare the service for customizations. As revealed by the screenshot, the *requires* and *excludes* cross-tree constraints are defined and represented as conditional formulas among the features of the service feature model. This ensures automatic

selection and de-selection of connected and/or linked features, and enables customizations with minimum efforts, while maintaining the validity of the resulting service customization requests.

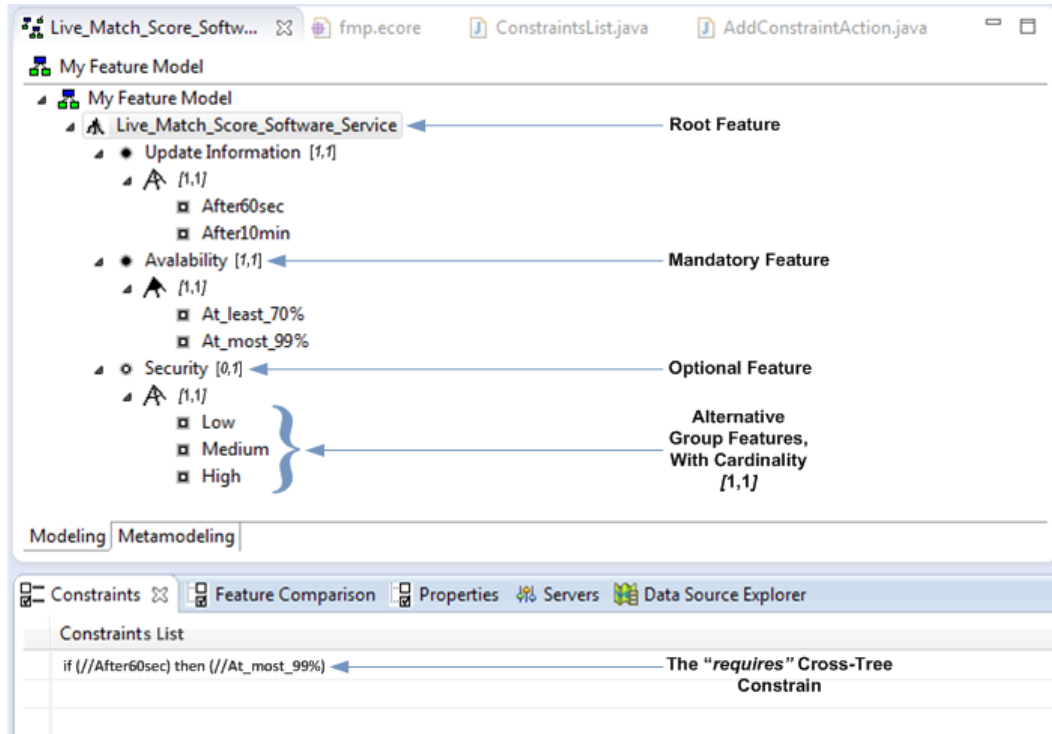


Figure 4.6: Provider’s Live Match Score Service Feature Model in FreeCust

Following the service provider in creating the live match score service feature model, the service needs to be advertised by publishing or registering it into a service registry, to enable potential consumers to discover it and formulate customization requests. This process was realized using the XMethods service directory (see Figure 4.7), which is an open platform for advertising and registering software services. In reality, XMethods is an organization dedicated to supporting and promoting the development, deployment, and use of software services. Besides, XMethods support and promote research activities by providing a way through which service developers and researchers can register their software services as sample services for experimental purposes.

Hence, the live match score software service was registered as a demo service. However, there are terms and conditions for registering and using demo services (see Appendix E).

www.xmethods.net/ve2/Directory.po

XMETHODS

You are logged in as: SWD [Log out](#)

[Home](#) · [Tools](#) · [Implementations](#) · [Manage](#) · [Register](#) · [Tutorials](#) · [About](#)

xignite

Free Trial

Quotes, Forex, Rates, Metals, Financials, International, News, SEC Filings, etc...

#1 in Financial Web Services

MASH'EM UP!

CDYNE CORPORATION

Postal Address Verification Web Service

VALIDATE YOUR WEB USERS IN REAL-TIME

☒ Name

☒ Email

☒ Address

☒ IP

☒ Phone

ServiceObjects

INSIGHT ON DEMAND

STRIKEIRON

Data as a Service. Hundreds of Services to Integrate Now!

FREE TRIAL

Full Service List

Also accessible via XML Interfaces: [DISCO](#) [WS-Inspection](#) [RSS](#)

See the [interfaces section](#) for more information.

List is ordered by submission time, with most recent services listed first.

Publisher	Style	Service Name	Description	Implementation
ecubicle	DOC	Try It Google Search Results in RSS	Provides Google Search Results in RSS Format	
vbfacile	DOC	Try It Prime Numbers Generator	Generates a list of Prime Numbers that are less than a specified max.	
ecubicle	DOC	Try It Driving Directions Web Service	Provides driving directions	
ecubicle	DOC	Try It YouTube Downloader	Provides YouTube Download URL	
viju311985	RPC	Try It SMS Services	You Can Send Sms for free	
RestFulServices	DOC	Try It HCPCS Service	The Healthcare Common Procedure Coding System (HCPCS) is a set of health care procedure codes based on the American Medical Association's Current Procedural Terminology (CPT). Commonly pronounced Hick-Picks.	
SWD Services	DOC	Try It Live Match Score Service	The live match score service is a customizable software as a service (SaaS) component for displaying the most recent score of a particular sport match in business portals and websites.	
RestFulServices	DOC	Try It Current Converter	Provide conversion rates for given two currencies	MS.NET

Figure 4.7: XMethods Service Directory

As mentioned prior to this section, once consumers have searched and discovered their service of interest, for example, the live match score service in the XMethods service directory, they then retrieve the service’s feature model and, start fashioning-out their specialized software service, by selecting and disabling their required and unnecessary features. Figure 4.9 demonstrates how bet365 as a potential consumer of the live match score software service (cf. Section 3.4.1) can perform the selection of appropriate features and communicate a customization request. All the required and unwanted features are represented by a tick and a cross, respectively.

68

As pointed out earlier, the mutually inclusive and exclusive features are automatically selected and disabled, respectively, when the primary feature is selected (e.g., Figure 4.8). Figure 4.8 illustrates how the automatic selection or de-selection of interdependent features take place.

Referring back to the “live match score software feature model”, as described in Figure 3.5 (Section 3.4.1 of Chapter 3), the selection of the feature “After60sec” requires and/or includes the selection of the “At_most_99%” availability feature. Hence, the bet365 selection of the “After60sec” update information feature, as shown in Figure 4.8, compelled an automatic selection of the “At_most_99%” availability feature. This is illustrated by the two ticks in the check-boxes of these features.

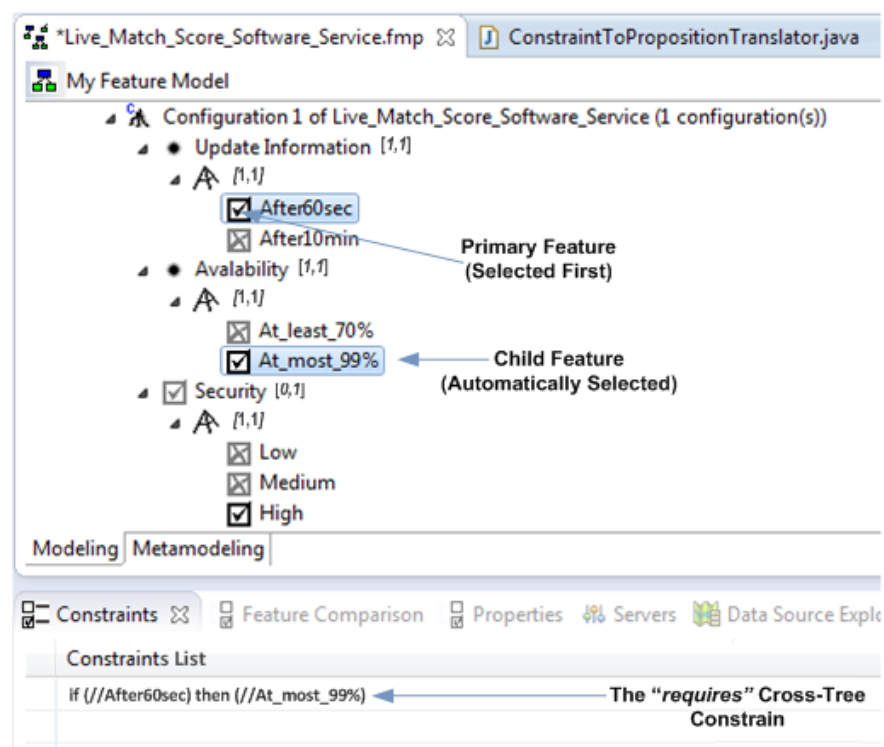


Figure 4.8: Automatic Selection of an Interdependent Feature (bet365)

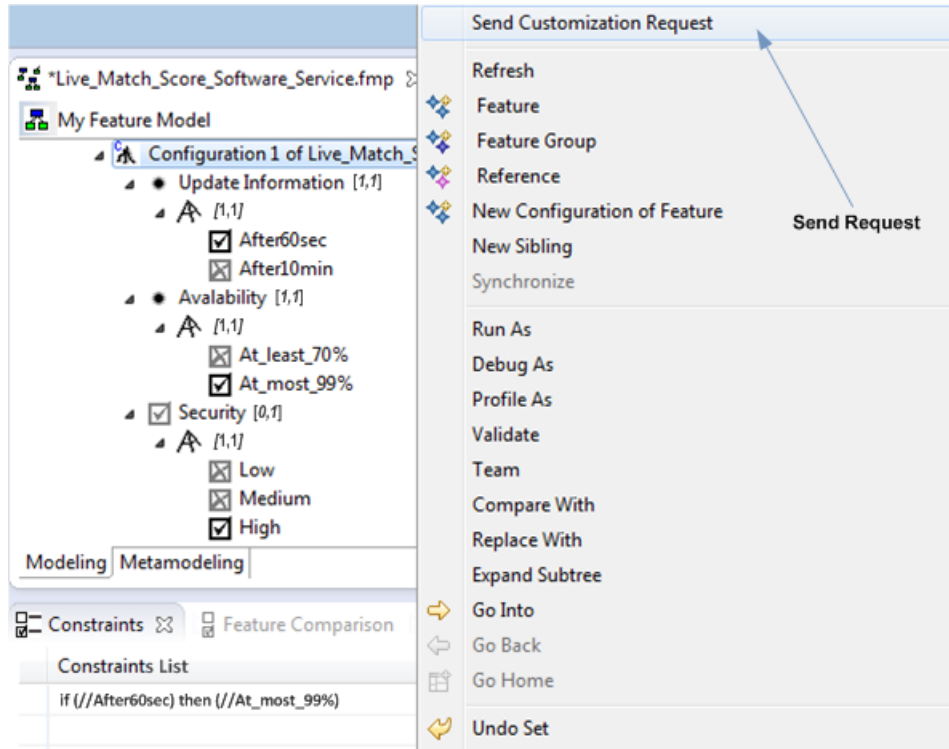


Figure 4.9: bet365 Customization Request

4.2.3.2. The Validation of Service Customization Requests

As mentioned before, in order to ensure the minimization of wastage of computing time and resources, service customization requests need to be validated. This is the main function of the feature management engine (cf. Section 3.3.1). This component was implemented by leveraging on the FAMA framework, which employs several Java logic representations and solvers, such as Boolean Satisfiability (SAT) and Constraint Satisfaction Problem (CSP) for enabling automated analysis of feature models. To validate a service customization request, the feature management engine takes as input the provider's software feature model and consumer's requested feature configurations, in an XML-based format (see Appendix F) and performs operations such as the validation of a feature model or validation of feature configurations.

