Requirement Analysis Technique for Configurable Platform: Case Study

by

Antonio Misaka, M.Sc.

A thesis submitted to the Faculty of Graduate and Postdoctoral Affairs in partial fulfillment of the requirements for the degree of

Master of Applied Science in Technology Innovation Management

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario, Canada, K1S 5B6

January 2013

Your file Votre référence
ISBN: 978-0-494-94258-1

Our file Notre référence
ISBN: 978-0-494-94258-1

The undersigned recommend to
the Faculty of Graduate and Postdoctoral Affairs
acceptance of the thesis

Requirement Analysis Technique for Configurable Platform: Case Study

submitted by

Antonio Misaka, M.Sc.

In partial fulfillment of the requirements for
the degree of Master of Applied Science in Technology Innovation Management

---

Director, Tony Bailetti, Technology Innovation Management

Thesis Supervisor, Michael Weiss

Carleton University
January 2013

Abstract

This thesis proposes a technique to map customer requirements to a platform configuration at the early stage of software requirements analysis and performs a case study with the proposed technique. Here a configuration is a selection of platform components that reduces the gap between requirements analysis and software development stages. The development of this solution indicates it is possible to create configurations using goal-based and scenario-based notations offered by User Requirements Notation (URN) modelling and analysis.

Acknowledgements

I give thanks to Almighty God for giving me the strength and enlightenment to develop this research. I acknowledge Prof. Bailetti and Prof. Weiss for providing me the guidance in experimenting ideas during the Carleton Entrepreneurs project and for my business opportunity in developing a foundation to create a configurable software platform. Thank you to Ludovico Prattico and Leonard de Baets during our discussion phase about the process of creation of a keystone software platform, and to Gunter Mussbacher for helping me out with User Requirements Notation (URN) and jUCMNav modelling process. Lastly, I would like to thank my wife, Sawaco, for her unconditional support all these years.

Table of Contents

List of Tables

List of Illustrations

List of Appendices

Glossary

CMS: Content Management System
COPE+: Customer Oriented Product Evolution
CRM: Customer Relationship Management
FAST: Family-oriented Abstraction, Specification and Translation
FOOM: Feature-based Object Oriented Modelling
GORE: Goal-Oriented Requirements Engineering
GRL: Goal-oriented Requirement Language
ITU-T: ITU Telecommunication Standardization Sector
ITU: International Telecommunication Union
MRAM: Method for Requirements Authoring Management
NFR: Non-Functional Requirements
PLUSS: Product Line Use case modelling for Systems and Software engineering
RE: Requirements Engineering
SPL: Software Product Line
SPLE: Software Product Line Engineering
UCM: Use Case Map
UML: Unified Modelling Language
URN: User Requirements Notation

Chapter 1: Introduction

The motivation for this research is to uncover how to link the configuration of a configurable software platform with customer requirements needs. Illustration 1 shows that there is gap between the requirements and software platform implementation phases, creating a challenge for software developers to match user satisfaction, increased quality, and lower cost and time in delivering a configurable software platform.