Figure 4.10 represents the key code fragment for validating service customization requests. In the first line, the FAMA framework main class is instantiated, while lines 3 to 5 load the software service feature model. To read and get the features of the software service in question, lines 7 to 8 create two variables, one variable for iterating and reading through the service feature model, and the other for storing the returned element(s). Line 9 generates a configuration container, to store the requested set of (selected and disabled) features configurations, while lines 11 to 19 perform the actual reading and storing of required (selected) and unnecessary (disabled) features. The resulting set of feature configurations are applied to the provider's software service feature model (line 20). To finish, lines 21 to 26 query the validity of the customization request and check the results of the query.

```

1  QuestionTrader feature_engine_tester = new QuestionTrader();
2
3  GenericFeatureModel Software_Service_FM = (GenericFeatureModel)
4  feature_engine_tester.openFile("models/Live_Match_Score_Software_Service_FM.xml");
5  feature_engine_tester.setVariabilityModel(Software_Service_FM);
6
7  Iterator<? extends GenericFeature> iterate = Software_Service_FM.getFeatures().iterator();
8  VariabilityElement feature;
9  Configuration Software_Service_FM_Configs = new Configuration();
10
11 while(iterate.hasNext())
12 {
13     feature = iterate.next();
14     featureName = feature.getName();
15     if (feature.getIntegerValue(featureName) == 1)
16         Software_Service_FM_Configs.addElement(feature, 1);
17     else
18         Software_Service_FM_Configs.addElement(feature, 0);
19 }
20 feature_engine_tester.addStagedConfiguration(Software_Service_FM_Configs);
21 ValidQuestion Check_Service_Cust_Request = (ValidQuestion) feature_engine_tester.createQuestion("Valid");
22
23 if (Check_Service_Cust_Request.isValid())
24     System.out.println("Valid Customization Request");
25 else
26     System.out.println("Invalid Customization Request");

```

Figure 4.10: Key Java Snippet for Validating Service Customization Requests

4.3. Experimental Evaluations

This section presents the experiments conducted to evaluate the FreeCust approach described in this dissertation. The experiments are divided into two parts: usability experiment and request validation experiment. Each of these parts begins with a description of the experimental design and then details the evaluation.

In order to conduct the usability experiment, as reported or discussed in the succeeding subsection (section 4.3.1), appropriate measures were taken to ensure that the rights and welfare of participants in the study were protected. All participants were given the information and informed consent form (Appendix A). This information and informed consent form introduced the researcher, explained the purpose of the study, the risks involved, and the fact that participants are free and/or have the right to withdraw at any stage from the study. The manner in which the results of the study would be used was also explained. No specific ethical considerations were applicable to this study.

4.3.1. Usability Experiment

A usability evaluation was conducted to determine consumers' perception of FreeCust after an interaction experience. According to the International Organization for Standardization [ISO] 924-11 (1998) as cited in Daramola (2009) and ISO/NP 9241-11 (2008), usability is the degree to which specified stakeholders can use a system to achieve specified goals with effectiveness, efficiency and satisfaction. Moreover, usability is a perception of a system's ease of learning and use, from both the experienced and inexperienced stakeholders' viewpoint (Lindgaard, 1994 as cited in Daramola, 2009).

4.3.1.1. Experimental Design

The fundamental goal of the usability evaluation was to measure consumers' perception of FreeCust after an interaction experience. Correspondingly, the aim was to respond to the main research question raised in Chapter 1, namely, how can we ensure that the complexities involved in consumers' service customization processes are minimized to a potential efficient level? Efficient level is defined as a level at which non-ICT experts (such as SMMEs) without good background knowledge in Web service related technologies are able to customize software services.

The usability evaluation was conducted in the Postgraduate lab at the University of Zululand. The targeted population was non-ICT professionals (e.g., SMMEs), as well as other professionals with little knowledge in Web service related technologies. However, a representative sample size consisting of ten participants (six males and four females) was used in this evaluation. This was because a good quality usability test requires a set of five participants (Nielsen & Landauer, 1993; Nielsen, 2000; Virzi, 1992); while ten participants can give the most efficient feedback (Faulkner, 2003; Schmettow, 2012).

According to Steinberg (2011), a representative sample is a population in which participants have characteristics related to the actual targeted population of the study. Thus, the participants were required to have some experience and/or knowledge in operating a computer system and using the Internet. The participants were also required to have a little knowledge on Web services tools and software services in general. This type of data was collected using a pre-experiment background questionnaire (Appendix B). The usability metrics that were used in this evaluation are described as follows:

- (i) *Effectiveness*, which measures task success (the level of success to which participants completed the usability evaluation tasks), and
- (ii) *Consumer satisfaction*, the degree to which participants are content with FreeCust.

In order to obtain and determine participants' views on effectiveness and consumer satisfaction, a post-experiment usability questionnaire (see Appendix C) was used. The following section presents how the evaluation took place and the results that were obtained.

4.3.1.2. Process, Results and Analysis

As the first step, all participants were required to provide their informed consent to participate in the evaluation (Appendix A), and complete a background questionnaire (Appendix B). After that, the participants were briefed about the main purpose of the evaluation and the evaluation procedure, as well as the evaluation tasks (Table 4.2). Then, the evaluation started, with 35 minutes being set as the finish point for completing all the evaluation tasks. This was to reveal the approximate effective time for completing all the evaluation tasks.

Each participant was given a detailed task list (see Appendix D), and evaluated the notion of FreeCust individually. After the allocated time elapsed, all participants stopped and completed the post-experiment questionnaire (see Appendix C), which was adapted from the Computer System Usability Questionnaire (CSUQ) (Lewis, 1995) and NASA Task Load Index (NASA-TLX) (Hart & Staveland, 1988). Finally, the post-experiment questionnaires were collected from each participant, and the responses were correspondingly verified with respect to the (very) last executed task in the computer system.

Table 4.2: Usability Evaluation Tasks

Evaluation Tasks
1. Search for the live match score software service in the XMethods service directory.
2. Retrieve the software service feature model (i.e. the Live_Match_Score_Software_Service.fmp) file.
3. Import the software service feature model file in Eclipse, where the FreeCust GUI-flavored implementation is running.
4. Select and disable all the required and unnecessary features (features of interest).
5. Send customization request.

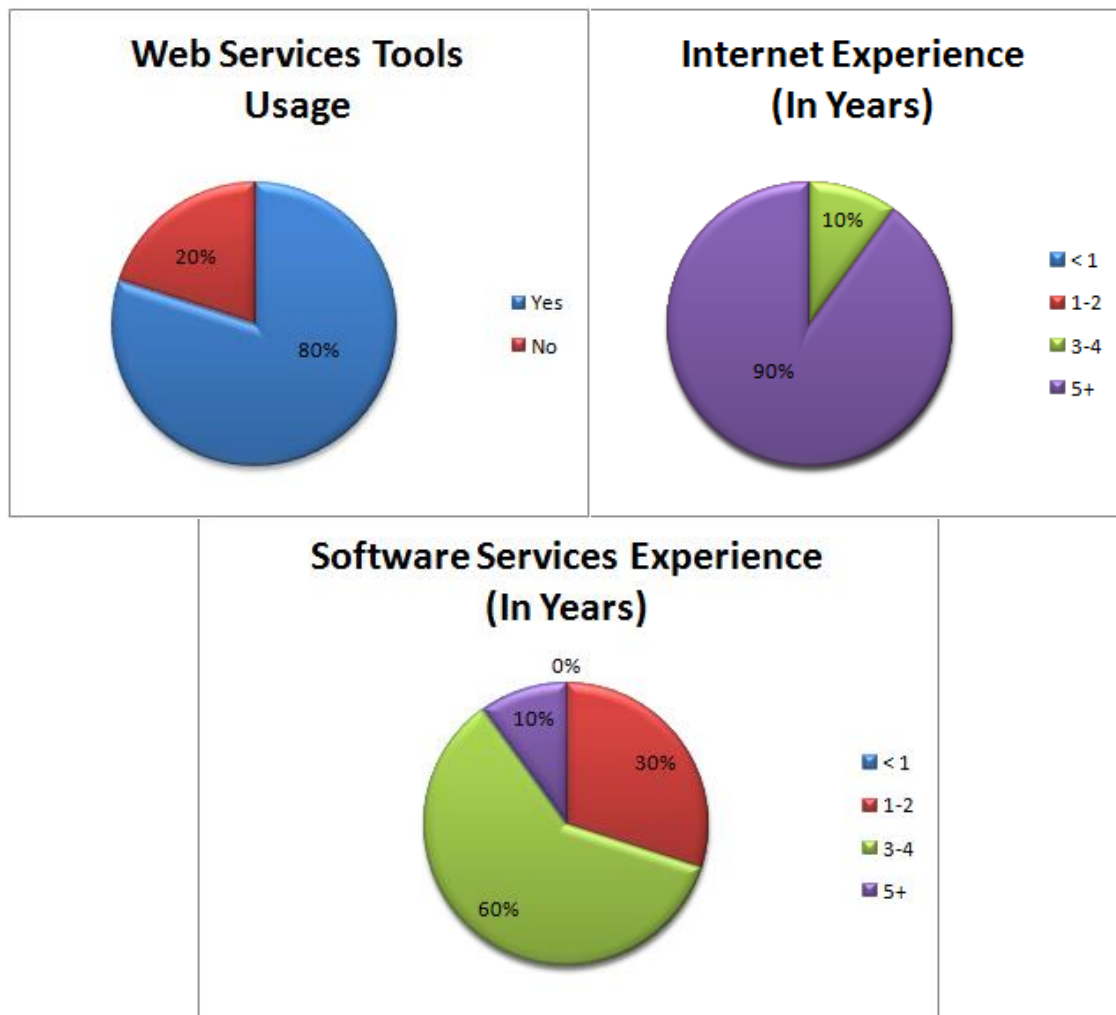


Figure 4.11: Participants Demographic Results (n=10)

Participants were in the age range of 20-30 years, and they all had at least five years' experience in operating a computer system. They were also asked whether they had any prior experience in using the Internet and Web services tools, as well as knowledge of software services (cf. Figure 4.11). 90 and 80 percent of the participants had respectively used Internet and Web services tools prior to taking part in the evaluation, and most of them had at least two years' experience and/or knowledge of software services.

As indicated earlier, FreeCust usability was tested using two usability metrics: effectiveness and consumer satisfaction. The results were captured in the post-experiment questionnaires, using a 7-point Likert scale and a 7-point semantic differential scale, and are presented next.

Effectiveness: for each task in the evaluation task list, (refer to Table 4.2), participants were required to respond to a question relating to each task. Accordingly, task success was measured as the percentage to which participants are able to complete each evaluation task. For instance, each completed task without assistance from the evaluator implies a successful rate of 100% task completion, while with a single iteration of the evaluator's assistance each completed task is given a 75% completion rate. Figure 4.12 displays the task success and failure rates for tasks 1 to 5. Each bar represents a single task, and is sub-divided into chunks, which represent the completion rate attained by participants. As the results indicate, all the evaluation tasks were successfully completed with a 100% completion rate, except for task 3, where 30% (n=3) of the participants completed the task with 75% completion rate, meaning that these participants needed some help from the evaluator. Nevertheless, the overall results show that, although participants were not proficient or familiar with the evaluation tasks, they managed to complete all the tasks within the finish mark, without facing any major challenges.