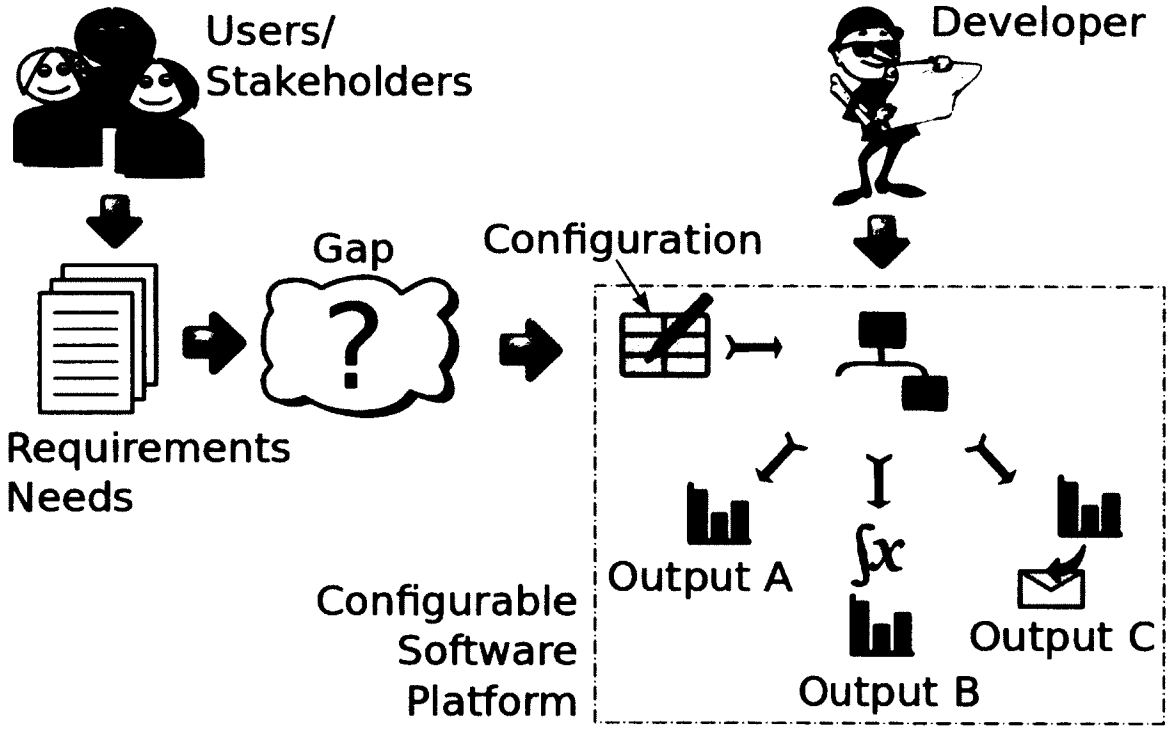


Illustration 1: Motivation for this research to develop a configurable software platform.

Most solutions for software providing service-oriented systems and customization services make some considerable restrictions for the service buyers regarding the business means that the service buyer has to adapt to the service offered by the platform owner or one has to buy a customized version for one's business process, resulting in a

more expensive solution.

Although building a customizable software platform is well known for software developers, there are many examples of platforms such as SugarCRM[1], Drupal[2] and WordPress[3]. The unsolved question is how to build a configurable and customizable software platform which may satisfy more than one stakeholder requirement at the same time and uncovering the potential configuration for the software platform to satisfy these requirements.

Configurable software platform, for the context of this research, is a software framework composed of software components and components library set. It loads a configuration for every request identified by a recorded key identification and runs a sequence of activities for this specific key identification.

1.1 Objective

There is a gap between the requirements engineering process and software development process, current proposals to bridge the gap include Use Case Maps (UCM) (Amyot & Mussbacher, 2001; Amyot, 2003), Aspect Oriented Software Development (Dahiya & Sachdeva, 2006), Goal-Based Approach and Model Bridging (Galster et al., 2006), Service Diagram (Gruler, Harhurin, & Hartmann, 2007), UML Sequence Diagram using Constraint Automata (Arbab & Meng, 2008), and Decision Models (Arboleda, Casallas, & Royer, 2009). Although we can find several approaches, there was an open question

1 SugarCRM is a software company producing customer relationship management system. http://www.sugarcrm.com
2 Drupal is an open-source content management framework. http://www.drupal.org
3 WordPress is an open-source blogging tool and content management system. https://wordpress.org

related to how to find a configuration for software platform in the early stages resulting in increasing software component reuse, development and maintenance costs reduction, increasing customer satisfaction and quality of product and reducing time to the market.

This research provides the answer for the following research question: How can we select a configuration for a software platform that meets customer requirements during the early stages of requirements engineering process?

The objective of this research is to develop and test a technique to elicit requirements from customers of different platforms interested in buying or using similar services and to show how to model these requirements to create a configurable software platform at the domain level. The service running in the platform should satisfy each customer requirement using its own configuration.