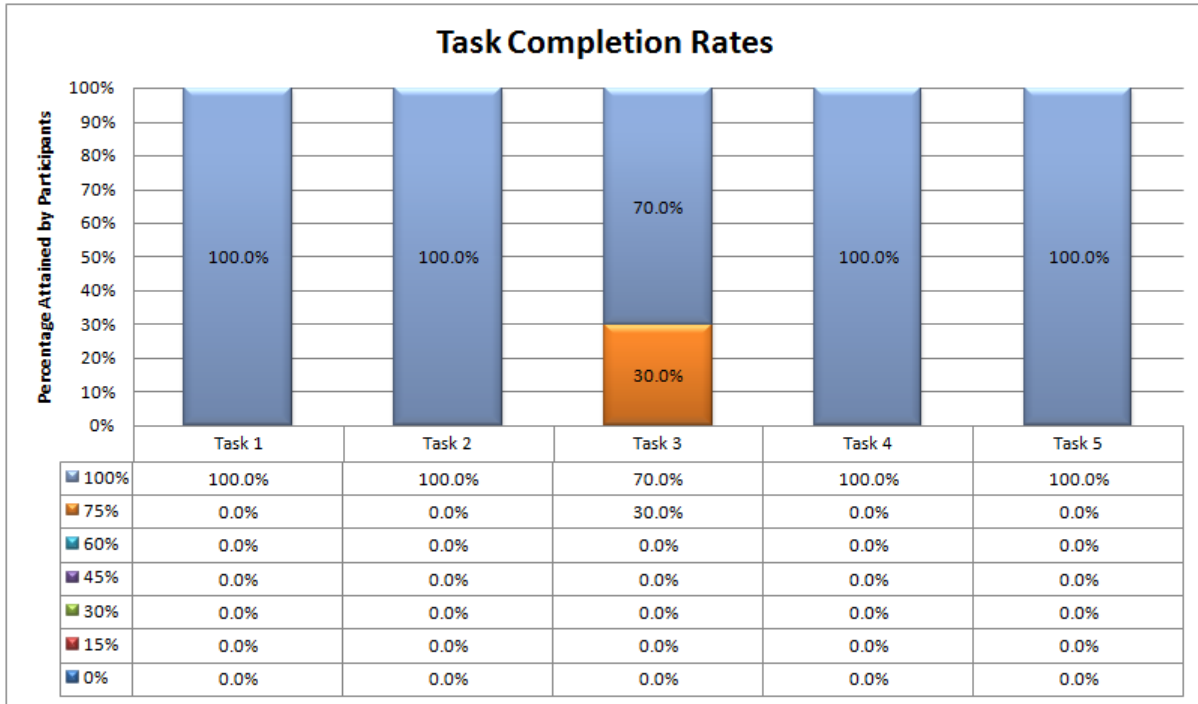


Figure 4.12: Participants Task Success Levels (n=10)

Consumer Satisfaction: the results of consumer satisfaction were captured and determined using two different aspects: (i) cognitive load and (ii) ease of use. Figures 4.13 and 4.14 show the results of these measures respectively. From the results, it can be deduced that the usability evaluation tasks outlined in Table 4.2 and the set of instructions detailed in Appendix D were not difficult to learn and perform. This is reflected in the fact that the mean average scores of “ease of use” ratings (see Figure 4.14) are above 6, on a scale 1 to 7. Accordingly, the results show and attest that all participants could effectively and efficiently complete the usability evaluation tasks, and that the process of learning and performing the novel tasks was not difficult (see Figure 4.13). This is also supported by the results showed in Figure 4.15, which highlight that participants were very satisfied with FreeCust.

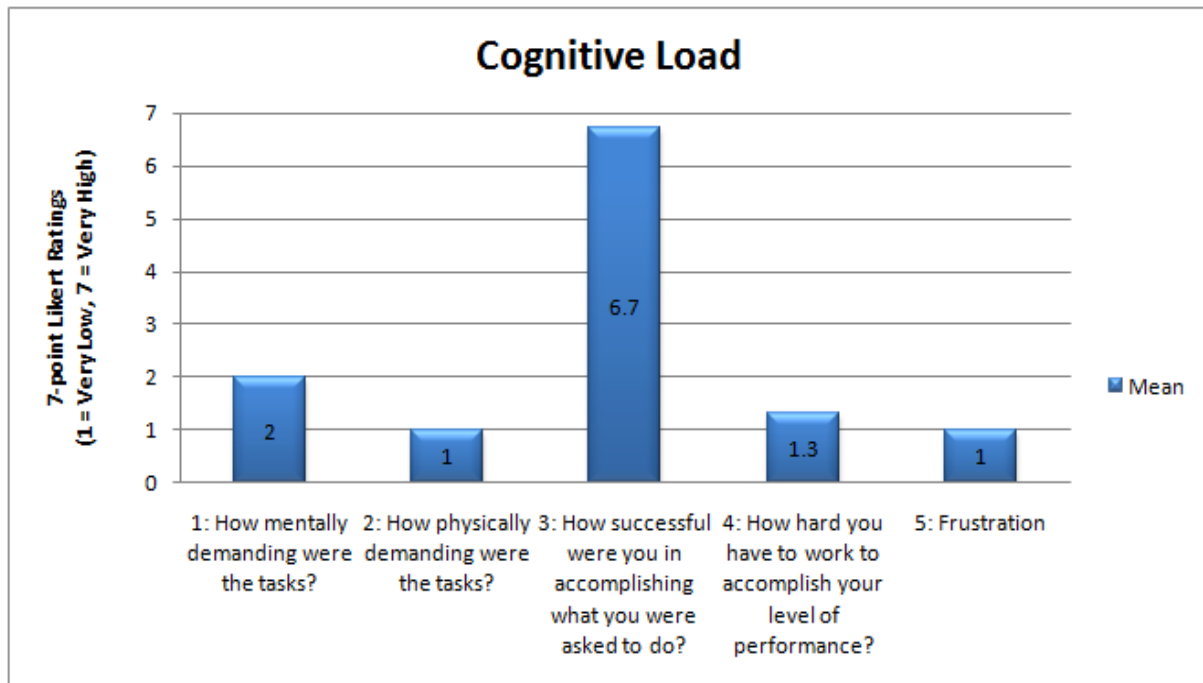


Figure 4.13: Participants Cognitive Load Using a 7-point Semantic Differential Scale (n=10)

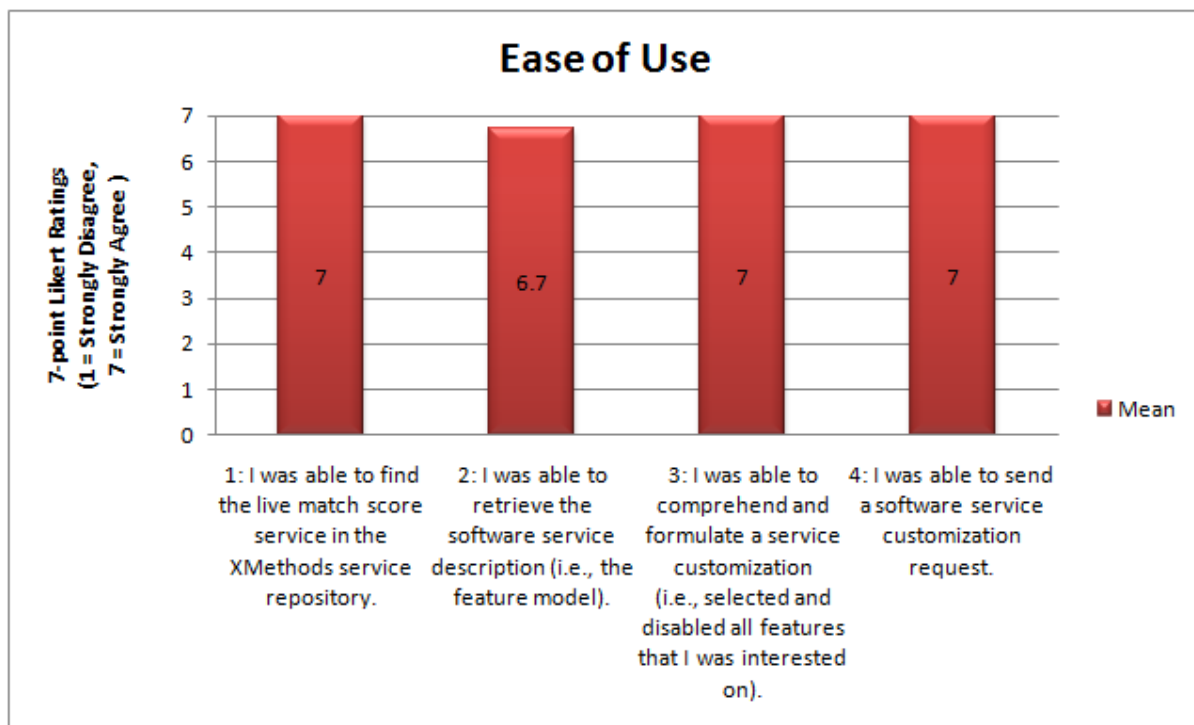


Figure 4.14: Ease of Use Results Using a 7-point Likert scale (n=10)

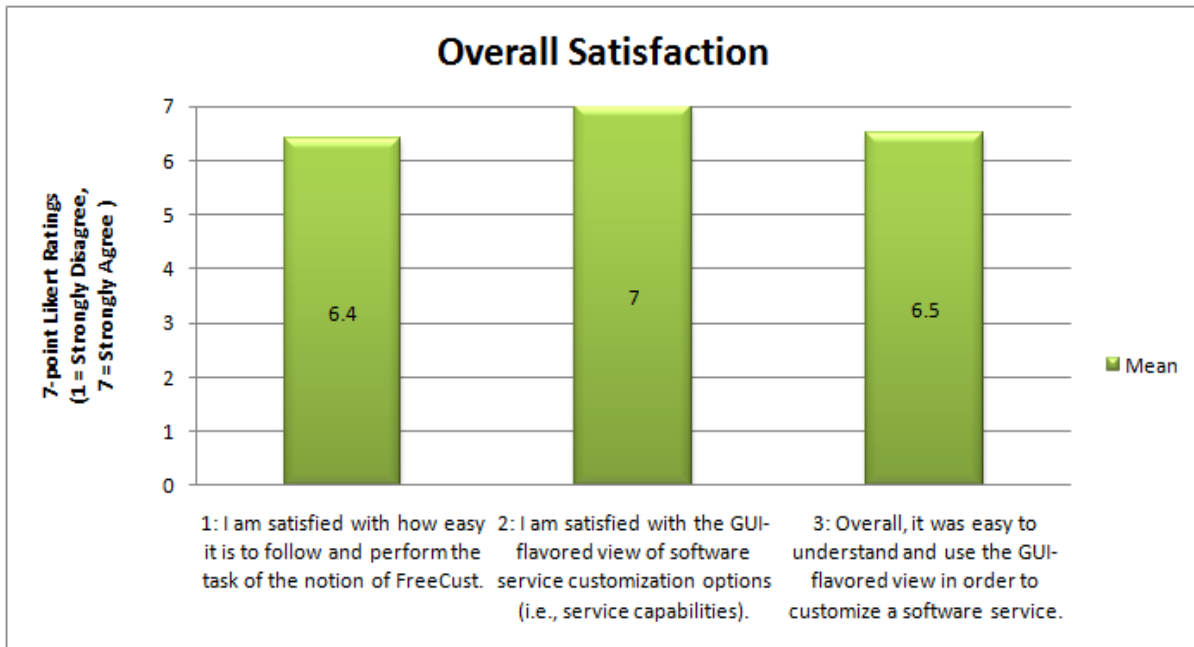


Figure 4.15: Overall Satisfaction Using a 7-point Likert scale (n=10)

4.3.2. Request Validation Experiment

This section describes the experiment conducted to evaluate the effectiveness of the feature management engine. In particular, the experiment was carried out to test the capability of the feature management engine in validating customization requests, i.e., checking whether a software service customization request is valid or not.

Taking into account the fact that the FreeCust GUI-flavored mechanism provides the capability to automatically confirm the validity of feature configurations (because of the automated selection and de-selection support of connected or dependent features) during the consumer's customization process. It means all feature configuration requests produced by the GUI-flavored mechanism are valid with respect to the original software service feature model. However, for testing whether the feature management engine is able to identify invalid requests, the FreeCust

GUI-flavored mechanism was not sufficient. The following sections describe how the invalid test was conducted as well as the outcome of the test.

4.3.2.1. Experimental Design

As mentioned in the foregoing section, the FreeCust GUI-flavored implementation only permits valid customization requests. Therefore, to generate and test if the feature management engine is able to detect invalid customization requests, the implementation was bundled as a jar file, and the console window program was used to simulate or send invalid customization requests. The test computations ran on a Windows 7 Home Premium Machine, with 3GB of RAM, 2.67 GHz Intel Core *i5* processor, and version 7 Java Runtime Environment (JRE). The live match score software service feature model (cf. Sections 4.3.4.1 and 3.4.1) was used for the test.

4.3.2.2. Process, Results and Analysis

Because of the small size of the live match score service feature model, only two invalid requests were simulated, as demonstrated in Figure 4.16. Evidently, from the figure, both requests are completely invalid; in particular, the first customization request is invalid in the sense that the mandatory “Update Information” feature was disabled, while the second request is invalid since the implicit interdependent feature of the “After60sec” update information child feature was disabled (i.e., the “At_most_99%” availability child feature). Hence, this violated the cross-tree constraint defined on these features. The responses shown in the console window (Figure 4.16), for each customization request confirm the correctness of the feature management engine in validating service customization requests.

```
C:\Users\SWD\Documents\Masters>cd "Feature Managment Engine"
C:\Users\SWD\Documents\Masters\Feature Managment Engine>cd src
C:\Users\SWD\Documents\Masters\Feature Managment Engine\src>java -jar FreeCust.jar Live_Match_Score_FM_Configurations.xml
Feature model file: Live_Match_Score_FM_Configurations.xml
Selected features: Availability, At_least_70%, Security, Medium
Disabled features: Update Information, After60sec, After10min, At_least_99%, Low, High
Response: Invalid Service Customization Request
C:\Users\SWD\Documents\Masters\Feature Managment Engine\src>java -jar FreeCust.jar Live_Match_Score_FM_Configurations.xml
Feature model file: Live_Match_Score_FM_Configurations.xml
Selected features: Update Information, After60sec, Availability, At_least_70%, Security, High
Disabled features: After10min, At_most_99%, Low, Medium
Response: Invalid Service Customization Request
```

Figure 4.16: Service Customization Request Validation

4.4. Experimental Evaluations Result Discussion

As mentioned earlier, the main goal of the foregoing experimental evaluations, particularly the usability experimental evaluation, was to test the notion of FreeCust and correspondingly answer the key research question raised in Chapter 1. The question reads as follows: how can we ensure that the complexities involved in consumers' service customization processes are minimized to a potential efficient level? Efficient level was defined as a level at which non-ICT experts (such as SMMEs) without good background knowledge in Web service related technologies are able to customize software services. From the overall usability evaluation results, it can be inferred that the FreeCust approach simplifies the service customization process, since it provides a user-friendliness solution that is easy enough to be used by novice users and/or non-ICT experts. In particular, the results show that the complexities involved in consumer's service customization processes can be effectively dealt with by describing and representing service capabilities (i.e., functional and non-functional properties) at the highest level of abstraction, and by introducing a visual view mechanism for representing customization options, as well as a mechanism that can

automatically select and disable dependent customization options. These enable and help service consumers to easily comprehend and make business-oriented service customization choices.

4.5. Comparative Analysis

According to Hofstee (2006), a comparative analysis plays a critical role in assessing any new solution against the existing similar solutions. Thus, to systematically compare FreeCust with existing service customization solutions, and note its benefits, the research in this dissertation devised a number of criteria that need to be supported by service customization solutions, in order to efficiently deliver customizable software services (that best meet the varying demands of consumers), in a rapid and cost-effective manner and to provide a consumer-aware and friendly customization solution.

As motivated by (Czarnecki *et al.*, 2012; Schmid, Rabiser, & Grunbacher, 2011), the group of criterions that formed part of the systematic comparison were defined by following the bottom-up and top-down practices. That is, in following the bottom-up practice, a number of important aspects that were identified from analyzing and reviewing existing related works were included in the criteria group. The other important aspects that were included in the group, through the top-down practice, are the aspects that were gathered from critically reviewing, analyzing, and reasoning about existing literature on software variability management (e.g., how is consumer-introduced variability handled in the SPL engineering discipline).

These criteria include: (i) minimizing customization complexity, (ii) request validation, (iii) technical blindness, (iv) implicit interdependencies, and (v) the management of functional and non-functional requirements. It is not claimed that the aforementioned list is complete, but it does provide appropriate aspects to compare the FreeCust approach with existing approaches.

The rest of this section analyzes existing service customization approaches, with respect to the aforementioned criteria. The results of the comparative analysis are summarized in Table 4.3 using the following symbols: (+) criterion met, (-) criterion not met, and (+/-) criterion partially met.

Minimizing Customization Complexity: as mentioned in the rest of this dissertation, service capabilities should be modeled and/or represented at the highest level of abstraction. This is because the number of variation points and variants increase when variability is modeled at the lowest levels of abstraction; which in turn creates a burden and becomes a complex task for service consumers to easily interpret and make customization decisions. Hence, describing and representing service properties at the highest level of abstraction (i.e., at feature or requirements level) exhibits fewer variation points and variants compared to the service interface level. Thus, when comparing **FreeCust** – the **Feature Model-based Service Customization** approach with existing similar approaches reviewed in Chapter 2, that describe variability at technical levels, FreeCust is the only approach which describes and represent service capabilities at the highest level of abstraction. Accordingly, it minimizes customization complexities and simplifies the customization task for service consumers.

Request Validation: as also mentioned in the rest of this dissertation, the excess number of possible customization options that software services exhibit make customizing a service a very error-prone task. Thus, to ensure that service consumers do not breach any properties described by service providers, and to minimize wastage of computing time and resources, a mechanism for automatic validation of consumers' customization requests is necessary. Hence, the FreeCust

approach and most of the other reviewed service customization approaches, expect Liang *et al.* (2006) and Barros *et al.* (2011), support such a mechanism.

Technical Blindness: as compared to other customization approaches reviewed in Chapter 2, FreeCust is the only approach that does not require service consumers to be familiar with the technicalities of realizing software services in order to perform service customizations. Rather, it enables consumers, during the customization task, to reason more about what a software service can provide and achieve, at the highest level of abstraction.

Implicit Interdependencies: most of the existing service customization approaches represent the dependencies among service customization options using natural languages. However, this adds more complexities in consumers' service customization processes. The employment of the feature modeling technique in FreeCust enables the capturing of (inter) dependencies among customization options as cross-tree constraints (cf. Sections 2.5.3 and 4.3.4.1). This ensures a consistent way of representing customization options and helps in reasoning more about service customization decisions.

Managing Functional and Non-functional Requirements: as stated in Chapter 2, existing service customization approaches only focus on managing variable functional requirements of service consumers; they do not take into consideration the fact that, in reality, consumers usually exhibit varying functional and non-functional requirements. Accordingly, the FreeCust approach covers both the variable functional and non-functional requirements of service consumers.

Table 4.3: Comparative Analysis of Existing Service Customization Approaches

<div>Criteria</div> <div>Approach</div>	Minimizing Customization Complexity	Request Validation	Technical Blindness	Implicit Interdependencies	Managing Functional and Non-Functional Requirements
Liang <i>et al.</i> (2006)	-	-	-	-	+/-
Stollberg and Muth (2009)	-	+	-	-	+/-
Nguyen and Colman (2010)	-	+	-	+/-	+/-
Barros <i>et al.</i> (2011)	-	+/-	-	-	+/-
The FreeCust Approach	+	+	+	+	+

The comparative analysis presented in this section revealed some distinctive differences between the FreeCust approach and existing similar solutions. Where essential aspects were found to be supported across other solutions, the differences between those aspects also revealed how the FreeCust approach addresses the challenges of building and provisioning efficient customizable software services.

4.6. Summary

In this chapter, the proof-of-concept implementation and technologies exploited to realize the notion of FreeCust were demonstrated. First, the chapter gave a high-level overview of the FreeCust proof-of-concept prototype. Thereafter, the platforms, tools and frameworks that were used in the implementation to realize the functions of FreeCust were presented. Then, the notion and functions of FreeCust were evaluated, with the aim of testing the effectiveness of FreeCust

and responding to the key research question raised in Chapter 1. The results of the usability evaluation show that this research question was dealt with. Finally, the chapter, systematically compared FreeCust with other existing service customization approaches. The comparative analysis results demonstrate that the FreeCust approach has and/or delivers the necessary characteristics for an effective and efficient service customization solution. To round off the work done in this research, the next chapter summarizes the dissertation by highlighting the key achievements and contributions of this work. The chapter also gives the limitations of the FreeCust approach as presented in this dissertation, as well as future directions for further investigations.

CONCLUSION AND FUTURE WORK

This chapter concludes the research work presented in this dissertation. It first highlights the achievements, according to the objectives set out at the beginning of this dissertation (Section 5.1), and the major conclusions drawn from this research (Section 5.2). Thereafter, it reviews the research methodology, through which the contributions of this research to the field of service-oriented computing are outlined (Section 5.3). Finally, the chapter describes the limitations of the feature model-based service customization approach as presented in this dissertation, and points out a number of interesting avenues of future research (Section 5.4).

5.1. Research Summary

As mentioned throughout this dissertation, SOA enables the design and realization of Software-as-a-Service (SaaS) business model, where organizations such as SMMEs can acquire software services embodied and maintained by service providers, through the Internet, without having to purchase their own ICT infrastructures. Although these organizations (i.e., service consumers) can have access to software services and utilize them in their business processes, in reality, they often have diverse business and feature requirements. As a result, service providers are faced with various challenges to stay relevant in today's global economy. This, together with new market opportunities, necessitates mechanisms for rapid development and delivery of software services that best meet consumers' demands. Hence, some service providers have motivated the acceptance of customizable software-development methods, to develop and deploy customized services for their potential consumers. However, during this process of offering customizable

services, for service providers to systematically provide efficient services, they need to consider both the technological knowledge as well as variable functional and non-functional requirements of service consumers.

Chapter 1 (Section 1.4) stated that the main goal of this dissertation was to develop a service customization mechanism for managing consumer-introduced variability in SOA environments. This mechanism needed to have less customization complexity. To tackle this, several research objectives and questions were formulated as a guide for which the research in this dissertation was conducted. The primary research question was phrased as follows: how can we ensure that the complexities involved in consumers' service customization processes are minimized to a potential *efficient level*? Efficient level is the level at which non-ICT experts without good background knowledge in Web service related technologies can be able to customize software services. The aforementioned research question generated two sub-questions: (i) how can a mechanism that enables graphical and/or visual representation of service customization options be implemented? And, (ii) can the number of possible interdependencies among service customization options be automatically selected? The following summarizes how the primary research question and its sub-questions were addressed.

Sub-Research Questions: the FreeCust consumer-oriented GUI implementation as presented in Chapter 4 addressed both the sub-research questions. In particular, the sub-questions were dealt with by exploiting the concepts, techniques, and tools of feature modeling from the SPL engineering discipline. Precisely, in order to graphically represent service customization options, the open-source tool called featurePlugin was leveraged. The function of automatic selection and de-selection of interdependent service customization options was also implemented through

the exploitation of this tool; as it also supports the feature cross-tree constraint mechanism. This covered the second sub-research question.

Main Research Question: with the support of the answers of the sub-research questions, the main research question was addressed through a usability test (cf. Section 4.4.1), where a sample of ten representative participants tested the effectiveness of the notion of FreeCust. The overall results of usability testing show that the complexities faced by service consumers when they are performing service customization tasks can be efficiently minimized by taking advantage and/or bringing in the concepts and techniques of SPL engineering to the development and delivery of customizable software services. Hence, FreeCust was guided by the feature modeling concept.