As a result it provides a technique to bridge the gap between the Requirements Analysis (RA) and Software Development stage (implementation) to create a configuration for a software platform during the early stage of Requirements Engineering Process.

1.2 Relevance

This work is relevant for software developers, academics, software platform owners and customers. For this research, the focus is a software platform that involves various users and it may need to provide a set of flexible functionality to satisfy a set of shared requirements requested by different stakeholders through the configurable software platform.

There are two types of software developers in this context: one can identify how each requirement from different stakeholders may affect or impact the design of the software platform and create a solution that is easier to configure and adapt for each stakeholder's requirement. The other type are software integrators using customer requirements to integrate existing platforms creating a configurable software platform in doing the right software configuration and matching their stakeholder's need.

Academics would be interested in evaluating the results of this research regarding to how to create a configurable software platform from the early stages of the requirements analysis. They can propose improvements and adjustments both to this technique which is developed to model requirements to create a platform configuration and to existing software development methods that could be merged with this technique.

Software platform owners might be interested in the results to potentially reduce their operational and maintenance costs, time to the market and improving quality of offered services. They would be able to provide a more attractive solution for their customers who are buying the services offered to them. As a consequence, they could be more profitable through costs reduction.

Customers of the software platform will be interested in finding a solution that would meet better their needs for their business operations. That reduces their need to be adapted in terms of business process to the offered services by the software platform.

1.3 Deliverables

This research is providing a technique with a set of steps that may reduce the gap between the requirements analysis and the implementation of the software development stage. It is based on existing methodology and software tool. Applying this set of steps will allow to uncover and create an initial configuration to meet the needs from several stakeholders in the early stages of requirements analysis.

The approach to develop this research project is supported starting from an initial set of requirements from various stakeholders in a similar set of operations that should be delivered through the software platform. Each customer requirement necessitates a specific result to satisfy the requirements gathered during the RA phase.

Illustration 2 shows a partial software platform architecture presenting the main components for this research. Configuration Tables, Glue Code and Components Library Set are the key elements to make this software platform perform the offered services. They are designed to handle distinct configuration tables and satisfy requirements from Subscribers 1 and 2. The Subscriber is the same as software platform customer. This illustration shows that application A triggers Service Z and application B triggers Services X and Z, what happens on the fly when the Glue Code loads each subscriber configuration. Each service is built from a selected set of software components and runs accordingly to the loaded configuration from a Subscriber or Customer.