5.2. Conclusions

As mentioned before, the main purpose of this research study was to investigate and formulate a service customization mechanism, so as to manage consumer-introduced variability and simplify the complexities involved in building and provisioning customizable software services. In an attempt to design and realize such a solution, it became essential to specify the design principles. These principles, which are presented and discussed in Chapter 3 (Section 3.2) emanated from the literature, that is, from critically reviewing, analyzing, and reasoning about existing related research and the relevant literature on software variability management.

The design principles provided the basis for designing and developing a solution that would not only address the challenges of building customizable software services, but would further ensure that the solution is future-proof in terms of extensibility and re-usability. Once the principles were specified, a **Feature Model-based Service Customization (FreeCust)** framework was designed and developed.

To validate the plausibility and the relevance of the FreeCust solution approach, a number of different research evaluation techniques were adopted. In particular, qualitative and quantitative approaches were exploited. Using two real-world use case scenarios, the utility and applicability of the FreeCust solution were demonstrated. This was followed by two controlled experimental evaluations (i.e., usability and effectiveness). The main goal of these experimental evaluations was to demonstrate the practicability and evaluate the significance and relevance of the notion of FreeCust.

From the overall results obtained in the experimental evaluation activities, particularly usability evaluation, it can be reasoned and concluded that the FreeCust solution minimizes customization complexities and simplifies the customization task and/or procedure for service consumers. This is due to the fact that FreeCust presents a user-friendly solution that is simple and yet useful for supporting either the novice users or non-ICT experts, as well as expert service engineers. In particular, the findings reveal that the complexities involved in service customization processes can be effectively addressed by: (i) describing and representing service capabilities at the highest level of abstraction, (ii) introducing a visual view and/or graphical mechanism for representing customization options, and (iii) building a mechanism that can automatically select and disable dependent customization options. These aforementioned aspects (will) enable and help service consumers to easily comprehend and select business-oriented customization options.

The FreeCust solution was further evaluated through a comparative analysis (referred to as *static analysis* in the evaluation stage of design science), which provided a setting for systematically and qualitatively gauging the solution with existing solutions in the literature.

In the analysis (cf. Section 4.5), the key differentiators between FreeCust and related solutions were highlighted and clarified. From the analysis, it became evident that the value propositions of the FreeCust solution are: (i) handling both the functional and non-functional requirements of service consumers, (ii) minimizing the complexities and simplifying the customization task for service consumers, and (iii) providing a consistent way for representing customization options and helping facilitate the reasoning of customization decisions.

5.3. Review of the Research Methodology and Contributions

As stated in Chapter 1 (Section 1.5), the research in this dissertation followed the design science research methodology. Thus, to assess the results of this research against the requirements for an effective design science research, Hevner *et al.* (2004) defined seven guidelines, which are summarized as follows:

Design as an Artifact: this guideline states that research efforts guided by the design science research paradigm must produce some viable artifacts. Thus, the research work presented in this dissertation produced two viable artifacts, that is, the FreeCust architecture framework and its proof-of-concept implementation, as well as the Non-Functional Property (NFP) model.

Problem Relevance: this guideline requires design science research to develop technology-based solutions to truly important and relevant business problems. As defined in Chapter 1, the consumer-introduced variability problem, which is caused by the diversity of business needs and requirements of service consumers, as well as the technological knowledge level of consumers in modern service-oriented business environments, is truly an important business problem. Hence, it requires organizations (i.e., service providers) to keep up with global market changes, and necessitates efficient mechanisms and/or solutions for the rapid development and delivery of

customizable software services that best meet consumers' demands. The FreeCust technology-based solution as developed in this dissertation is totally in line with and attempts to address the aforementioned business problem.

Design Evaluation: this aspect involves and/or requires well-executed evaluation methods to evaluate the utility, relevance and applicability of the proposed solution to the identified business problem. In the context of the research in this dissertation, two real-world application scenarios were constructed to demonstrate the utility and applicability of the FreeCust solution approach. Furthermore, two controlled experimental evaluations (i.e., usability and effectiveness) were also conducted to illustrate the practicability and evaluate the significance and relevance of FreeCust. The key characteristics and benefits of the proposed (FreeCust) approach were presented via a comparative analysis against other existing similar approaches.

Research Contributions: this guideline states that the proposed solution must provide clear contributions. The conceptual FreeCust architecture framework as suggested in this dissertation serves as the primary contribution to the field of service-oriented computing, as it provides a solution or method in which organizations such as service providers and service consumers can, respectively, realize the design and provisioning of efficient customizable software services, and help in minimizing development time and costs of software services, since consumers may not need to hire too many experts from different fields in order to customize software services. The NFP model (cf. Section 3.3.2) is the secondary contribution. It contributes to both the fields of service and software product line engineering.

Research Rigor: this refers to the rigorous way in which the design science research was undertaken. For the research in this dissertation, rigorous methods were used in both the

construction and evaluation of the FreeCust approach. As a case in point, the FreeCust framework was formulated based on the results that were gleaned from reviewing, analyzing, and reasoning about the characteristics of an efficient and effective software customization solution, and from analyzing relevant existing service customization solutions. The feasibility, applicability, and utility of the FreeCust approach were demonstrated through real-world use case scenarios and proof-of-concept experimentations. In addition, a comparative evaluation (called *static analysis* in the evaluation stage of design science research) was performed, to systematically compare FreeCust with other existing service customization solutions.

Design as a Search Process: this guideline requires that when conducting design science research, an effective solution to the identified business problem must be found through a thorough search process. In view of that, the development of the FreeCust solution approach was really a search process. Hence, the research was conducted over the period of two and a half years, where different research methods and strategies were employed in the development and evaluation of FreeCust.

Communication of Research: this is the final guideline. It states that design science research must be reported or presented to both technology-oriented and management-oriented audiences. As mentioned at the beginning of this dissertation (cf. LIST OF PUBLICATIONS), portions of the work reported in this dissertation have been published and/or presented at the 16th Annual Southern Africa Telecommunication Networks and Applications Conference (SATNAC) 2013, 15th Annual Conference on the World Wide Web Applications (ZA-WWW) 2013, and Volume 8, Issue No.: (1) of the International Journal of Information Technology and Computer Science (IJITCS) 2013.

5.4. Limitations and Future Work

The FreeCust approach as implemented in this dissertation has several limitations. This section describes the limitations and discusses the possible directions of future work, to which the FreeCust solution can be improved and enhanced for addressing its limitations.

As mentioned in Chapter 1 (Section 1.6), the service customization process can be generally broken down into two parts. That is, the part whereby consumers comprehend customization options and perform service customizations, and the part where service providers derive and deploy customized software service variants and manage those variants. The FreeCust proof-of-concept implementation, as carried out in this dissertation focused only on the first part of the service customization process, as described above. The realization of the second part is the subject of future research. In addition, the following techniques - (Java) reflection, XSL and ATLAS model transformation languages, as well as Acceleo and WSDL2Java tools could be investigated in trying to realize this part.

Although the use of feature models has its merits in the FreeCust approach, when addressing the second part of the service customization process, it should be noted that feature models might only be more appropriate for software services that are owned and maintained by the developing organization. For instance, when taking GUISET into context, feature models might be more appropriate for GUISET native services only; this is because the logic and implementations of software services would be owned by GUISET and available for alterations. In the case where a customizable service requires a feature that is provided by an external entity to GUISET, the use of feature models might not be sufficient. Hence, more work would have to be carried out to

explore how the identification and mapping of features to the implementation logic of external services can be integrated in the FreeCust service derivation and deployment process.

BIBLIOGRAPHY

- Acher, M., Collet, P., Lahire, P., & France, R. (2010). Managing variability in workflow with feature model composition operators. In B. Baudry & E. Wohlstadter (Eds.), *Software Composition* (pp. 17–33). Springer Berlin Heidelberg.
- Adigun, M., Emuoyibofarhe, O., & Migira, S. (2006). *Challenges to access and opportunity to use SMME enabling technologies in Africa*. Paper presented at the 1st All African Technology Diffusion Conference, Johannesburg, South Africa.
- Aiello, M., Bulanov, P., & Groefsema, H. (2010). Requirements and tools for variability management. *Proceedings of the 2010 IEEE 34th Annual Computer Software and Applications Conference Workshops* (pp. 245–250). Washington, DC, USA: IEEE Computer Society.
- Al-Masri, E., & Mahmoud, Q. H. (2008a). Investigating web services on the World Wide Web. *Proceedings of the 17th International Conference on World Wide Web* (pp. 795–804). New York, NY, USA: ACM.
- Al-Masri, E., & Mahmoud, Q. H. (2008b). Toward quality-driven web service discovery. *IT Professional*, 10(3), 24–28. doi:10.1109/MITP.2008.59
- American National Standards Institute. (1990). *IEEE Std 610.12-1990: IEEE standard glossary of software engineering terminology*. doi:10.1109/IEEESTD.1990.101064
- Andrés, R. de, García-Lapresta, J. L., & Martínez, L. (2010). A multi-granular linguistic model for management decision-making in performance appraisal. *Soft Computing*, 14(1), 21–34. doi:10.1007/s00500-008-0387-8
- Antkiewicz, M., & Czarnecki, K. (2004). FeaturePlugin: Feature modeling plug-in for eclipse. *Proceedings of the 2004 OOPSLA Workshop on Eclipse Technology eXchange* (pp. 67–72). New York, NY, USA: ACM.
- Arsac, W., Laube, A., & Plate, H. (2013). Policy chain for securing service-oriented architectures. In R. D. Pietro, J. Herranz, E. Damiani, & R. State (Eds.), *Data Privacy*

- Management and Autonomous Spontaneous Security* (pp. 303–317). Springer Berlin Heidelberg.
- Babar, M. A., Chen, L., & Shull, F. (2010). Managing variability in software product lines. *IEEE Software*, 27(3), 89–94. doi: 10.1109/MS.2010.77
- Bachmann, F., Goedicke, M., Leite, J., Nord, R., Pohl, K., Ramesh, B., & Vilbig, A. (2004). A meta-model for representing variability in product family development. In F. J. van der Linden (Ed.), *Software Product-Family Engineering* (pp. 66–80). Springer Berlin Heidelberg.
- Barros, A., Allgaier, M., Charfi, A., Heller, M., Kylau, U., Schmeling, B., & Stollberg, M. (2011). Diversified service provisioning in global business networks. *Annual SRII Global Conference (SRII)* (pp. 716–728). doi:10.1109/SRII.2011.78
- Benavides, D. (2009). *Automated analysis of feature models: A detailed literature review (Version 1.0)*. Retrieved from <http://www.lsi.us.es/~dbc/en/?download=isa-09-tr-04-v1.pdf>
- Benavides, D., Segura, S., & Ruiz-Cortés, A. (2010). Automated analysis of feature models 20 years later: A literature review. *Information Systems*, 35(6), 615–636.
- Benavides, D., Trinidad, P., & Ruiz-Cortés, A. (2005). Automated reasoning on feature models. In O. Pastor & J. F. e Cunha (Eds.), *Advanced Information Systems Engineering* (pp. 491–503). Springer Berlin Heidelberg.
- Bézivin, J. (2005). On the unification power of models. *Software & Systems Modeling*, 4(2), 171–188. doi:10.1007/s10270-005-0079-0
- Bianco, P., Kotermanski, R., & Merson, P. (2007). *Evaluating a service-oriented architecture* (Technical Report No. CMU/SEI-2007-TR-015). Pittsburgh, PA, Carnegie Mellon University: Software Engineering Institute.
- Blewitt, A. (2012). *Eclipse juno and the future of the eclipse platform*. Retrieved March 9, 2014, from <http://www.infoq.com/news/2012/01/eclipse-juno>

- Bloomberg, J. (2007). *How to define a business service* (White paper). Retrieved from http://www.softwareag.com/jp/Images/HowDefineBusinessService-SoftwareAG-112007-WP-0165-1_tcm87-35243.pdf
- Bocchi, L., Fiadeiro, J. L., & Lopes, A. (2008). A use-case driven approach to formal service-oriented modeling. In T. Margaria & B. Steffen (Eds.), *Leveraging Applications of Formal Methods, Verification and Validation* (pp. 155–169). Springer Berlin Heidelberg.
- Bosch, J. (2000). *Design and use of software architectures: Adopting and evolving a product-line approach*. New York, NY, USA: ACM Press/Addison-Wesley Longman Publishing Co., Inc.
- Bosch, J. (2009). From software product lines to software ecosystems. *Proceedings of the 13th International Software Product Line Conference* (pp. 111–119). Pittsburgh, PA, USA: Carnegie Mellon University.
- Bosch, J., & Capilla, R. (2012). Dynamic variability in software-intensive embedded system families. *Computer*, 45(10), 28–35. doi:10.1109/MC.2012.287
- Capilla, R., & Bosch, J. (2011). The promise and challenge of runtime variability. *Computer*, 44(12), 93–95. doi:10.1109/MC.2011.382
- Cavanaugh, E. (2006). *Web services: Benefits, challenges, and a unique, visual development solution* (White paper). Retrieved from <http://www.altova.com/whitepapers/webservices.pdf>
- Chang, S. H., & Kim, S. D. (2007). A variability modeling method for adaptable services in service-oriented computing. *Proceedings of the 11th International Software Product Line Conference* (pp. 261–268). Washington, DC, USA: IEEE Computer Society.
- Chen, L., Ali Babar, M., & Ali, N. (2009). Variability management in software product lines: a systematic review. *Proceedings of the 13th International Software Product Line Conference* (pp. 81–90). Pittsburgh, PA, USA: Carnegie Mellon University.