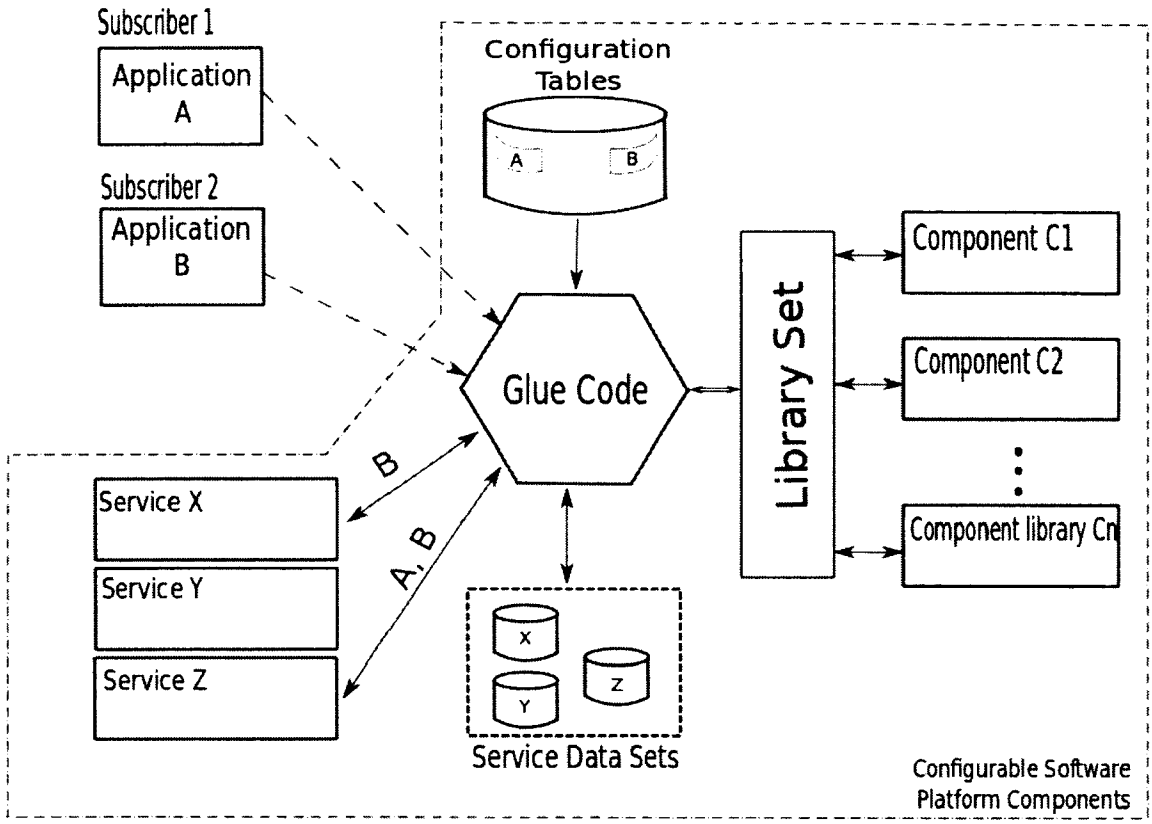


Illustration 2: Partial Architecture for Software Platform

A requirement modelling process is going to be applied to support the requirements analysis and it is based on an User Requirements Notation (URN). The URN method composed of Goal-oriented Requirement Language (GRL) and User Case Maps (UCM), is a recommended standard (Z.151) from International Telecommunication Union-Telecommunication (ITU-T) and has been used by the telecommunication industry for more than 10 years. For this research approach, URN was selected during the proof of concept and demonstrated that we could apply it successfully to answer the research question. The modelling process creates a set of GRL graphs and UCM diagrams. GRL graphs are representing sets of goals from each stakeholder involved in a software

platform and allow to identify the strategies to satisfy each non-functional requirement (NFR). UCM diagrams are representing the process/work-flow to meet the needs of each customer functional requirements. Then a set of activities are performed to uncover the configuration at the domain level for the software platform mapping the requirements to the platform configuration.

1.4 Contributions

This proposed solution is intended to focus on user-driven aspects instead of market-driven or technology-driven aspects and provides a technique to identify and create a configuration for software platform during the early stages based on process or workflow parameters.

A software platform increases its complexity when several stakeholders are requesting a similar functionality to satisfy their requirements and they do not agree to use one's functionality. The existing approaches to elicit requirements and design an application for a software platform do not fit together properly based on our experience with the software development process applied in one Technology Innovation Management project. It is a software platform developed for Carleton Entrepreneurs program which combines Content Management System (CMS), Customer Relationship Management (CRM) and glue code.

During this software development process, we identified that for each stakeholder of this software platform there is an analysis process that needs to be considered in terms of the

impact from one stakeholder's requirement over the other stakeholders' requirements. In a context composed of two or more stakeholders requesting similar outputs with few differences, it might create a more complex analysis based on the conventional approach eliciting requirements and designing the software platform over an initial configuration for the software application during the early stages of requirements analysis.

The proposed technique provides a solution to identify and create a configurable software platform from stakeholder requirements gathered at the domain level. As a result, customers can have quick access to a prototype and verify whether they want or do not want the configured process or workflow. Furthermore, this approach allows the Software Developer to document the system behaviour with a close relationship at the stakeholder requirements. A configurable software platform may provide a proper way to maintain or change part of the system because it may reduce the complexity of the software design.