- Clement, L., Hately, A., von Riegen, C., Rogers, T., Kochman, R., Macias, P., Dovey, M. (2004). *UDDI Version 3.0, Organization for the Advancement of Structured Information Standards (OASIS)*. Retrieved from http://uddi.org/pubs/uddi_v3.htm
- Clements, P., & Northrop, L. (Eds.). (2001). *Software product lines: Practices and patterns* (3rd ed.). Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.
- Cohen, S., & Krut, R. (2010). *Managing variation in services in a software product line context*. (Technical Report No. CMU/SEI-2010-TN-007). Pittsburgh, PA, Carnegie Mellon University: Software Engineering Institute.
- Czarnecki, K., & Eisenecker, U. W. (2000). *Generative programming: methods, tools, and applications*. New York, NY, USA: ACM Press/Addison-Wesley Longman Publishing Co., Inc.
- Czarnecki, K., Grünbacher, P., Rabiser, R., Schmid, K., & Wkasowski, A. (2012). Cool features and tough decisions: a comparison of variability modeling approaches. *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems* (pp. 173–182). New York, NY, USA: ACM.
- Czarnecki, K., & Helsen, S. (2006). Feature-based survey of model transformation approaches. *IBM Systems Journal*, 45(3), 621–645. doi:10.1147/sj.453.0621
- Czarnecki, K., Helsen, S., & Eisenecker, U. (2005). Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1), 7–29.
- Dai, W. (2010). The impact of emerging technologies on small and medium enterprises. *Journal of Business Systems, Governance and Ethics*, 4(4), 53–60.
- Daramola, J. (2009). *A software product line approach to ontology-based recommendations in e-tourism systems* (Doctoral dissertation). Covenant University, Ota, Nigeria.
- de Oliveira Junior, E. A., Gimenes, I. M. S., Huzita, E. H. M., & Maldonado, J. C. (2005). A variability management process for software product lines. *Proceedings of the 2005*

- Conference of the Centre for Advanced Studies on Collaborative Research* (pp. 225–241). Toronto, Ontario, Canada: IBM Press.
- Deelstra, S., Sinnema, M., & Bosch, J. (2005). Product derivation in software product families: A case study. *Journal of Systems and Software*, 74(2), 173–194. doi:10.1016/j.jss.2003.11.012
- Dhungana, D., Grünbacher, P., & Rabiser, R. (2011). The DOPLER meta-tool for decision-oriented variability modeling: A multiple case study. *Automated Software Engineering*, 18(1), 77–114. doi:10.1007/s10515-010-0076-6
- Dlamini, S. W., Tarwireyi, P., & Adigun, M. O. (2013). Maximizing Web Service Applicability and Consumption through Customization with Feature Modelling. In *Proceedings of the 15th Annual Conference on World Wide Web Applications*. CPUT, Cape Town, Western Cape
- Dustdar, S., & Treiber, M. (2005). A view based analysis on web service registries. *Distributed and Parallel Databases*, 18(2), 147–171. doi:10.1007/s10619-005-2460-y
- Elio, R., Hoover, J., Nikolaidis, I., Salavatipour, M., Stewart, L., & Wong, K. (2008). *About computing science research methodology penned by Jose Nelson Amaral with significant contributions from Michael Buro* (Research Report No. 45940429). Retrieved from <http://webdocs.cs.ualberta.ca/~c603/readings/research-methods.pdf>
- Erl, T. (2005). *Service-oriented architecture: Concepts, technology, and design*. Upper Saddle River, NJ: Prentice Hall Professional Technical Reference.
- Faulkner, L. (2003). Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 379–383. doi:10.3758/BF03195514
- Fenton, N. E., & Pfleeger, S. L. (Eds.). (1998). *Software metrics: A rigorous and practical approach* (2nd ed.). Boston, MA, USA: PWS Publishing Co.

- Fernandez-Amoros, D., Gil, R. H., & Somolinos, J. C. (2009). Inferring information from feature diagrams to product line economic models. *Proceedings of the 13th International Software Product Line Conference* (pp. 41–50). Pittsburgh, PA, USA: Carnegie Mellon University.
- France, R., & Rumpe, B. (2007). Model-driven development of complex software: A research roadmap. *Proceedings of FOSE 2007 Future of Software Engineering* (pp. 37–54). Washington, DC, USA: IEEE Computer Society.
- Frei, F. X. (2006). Breaking the trade-off between efficiency and service. *Harvard Business Review*, 84(11), 93–101, 156.
- Genova, G., Valiente, M., & Marrero, M. (2009). On the difference between analysis and design, and why it is relevant for the interpretation of models in model driven engineering. *Journal of Object Technology*, 8(1), 107–127. doi:doi:10.5381/jot.2009.8.1.c7
- Geurts, B. (2003). Reasoning with quantifiers. *Cognition*, 86(3), 223–251.
- Glinz, M. (2007). On non-functional requirements. *Proceedings of the 15th IEEE International Requirements Engineering Conference, RE '07* (pp. 21–26). Delhi: IEEE Computer Society.
- Gohar, A. (2010). *Analyzing service-oriented architecture (SOA) in open source products* (master's thesis). Uppsala University, School of Innovation, Design and Engineering (IDT). Retrieved from www.diva-portal.org/smash/get/diva2:360992/FULLTEXT01
- González-Baixauli, B., Laguna, M., & Crespo, Y. (2004). *Product line requirements based on goals features and use cases*. Paper presented at the International Workshop on Requirements Reuse in System Family Engineering (IWREQFAM). Retrieved from http://giro.infor.uva.es/Publications/2004/GLC04/bruno-ws_icsr_v3.pdf
- Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J.-J., Nielsen, H., Karmarkar, A., & Lafon, Y. (2007). *SOAP version 1.2 part 2: Adjuncts*. Retrieved from <http://www.w3.org/TR/soap12-part2/>

- Gumbo, S., Terzoli, A., & Thinyane, M. (2013). Living labs as South Africa's enabler for ICT services creation: The siyakhula living lab experience. *Proceedings of the 16th Annual Southern Africa Telecommunication Networks and Applications Conference (SATNAC)* (pp. 1–5). Spier Wine Estate, Stellenbosch, Western Cape, South Africa.
- Hart, S. G., & Staveland, L. E. (1988). *Development of NASA-TLX: Results of empirical and theoretical research*. Retrieved from http://archive.org/details/nasa_techdoc_20000004342
- Hassanzadeh, A., Namdarian, L., & Elahi, S. (2011). Developing a framework for evaluating service-oriented architecture governance (SOAG). *Knowledge-Based Systems*, 24(5), 716–730.
- Heidenreich, F., Sánchez, P., Santos, J., Zschaler, S., Alférez, M., Araújo, J., Rashid, A. (2010). Relating feature models to other models of a software product line. In S. Katz, M. Mezini, & J. Kienzle (Eds.), *Transactions on Aspect-Oriented Software Development VII* (pp. 69–114). Springer Berlin Heidelberg.
- Helferich, A., Herzwurm, G., Jesse, S., & Mikusz, M. (2007). Software product lines, service-oriented architecture and frameworks: Worlds apart or ideal partners? *Proceedings of the 2nd International Conference on Trends in Enterprise Application Architecture* (pp. 187–201). Berlin, Heidelberg: Springer-Verlag.
- Hevner, A., March, S., Park, J., & Ram, S. (2004). Design science in information systems research. *Management Information Systems Quarterly*, 28(1), 75–105.
- Hofstee, E. (2006). *Constructing a good dissertation: A practical guide to finishing a Masters, MBA or PhD on schedule*. Sandton, South Africa: Exactica.
- Huhns, M. N., & Singh, M. P. (2005). Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1), 75–81. doi:10.1109/MIC.2005.21
- Hunaity, M. A. R. (2008). Towards an efficient quality based web service discovery framework. *Proceedings of the IEEE Congress on Services - Part I* (pp. 261–264). Honolulu, HI: IEEE Computer Society.

- International Organization for Standardization. (1998). *ISO 9241-11: Ergonomic requirements for office work with visual display terminals (VDTs) — Part 11: Guidance on usability*. Retrieved from <https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-1:v1:en>
- International Organization for Standardization. (2008). *ISO/NP 9241-11: Ergonomics of human-system interaction - Part 11: Usability: Definitions and concepts*. Retrieved from http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=63500
- Jarzabek, S., Yang, B., & Yoeun, S. (2006). Addressing quality attributes in domain analysis for product lines. *IEE Proceedings - Software*, 153(2), 61–73. doi:10.1049/ip-sen:20050008
- Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E., & Huh, M. (1998). FORM: A feature-oriented reuse method with domain specific reference architectures. *Annals of Software Engineering*, 5(1), 143–168. doi:10.1023/A:1018980625587
- Kang, K. C., & Lee, H. (2013). Variability modeling. In R. Capilla, J. Bosch, & K. C. Kang (Eds.), *Systems and Software Variability Management* (pp. 25–42). Springer Berlin Heidelberg.
- Kang, K. C., Lee, J., & Donohoe, P. (2002). Feature-oriented product line engineering. *IEEE Software*, 19(4), 58–65. doi:10.1109/MS.2002.1020288
- Kang, K., Cohen, S., Hess, J., Nowak, W., & Peterson, S. (1990). *Feature-oriented domain analysis (FODA) feasibility study* (Technical Report No. CMU/SEI-90-TR-021). Pittsburgh, PA, Carnegie Mellon University: Software Engineering Institute.
- Kannan, P. K., & Proenca, J. F. (2008). Design of service systems under variability: Research issues. *Proceedings of the 41st Annual Hawaii International Conference on System Sciences* (pp. 116–126). Waikoloa, HI: IEEE Computer Society.
- Kanneganti, R., & Chodavarapu, P. (2008). *SOA security*. Greenwich, CT: Manning Pubns Co.
- Kim, Y., & Doh, K.-G. (2008). Adaptable web services modeling using variability analysis. *Proceedings of the Third International Conference on Convergence and Hybrid*