1.5 Organization

This thesis is organized as follow, Chapter 2 is the literature review and lessons learnt, Chapter 3 presents the research method, Chapter 4 provides additional background to a better understanding of the selected methods and tools, Chapter 5 describes the process used to develop the solution for the research question, Chapter 6 provides the validation for the process presented in the previous chapter, Chapter 7 presents discussions of the results and Chapter 8 describes the conclusions, limitations and future research.

Chapter 2: Literature Review

This chapter has relevant information from the literature review related to the software development process from requirements gathering and analysis to software design and implementation. This research focus is mainly on user and customer requirements to build a configurable software platform. The information presented in this chapter provides the support for the proposed research answering the research question presented in the Chapter 1.

During the system development cycle, the produced documents can easily be communicated among both non-technical stakeholders and the engineering disciplines. This situation can be transferred for software development cycle, involving both users and customers, and software developers (Eriksson, 2007).

Software Requirements Engineering and Software Architecture affect the success and quality of software projects during the software development cycle (Galster et al., 2006). In the software development cycle, there is a gap between Requirements Engineering and Software Architecture phases[4]. Galster et al. (2006) says that although some attention is given to bridge this gap, there is a lack of systematic guidelines, processes, and tools supporting the building of architectures based on requirements.

The following sections will present aspects such as software product line engineering,

4 Considerable literature exists in this area. See (Amyot & Mussbacher, 2001), (Amyot, 2003), (Dahiya & Sachdeva, 2006), (Galster, Eberlein, & Moussavi, 2006), (Gruler, Harhurin, & Hartmann, 2007), (Arbab & Meng, 2008), (Arboleda, Casallas, & Royer, 2009)

requirements analysis, architecture and software components and they were considered in supporting of this research.

2.1 Software Product Line Engineering

Software Product Line (SPL) development is an approach to support software reuse that has been successfully applied in many organizations. It helps to identify common and variable parts of a product family through the use of domain knowledge. As a result a configurable platform can be built and become a common baseline for all products belonging to a product family. One approach to develop software products can be done using clone-and-own reuse. It usually can produce savings when compared to a product development from scratch. However, the maintenance is inefficient because this approach adds additional costs to maintain common parts of the products (Eriksson, 2007).

SPL is a growing software engineering sub-discipline because it promises results in cost reduction, gains in productivity, time to market and product quality. SPL is a set of software-intensive systems. These systems share a common, managed set of features satisfying needs of specific market segment or mission. These features are developed and evolved from a common set of successful core assets, attracting customers from a large variety of domains (Ahmed & Capretz, 2008).

SPL shares a common architecture, consisting of a set of reusable assets to produce individual products and a set of products derived from them. Variability in SPL makes products differ significantly from each other. As a consequence there is a need for

methods to represent variability in an efficient way. There are similar definitions about SPL, however they can emphasize different aspects such as, market driven as defined by Clements and Northrop (2001) and technology-driven as defined by Bosch (2000).

The development of a software product line is organized as a two phase activity: development and deployment. Development implements common components present in each product of a product line. Deployment creates derived individual products to satisfy a set of specific market or customer requirements. A configurable SPL is characterized by little or no need to develop product-specific software in the deployment phase. SPL engineering has two key elements differentiating it from other software development approaches, namely variability and its management (Asikainen, Männistö, & Soininen, 2007).

There are methods created to design and manage SPL building process such as Family-oriented Abstraction, Specification and Translation (FAST) (Weiss & Lai, 1999), Product Line Use case modelling for Systems and Software engineering (PLUSS) (Eriksson, Börstler, & Borg, 2005), Method for Requirements Authoring Management (MRAM) (Mannion & Kaindl, 2008). However, they are considered more market-driven or technology-driven approaches.

Customer Oriented Product Evolution (COPE+) (Ullah, 2009) is one of methods using a customer-oriented approach. It uses a voting and ranking results to select product variants to the next release. Feature-based Object Oriented Modeling (FOOM) (Ajila & Kaba,

2008), (Ajila & Tierney, 2002) has a main focus on user-driven features through a product line's architecture. UML is used to describe high level abstraction of whole product families. Unified Software Development Process has a feature model which formalizes the relationship between the user's perceptions of the system, how developers design it, and the implementation of the product.