- Information Technology, ICCIT '08, volume 1* (pp. 700–705). Busan: IEEE Computer Society.
- Kim, Y.-G., Lee, S. K., & Jang, S.-B. (2011). Variability management for software product-line architecture development. *International Journal of Software Engineering and Knowledge Engineering*, 21(07), 931–956. doi:10.1142/S0218194011005542
- Krafzig, D., Banke, K., & Slama, D. (2005). *Enterprise SOA: Service-oriented architecture best practices*. Indianapolis, IN: Prentice Hall Professional Technical Reference.
- Kramer, J. (2007). Is abstraction the key to computing. *Communications of the ACM*, 50 (4), 36–42. doi:10.1145/1232743.1232745
- Kulkarni, N., & Dwivedi, V. (2008). The role of service granularity in a successful SOA realization: A case study. *Proceedings of the 2008 IEEE Congress on Services - Part I* (pp. 423–430). Washington, DC, USA: IEEE Computer Society.
- Lee, J., & Muthig, D. (2006). Feature-oriented variability management in product line engineering. *Communications of the ACM*, 49(12), 55–59. doi:10.1145/1183236.1183266
- Lee, K., Kang, K. C., & Lee, J. (2002). Concepts and guidelines of feature modeling for product line software engineering. *Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools* (pp. 62–77). London, UK: Springer-Verlag.
- Lewis, G., Smith, D. B., & Kontogiannis, K. (2010). *A research agenda for service-oriented architecture (SOA): Maintenance and evolution of service-oriented systems* (Technical Report No. CMU/SEI-2010-TN-003). Pittsburgh, PA, Carnegie Mellon University: Software Engineering Institute.
- Lewis, J. R. (1995). IBM computer usability satisfaction questionnaires: Psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1), 57–78. doi:10.1080/10447319509526110

- Liang, H., Sun, W., Zhang, X., & Jiang, Z. (2006). A policy framework for collaborative web service customization. In *Second IEEE International Workshop on Service-Oriented System Engineering, SOSE '06* (pp. 197–204). Shanghai: IEEE Computer Society.
- Linden, F. J. van der, Schmid, K., & Rommes, E. (2007). *Software product lines in action: The best industrial practice in product line engineering*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.
- Lindgaard, G. (1994). *Usability testing and system evaluation*. London: Chapman & Hall.
- Lu, J., & Yu, Y. (2007). Web service search: Who, when, what, and how. *Proceedings of Web Information Systems Engineering* (pp. 284–295). Nancy, France: Springer Berlin Heidelberg.
- Luisa, M., Mariangela, F., & Pierluigi, I. (2004). Market research for requirements analysis using linguistic tools. *Requirements Engineering*, 9(1), 40–56. doi:10.1007/s00766-003-0179-8
- Mairiza, D., Zowghi, D., & Nurmuliani, N. (2010). An investigation into the notion of non-functional requirements. *Proceedings of the 2010 ACM Symposium on Applied Computing* (pp. 311–317). New York, NY, USA: ACM.
- Manes, A. (2005, July 6). *The elephant has left the building*. InformationWeek. Retrieved from <http://www.informationweek.com/software/business-intelligence/the-elephant-has-left-the-building/164301126>
- Medeiros, F. M., Almeida, E. S. de, & Meira, S. R. de L. (2010). Designing a set of service-oriented systems as a software product line. In *2013 VII Brazilian Symposium on Software Components, Architectures and Reuse* (pp. 70–79). Los Alamitos, CA, USA: IEEE Computer Society.
- Moore, J. (2006, February 7). *Primer: Web services governance*. Baseline. Retrieved from <http://www.baselinemag.com/c/a/Tools-Primers-hold/Primer-Web-Services-Governance>

- Muller, P. A., Fondement, F., Baudry, B., & Combemale, B. (2012). Modeling modeling modeling. *Software & Systems Modeling*, 11(3), 347–359. doi:10.1007/s10270-010-0172-x
- Nassar, M., Anwar, A., Ebersold, S., Elasri, B., Coulette, B., & Kriouile, A. (2009). Code generation in VUML profile: A model driven approach. *Proceedings of the IEEE/ACS International Conference on Computer Systems and Applications (AICCSA 2009)* (pp. 412–419). Rabat: IEEE Computer Society.
- Nguyen, T., & Colman, A. (2010). A feature-oriented approach for web service customization. *Proceedings of the 2010 IEEE International Conference on Web Services (ICWS)* (pp. 393–400). Miami, FL: IEEE Computer Society.
- Nielsen, J. (2000). *Why You Only Need to Test with 5 Users* (Research Report). Retrieved from <http://www.biology.emory.edu/research/Prinz/Cengiz/cs540-485-FA12/resources/5userTesting.pdf>
- Nielsen, J., & Landauer, T. K. (1993). A mathematical model of the finding of usability problems. *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems* (pp. 206–213). New York, NY, USA: ACM.
- Oracle. (2007). *Getting started with use case modeling* (White paper) (pp. 1–19). Retrieved from <http://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf>
- Papazoglou, M. P. (2003). Service-oriented computing: concepts, characteristics and directions. *Proceedings of the Fourth International Conference on Web Information Systems Engineering, WISE '03* (pp. 3–12). Washington, DC, USA: IEEE Computer Society.
- Papazoglou, M. P., & Georgakopoulos, D. (2003). Service-oriented computing: Introduction. *Communications of the ACM*, 46(10), 25–28.
- Papazoglou, M. P., & Heuvel, W. V. D. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4), 412–442. doi:10.1504/IJWET.2006.010423

- Papazoglou, M. P., & Heuvel, W.-J. (2007). Service-oriented architectures: Approaches, technologies and research issues. *The VLDB Journal*, 16(3), 389–415. doi:10.1007/s00778-007-0044-3
- Papazoglou, M. P., Traverso, P., Dustdar, S., & Leymann, F. (2007). Service-oriented computing: State of the art and research challenges. *Computer*, 40(11), 38–45. doi:10.1109/MC.2007.400
- Papazoglou, M., Traverso, P., Dustdar, S., & Leymann, F. (2008). Service-oriented computing: A research roadmap. *International Journal of Cooperative Information Systems*, 17(02), 223. doi:10.1142/s0218843008001816
- Parastatidis, S., Webber, J., Woodman, S., Kuo, D., & Greenfield, P. (2005). *An introduction to the SOAP service description language* (White paper, version 1.3). Retrieved from <http://savas.me/ssdl/docs/v1.3/html/SSDL%20whitepaper%20v1.3.html>
- Pastore, S. (2008). The service discovery methods issue: A web services UDDI specification framework integrated in a grid environment. *Journal of Network and Computer Applications*, 31(2), 93–107. doi:10.1016/j.jnca.2006.05.001
- Pautasso, C., & Wilde, E. (2009). Why is the Web loosely coupled?: A multi-faceted metric for service design. *Proceedings of the 18th International Conference on World Wide Web* (pp. 911–920). New York, NY, USA: ACM.
- Perrouin, G., Klein, J., Guelfi, N., & Jézéquel, J.-M. (2008). Reconciling automation and flexibility in product derivation. *Proceedings of the 12th International Software Product Line Conference* (pp. 339–348). Washington, DC, USA: IEEE Computer Society.
- Pohl, K., Bockle, G., & Linden, F. J. van der. (2005). *Software product line engineering: Foundations, principles and techniques*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.
- Sarang, P., Jennings, F., Juric, M., & Loganathan, R. (2007). *SOA approach to integration: XML, Web services, ESB, and BPEL in real-world SOA projects*. Packt Publishing.

- Schmettow, M. (2012). Sample size in usability studies. *Communications of the ACM*, 55(4), 64–70. doi:10.1145/2133806.2133824
- Schmid, K., & John, I. (2004). A customizable approach to full lifecycle variability management. *Science of Computer Programming*, 53(3), 259–284. doi:10.1016/j.scico.2003.04.002
- Schmid, K., Rabiser, R., & Grünbacher, P. (2011). A comparison of decision modeling approaches in product lines. *Proceedings of the 5th Workshop on Variability Modeling of Software-Intensive Systems* (pp. 119–126). New York, NY, USA: ACM.
- Schoneveld, A., & Philip, M. (2007, February). *The service-oriented club*. Retrieved from <http://www.slideshare.net/mphilip/the-service-oriented-club>
- Selic, B. (2003). The pragmatics of model-driven development. *IEEE Software*, 20(5), 19–25. doi:10.1109/MS.2003.1231146
- Semmak, F., Laleau, R., & Gnaho, C. (2009). Supporting variability in goal-based requirements. *Proceedings of the Third IEEE International Conference on Research Challenges in Information Science* (pp. 271–280). Fès, Morocco. doi:10.1109/RCIS.2009.5089287
- Shaker, P. (2010). Feature-oriented requirements modeling. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering, volume 2* (pp. 365–368). New York, NY, USA: ACM.
- Shaw, M., & Garlan, D. (1996). *Software architecture: perspectives on an emerging discipline*. Upper Saddle River, N.J.: Prentice Hall.
- Sibiya, M., Jembere, E., Xulu, S., & Adigun, M. (2008). A web services based e-commerce business model for resource constrained SMMEs. *Proceedings of the 11th Annual Southern Africa Telecommunication Networks and Applications Conference (SATNAC)* (pp. 1–6). Wild Coast Sun, Wild Coast.
- Siegmund, N., Rosenmüller, M., Kuhlemann, M., Kästner, C., Apel, S., & Saake, G. (2012). SPL conqueror: Toward optimization of non-functional properties in software product lines. *Software Quality Journal*, 20(3-4), 487–517. doi:10.1007/s11219-011-9152-9

- Sinnema, M., Deelstra, S., Nijhuis, J., & Bosch, J. (2006). Modeling dependencies in product families with COVAMOF. *Proceedings of the 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems, ECBS '06* (pp. 299–307). Postdam, Germany: IEEE Computer Society.
- Skerrett, I. (2012, June). *Eclipse survey 2012 report*. Technology. Retrieved from <http://www.slideshare.net/IanSkerrett/eclipse-survey-2012-report-final>
- Sochos, P., & Riebisch, M. (2004). Feature-oriented development of software product lines: Mapping feature models to the architecture. *Proceedings of the 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World, Net.ObjectDays 2004* (pp. 138–152). Erfurt, Germany: Springer Berlin Heidelberg.
- Srinivasan, N., Paolucci, M., & Sycara, K. (2006). Semantic web service discovery in the OWL-S IDE. *Proceedings of the 39th Annual Hawaii International Conference on System Sciences, HICSS '06* (p. 109b–109b). doi:10.1109/HICSS.2006.431
- Steinberg, W. J. (2011). *Statistics alive!* Thousand Oaks, Calif.: Sage Publications.
- Stollberg, M. (2008). *Scalable semantic web service discovery for goal-driven service-oriented architectures* (Doctoral dissertation). Retrieved from <http://www.michael-stollberg.de/phd/docs/phdthesis-mstollberg.pdf>
- Stollberg, M., & Muth, M. (2009). Service customization by variability modeling. *Proceedings of the International Conference on Service-Oriented Computing/ Service Wave* (pp. 425–434). Stockholm, Sweden: Springer-Verlag Berlin Heidelberg.
- Stollberg, M., & Muth, M. (2010). Efficient business service consumption by customization with variability modeling. *Journal of Systems Integration*, 1(3), 17–32.
- Sun, C., Rossing, R., Sinnema, M., Bulanov, P., & Aiello, M. (2010). Modeling and managing the variability of web service-based systems. *Journal of Systems and Software*, 83(3), 502–516. doi:10.1016/j.jss.2009.10.011

- Svahnberg, M., van Gurp, J., & Bosch, J. (2005). A taxonomy of variability realization techniques: Research articles. *Software - Practice & Experience*, 35(8), 705–754.
- Tibco. (2011). *The Role of Governance in Ensuring SOA Success* (White paper). Retrieved from http://www.tibco.com/multimedia/wp-role-of-governance-ensuring-soa-success_tcm8-8998.pdf
- Tosic, V. (2004). *Service offerings for XML web services and their management applications* (Doctoral dissertation). Retrieved from http://atp-webproxy1.it.nicta.com.au/__data/assets/pdf_file/0010/21223/TosicPhDThesis-Final.pdf
- Tosic, V., Lutfiyya, H., & Tang, Y. (2006). Web service offerings language (WSOL) support for context management of mobile/embedded XML web services. *Proceedings of the International Conference on Internet and Web Applications and Services/Advanced International Conference on Telecommunications, AICT-ICIW '06* (pp. 156–161). doi:10.1109/AICT-ICIW.2006.208
- Tosic, V., Patel, K., & Pagurek, B. (2002). WSOL — Web service offerings language. In C. Bussler, R. Hull, S. McIlraith, M. E. Orlowska, B. Pernici, & J. Yang (Eds.), *Web Services, E-Business, and the Semantic Web* (pp. 57–67). Springer Berlin Heidelberg.
- Trkman, D. P., Kovačič, D. A., & Popovič, D. A. (2011). SOA adoption phases. *Business & Information Systems Engineering*, 3(4), 211–220. doi:10.1007/s12599-011-0168-2
- Trujillo, S., Batory, D., & Diaz, O. (2007). Feature-oriented model driven development: A case study for portlets. *Proceedings of the 29th International Conference on Software Engineering* (pp. 44–53). Washington, DC, USA: IEEE Computer Society.
- Valipour, M. H., Amirzafari, B., Maleki, K. N., & Daneshpour, N. (2009). A brief survey of software architecture concepts and service-oriented architecture. *Proceedings of the 2nd IEEE International Conference on Computer Science and Information Technology, ICCSIT '09* (pp. 34–38). Beijing: IEEE Computer Society.
- Virzi, R. A. (1992). Refining the test phase of usability evaluation: How many subjects is enough? *Human Factors*, 34(4), 457–468.