SPL also promotes extensive reuse in software development because of the many benefits that come with reuse. These benefits include increased quality and productivity, decreased cost and time-to-market, increase customer satisfaction and increased company competitiveness (Ajila & Tierney, 2002).

Domain engineering and application engineering are parts of SPL engineering. Domain engineering aims to create a product line, develops reusable assets and defines commonalities and variability in an explicit way. Application engineering builds products based on the product line and on the reusable assets. A key process considered in application engineering is product derivation. It aims to minimize product-specific development in application engineering and to maximize reuse. It is made through the selection and customization of assets from the product line in order to satisfy customer or market requirements. Although it is considered a key process, product derivation is time-consuming and expensive for software developing companies (Rabiser, O'Leary, & Richardson, 2011).

Commonality and variability management are the dimensions involved in software

product line approach. Commonality management is focusing on the way product features and characteristics are common across products belonging to the same product line (Ahmed & Capretz, 2008).

Variability management is dealing with the way product variable features and characteristic are managed between different products of the same product line (Ahmed & Capretz, 2008), (Asikainen et al., 2007). Variability is an important dimension of software architecture design because it is applied in a particular context of a system and it is a system's property that makes it efficient, extendable, changeable, customizable or configurable. A product with variability makes it applicable in a larger range of purposes such as, multiple user segments, price categorization, support for multiple platforms and operating systems, different sets of features for different needs and so on (Asikainen et al., 2007).

The approach of SPL has a good potential into the contribution process for this research. The result of this research can make some potential contribution to improve the selection process and creation of customizable software platform. The following section describes the requirements analysis stream identifying some of the approaches that can be applied along with the proposed technique.

2.2 Requirements Analysis

Requirements Engineering (RE) deals with elicitation, analysis, specification, validation, and management of requirements (Amyot et al., 2010). The use of functional and non-

functional goals during the RE process facilitates the task of deriving and refining architectures (Lamsweerde, 2001). The fulfilment of functional and non-functional goals is closely related to the software architecture design (Galster et al., 2006).

Goals and scenarios provide a technology neutral approach. They are complementing each other during the requirement analysis stage and support the decision to choose an initial software architecture, create more stable system requirements and refine them to a set of component goals and component scenarios. Goals capture stakeholder intentions and allow verification against system objectives. Scenarios describe the necessary sequence of actions to achieve the stakeholder captured goal (Jarke, Klamma, Pohl & Sikora, 2010). A set of business goals can provide architectural drivers which are requirements that impose an architectural solution. Most of the requirements are related to quality attributes such as exceeding high performance, security, availability, modifiability and so on (Clements & Bass, 2010).

Liu & Yu (2001) argue that different ways to use a system can be represented by scenarios. A scenario can describe system environment interactions and events inside a system. The purpose of scenarios is to elicit or validate system requirements and make real the descriptions of use-oriented systems. They describe the objectives for goals and scenarios as follows, goals have three main objectives: refine functional and non-functional requirements, explore alternatives and how to operate them into architectural constructs. Scenarios are used to represent the incremental elaboration and build requirements into architectural designs. Furthermore, goals are describing: objectives that

the system should achieve including cooperation of agents in the software-to-be and the environment; why data and functions are there; and if they are sufficient or not for achieving the high-level objectives. The authors also argue that putting together goals, scenarios and agents can mitigate some of the deficiencies and limitations in the context of requirements engineering to architectural design process. Although goal-driven and scenario-based approaches are useful, there are complicated and highly dynamic interactions. Both approaches have overlap and gaps.

Liu & Yu (2004) suggest a designing process using goal and scenario modelling approach. They propose a combined use of Goal-oriented Requirement Language (GRL) and scenario-based notation Use Case Maps (UCM) for representing design knowledge of information systems. This process models a social context through dependency relationships among agents and roles.

Clements & Bass (2010) say that business goals justify the creation of a particular system providing the rationale for requirements and architecture design. They are an important category of architectural knowledge when they support development and acquisition of a system.

Goal-Oriented Requirements Engineering (GORE) leads to software and systems requirements through uncovering, analyzing and describing stakeholder goals. GORE has being successfully applied with new types of analysis over nonfunctional properties and is facilitating the documentation of design rationale. GORE has several categories of

approaches such as satisfaction analysis, metrics, planning, simulation and model checking (Amyot et al., 2010). There are several benefits that analysis procedures to goal models can provide:

- Modellers can assess the satisfaction goals
- Evaluation of high-level design is facilitated
- Decision can be made on the high-level requirements and design of the system
- Sanity of a model can be tested
- Communication and learning are supported.

Lamsweerde (2001) states some advantages of GORE as follow:

- Object models and requirements can be derived systematically from goals
- Goals provide the rationale for requirements
- A goal graph provides vertical traceability from high-level strategic concerns to low-level technical details; it allows evolving versions of the system under consideration to be integrated as alternatives into one single framework
- Goal and/or graphs provide the right abstraction level at which decision maker can be involved for important decisions
- The goal refinement structure provides a comprehensible structure for the requirements document
- Alternative goal refinements and agent assignments allow alternative system proposals to be explored
- Goal formalization allows refinements to be proved correct and complete

According to Horkoff & Yu (2011), practitioners have restrictions adopting the goal-oriented approach because of the confusion due the diversity of the available analysis procedures. Goal models have a unique way to capture a system domain and requirements leading to analyze the achievements of objectives. They suggest to use Goal-oriented Requirement Language (GRL) Satisfaction Analysis and/or Metrics techniques for general GORE model.

2.3 Architecture

The following section describes the importance of the Architecture for software platform

development process.

According to Schewick (2004), a system architecture is a structured composition of components that performs the required functionality and the same functionality can be created by different architecture. Technical qualities are defined by the architecture which must support the desired qualities. Architecture also impacts firm and market structures for development, production and use of system components.

Furthermore, architecture design contributes to improve communication and understanding among stakeholders, allows evaluating early design decisions and provides a technology neutral solution with reusable and transferable concepts. Good architecture also addresses the solution for the problem and supports the software development process with good software quality. Characteristics such as high modularity, high cohesion and low coupling are features of a good architecture (Galster et al., 2006).

Software architecture and components from the development phase are adapted to satisfy the requirements during the deployment phase (Asikainen et al., 2007) while requirements engineering and architectural design are processes intended to create system specifications (Pohl & Sikora, 2007).

The approach to develop a software platform is considering the architecture dimension of the software product line that has been a key area of research and some researchers (Ahmed & Capretz, 2008), (Arboleda et al., 2009), (Asikainen et al., 2007), and (Gruler et al., 2007) are focusing on aspects like domain engineering, product line architecture,

commonality and variability management.

The decision to design the software architecture is close to which available software components are going to be selected for an SPL. The next section is providing some insights about the software component stream.

2.4 Software Components

Considering that a software platform is intended to satisfy two or more stakeholders on the business application software, the Commercial Off-The-Shelf software (COTS)[5] concept can be applied for selection and use of components.

Pohl & Ulfat-Bunyadi (2005) consider, for COTS components, that high-level components are more important than low-level components. High-level components provide a significant fraction of functionality of the software product line. During the software product line domain engineering process, the most important activities are evaluation and selection of COTS components. According to George, Fleurquin & Sadou (2008), considering the selection process of a large amount of markets and libraries for COTS components and the diversity of description formats, may affect the advantages of using COTS software development approach, voiding the cost and time reductions.