- Voelter, M. (2009). Using domain specific languages for product line engineering. *Proceedings of the 13th International Software Product Line Conference* (pp. 329–329). Pittsburgh, PA, USA: Carnegie Mellon University.
- W3C. (2007). *Web services description language (WSDL) version 2.0 part 1: Core language*. Retrieved from <http://www.w3.org/TR/wsdl20/>
- Williams, B. J., & Carver, J. C. (2010). Characterizing software architecture changes: A systematic review. *Information and Software Technology*, 52(1), 31–51.
- Woolf, B. (2007). *Introduction to SOA governance*. CT316. Retrieved April 3, 2014, from <http://www.ibm.com/developerworks/library/ar-servgov/#servver>
- Ying, H., Wu, Y., & Liu, F. (2010). Research on the SOA-based service granularity control. *Proceedings of the Second International Conference on Information Technology and Computer Science (ITCS)* (pp. 443–446). Kiev: IEEE Computer Society.
- Ying, M. (2006). Linguistic quantifiers modeled by Sugeno integrals. *Artificial Intelligence*, 170(6–7), 581–606. doi:10.1016/j.artint.2006.02.001
- Zadeh, A., Mukhtar, M., & Sahran, S. (2012). A comparative study of enterprise resource planning vs service-oriented architecture in small medium enterprises. *Journal of Computer Science*, 8(8), 1389–1396. doi:10.3844/jcssp.2012.1389.1396

APPENDICES

Appendix A: Consent Form



INFORMATION AND INFORMED CONSENT FORM

RESEARCHER'S DETAILS	
Title of the research project	A Model-Based Service Customization Framework for Managing Consumer Variability in SOA
Principal investigator	Sandile Wilmoth Dlamini
Contact telephone number (research office)	035 902 6012
A. DECLARATION BY OR ON BEHALF OF THE PARTICIPANT	
I, the participant and the undersigned	(full names)
A.1. HEREBY CONFIRM AS FOLLOW:	
I, the participant was invited to participate in the above-mentioned research project	
that is being undertaken by	Sandile Wilmoth Dlamini
from	Department of Computer Science
of the University of Zululand	
A.2. THE FOLLOWING ASPECTS HAVE BEEN EXPLAINED TO ME (THE PARTICIPANT)	
Aim	<p>The investigator is studying how the complexities involved in consumers' service customization processes can be minimized, in order to support non-ICT experts, without good background knowledge in web service related technologies to customize software services.</p> <p>The information will be used for research purposes.</p>
Risks	I understand that there are no risks involved in participating in this process.
Confidentiality	I am fully aware that my identity will be known and only be visible to me and the researcher. It will not be revealed in any discussion, description or scientific publications.
Access to findings	I am also aware that any new information that develops during this process would be shared as follows: In a <i>dissertation, journal or conference article.</i>

Voluntary participation / refusal / discontinuation	My participation is voluntary	Y	N	
	My decision whether or not to participate will in no way affect my present or future career/employment/lifestyle	T	F	
No pressure was exerted on me to consent to participate and I understand that I have the right to withdraw at any stage without penalization.				
Participation in this study will not result in any additional cost to me.				
I HEREBY VOLUNTARILY CONSENT TO PARTICIPATE IN THE ABOVE-MENTIONED RESEARCH PROJECT				
Signed at: (place) on the (date) of (month) 20 (year)				
Signature	Signature of witness:			
	Full name(s) of witness:			

Appendix B: Background Questionnaire (Pre-Experiment)

PARTICIPANT DETAILS					
1.	Name				
2.	Gender	Male		Female	
3.	Age	18-20	21-25	26-30	31-35
4.	Occupation	Student	Business Sector	Education Sector	ICT Sector
5.	Years of Computer Experience	< 1	1-2	3-4	5+
6.	Years of Internet Experience	< 1	1-2	3-4	5+
7.	Years of Software Services Experience	< 1	1-2	3-4	5+
8.	Have you used Web services tools before?	Yes		No	


Appendix C: Usability Questionnaire (Post-Experiment)


FreeCust Evaluation							
Section A: Cognitive Load							
1. Mental Demand: How mentally demanding were the tasks?	1	2	3	4	5	6	7
2. Physical Demand: How physically demanding were the tasks?	1	2	3	4	5	6	7
3. Performance: How successful were you in accomplishing what you were asked to do?	1	2	3	4	5	6	7
4. Effort: How hard did you have to work to accomplish your level of performance?	1	2	3	4	5	6	7
5. Frustration: How insecure, discouraged, irritated, stressed, and annoyed were you?	1	2	3	4	5	6	7
Section B: Overall Satisfaction							
1. I am satisfied with how easy it is to follow and perform the task of the notion of FreeCust.	1	2	3	4	5	6	7
2. I am satisfied with the GUI-flavored view of software service customization options (i.e., service capabilities).	1	2	3	4	5	6	7
3. Overall, it was easy to understand and use the GUI-flavored view in order to customize a software service.	1	2	3	4	5	6	7
Section C: Usability							
1. I was able to find the live match score service in the XMethods service repository.	1	2	3	4	5	6	7
2. I was able to retrieve the software service description (i.e., the feature model).	1	2	3	4	5	6	7
3. I was able to comprehend and formulate a service customization (i.e., selected and disabled all features that I was interested on).	1	2	3	4	5	6	7
4. I was able to send a software service customization request.	1	2	3	4	5	6	7

Appendix D: Detailed Usability Evaluation Instructions or Task List

Task 1: Search for the live match score software service in the XMethods service directory.	
1.1	Open a web browser and go to the XMethods web site (http://www.xmethods.net/ve2/index.po).
1.2	Click on the link “FULL LIST” to view all the software services available in the XMethods directory, and look for “Live Match Score Service” under the “Service Name” column. Once located,
Task 2: Retrieve the software service feature model (i.e., the Live_Match_Score_Software_Service.fmp) file.	
2.1	Download the Live_Match_Score_Software_Service.fmp file to your computer system.
Task 3: Import the software service feature model file in Eclipse, where the FreeCust GUI-flavored implementation is running.	
3.1	Start Eclipse and create a general project.
3.2	Import the Live_Match_Score_Software_Service.fmp file into the project.
Task 4: Select and disable all the required and unnecessary features.	
4.1	Double-click on the Live_Match_Score_Software_Service.fmp file will open it in the content panel of Eclipse.
4.2	Right click on the node “My Feature Model” in the content panel, then select “Expand Subtree” to expand the service feature model.
4.3	Scroll down to see the node “Configuration 1 of Live_Match_Score_Software_Service (1 configuration(s)).
4.4	Once opened - to select a feature, click on the box next to it. To disable a feature, hold the “Ctrl” key while clicking on the box. Repeat clicking to remove the selection.
Task 5: Send customization request.	
5.1	Right click on the node “Configuration 1 of Live_Match_Score_Software_Service” and select “Send Customization Request”.

Appendix E: XMethods Demo Service Terms and Conditions

 www.xmethods.net/v2/demoguidelines.html

 **XMETHODS**

XMethods Demo Services - Guidelines for Use

The "XMethods Demo Services" are sample services provided by XMethods to help developers experiment. There is no charge for their use. However, please take note of the following.

1. Our machine resources and bandwidth are limited. Please do not abuse these services through excessive invocation. Any such abuse may lead to blocking of your IP address without warning.
2. Any of these services may be discontinued at any time without warning.
3. While we do our best to keep the services running, there are absolutely no uptime guarantees.
4. We do not warrant the accuracy of the information provided by these services. Use at your own risk.
5. Because of the points listed above, we strongly discourage the use of these services for "real" production purposes, such as integration into a portal.
6. Use in demonstrations and reference in published material (articles, books, etc) is allowed, as long as the points listed above are noted. We do ask that you provide attribution and a link to the site, www.xmethods.net

If you have any questions about any of the above, please contact us at:

support at xmethods.net

Thanks!

Appendix F: Live Match Score Service Feature Model and Consumer's Feature Configurations Request in XML

```
<?xml version="1.0" encoding="UTF-8"?>
<feature-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.tdq-seville.info/benavides/featuremodelling/feature-model.xsd">
  <feature min="1" max="1" name="Live_Match_Score_Software_Service" type="NONE" id="liveMatchScoreSoftwareService">
    <feature min="1" max="1" name="Update Information" type="NONE" id="updateInformation">
      <featureGroup min="1" max="1" id="group">
        <feature min="0" max="1" name="After60sec" type="NONE" id="after60sec">
        </feature>
        <feature min="0" max="1" name="After10min" type="NONE" id="after10min">
        </feature>
      </featureGroup>
    </feature>
    <feature min="1" max="1" name="Avalability" type="NONE" id="avalability">
      <featureGroup min="1" max="1" id="group3">
        <feature min="0" max="1" name="At_least_70%" type="NONE" id="atleast70">
        </feature>
        <feature min="0" max="1" name="At_most_99%" type="NONE" id="atmost99">
        </feature>
      </featureGroup>
    </feature>
    <feature min="0" max="1" name="Security" type="NONE" id="security">
      <featureGroup min="1" max="1" id="group4">
        <feature min="0" max="1" name="Low" type="NONE" id="low">
        </feature>
        <feature min="0" max="1" name="Medium" type="NONE" id="medium">
        </feature>
        <feature min="0" max="1" name="High" type="NONE" id="high">
        </feature>
      </featureGroup>
    </feature>
  </feature>
  <requires name="include-1" feature="After60sec" requires="At_most_99%" />
</feature-model>

<?xml version="1.0" encoding="UTF-8"?>
<feature-model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://www.tdq-seville.info/benavides/featuremodelling/feature-model.xsd">
  <feature name="Live_Match_Score_Software_Service" type="NONE" id="liveMatchScoreSoftwareService">
    <feature name="Update Information" type="NONE" id="updateInformation">
      <feature name="After60sec" type="NONE" id="after60sec">
      </feature>
    </feature>
    <feature name="Avalability" type="NONE" id="avalability">
      <feature name="At_most_99%" type="NONE" id="atmost99">
      </feature>
    </feature>
    <feature name="Security" type="NONE" id="security">
      <feature name="High" type="NONE" id="high">
      </feature>
    </feature>
  </feature>
</feature-model>
```