The high-level COTS selection process is interrelated with domain requirements engineering and domain design in the Software Product Line (SPL) development. As input there are requirements, architecture and the variability model which are considered

5 COTS is a term coined since the 1980's by the software industry as an application designed for software reuse or component. Originally it was a Federal Acquisition Regulation term meaning a non-developmental item.

to identify the best-fitting COTS component. A comparison to differentiate between integrating a COTS component into a single system and integrating it into an SPL as domain artifact is the variability (Pohl & Ulfat-Bunyadi, 2005).

George, Fleurquin & Sadou (2008) consider that the use of COTS components for software applications can reduce development costs and deliver in a shorter time-to-market. The software development life-cycle must be rethought when using COTS components which are viewed as "black-box" units published by third party developers. This software development approach deals with constant trade-offs between requirement specification, architecture specification and COTS selection.

Custom-made software is very expensive, hard to maintain and to inter-operate with other systems. The changing business requirements demand makes it more difficult to keep it updated. To gain a market segment, the component software developer needs to create a set of components which covers variety and quality as is offered. Components buyers have some advantages in acquiring these components such as the quality of the component grows faster than custom-made ones, productivity and innovation from many vendors are combined providing more benefits, better support modularity of requirements, architectures, designs, and implementations (Szyperski, 2002).

Szyperski (2002) also proposes that component software has good balance between custom-made and standard software, because there are different types of components according to their qualities. They can be assembled matching a fixed budget at the level

of the individual priority. The upgrading process is implemented only as it is needed without a massive upgrade cycle.

2.5 Lessons Learned

The literature review provided the necessary information to decide on how this research would contribute for the requirements analysis process to create a configurable software platform in providing a set of steps to develop the research method. The following paragraphs are presenting the lessons learned from each stream as presented in the previous sections.

2.5.1 Software Products Line Engineering

SPL is recommended to be applied in case there is a need for product line solution and it helps to identify common and variable parts of software components.

There are methods (FAST, PLUSS, MRAM) for technology or market-driven approaches to design SPL solutions. However, fewer SPL methods based on user-driven requirements can be found, like COPE+ and FOOM.

2.5.2 Requirements Analysis

A refined software architecture design can be done using functional and non-functional goals during the Requirements Engineering (RE) process. There are recommendations to use goals and scenarios in the RE process.

Goal-oriented and scenario-based models are viable and known approaches to gather requirements from the software platform stakeholders. They are also technology neutral.

Goals capture stakeholder intentions and allow verification against system objectives. Scenarios describe the necessary sequence of actions to achieve the goals from each stakeholder.

A goal-oriented approach seems to be a good fit to accommodate several non-functional requirements from distinct stakeholders and scenario-based models can be applied to specify system functionality. The use of URN can be a choice for requirements analysis activities for a configurable software platform.

2.5.3 Architecture

Software architecture design creates a structured composition of components to perform required functionalities with desired qualities. It also contributes to improving communication and understanding among stakeholders, evaluation in the early design decisions and technology neutral solution.

The combination of common and variable components is a key activity to design a software platform as SPL. However, there is a gap to be reduced between requirements analysis and software implementation stages.

For this research there is an opportunity to reduce the gap by identifying a configurable software platform solution in the early stages of requirements analysis.

2.5.4 Software Components

In the literature review, the software components session is directing to COTS components. There are some advantages and disadvantages in using COTS components. The advantages are higher components quality, productivity and innovation. However, there are disadvantages if the selection process involves large amounts of markets and libraries components affecting the advantages by avoiding the cost and time reductions. Identifying modules is a sensitive issue in the product line and hard to do using the current RA processes with technology or market-driven approaches.

The most important selection process is the high-level COTS selection process. The inputs are requirements, architecture and the variability model which are used to identify the best-fitting COTS component.

During the initial stage of requirements analysis activities, providing the right combination of requirements, architecture and variability model can produce a good design for a configurable software platform.

The next section summarizes the findings from each stream and its key highlights which were considered as supporting points for this research.

2.6 Summary

Stream	Key highlights of the stream	Key references
SPL Engineering	- Selecting components is important - Technology and Market-driven focus mainly - Not much work has been done for configurable software platform supporting user point-of-view	Ahmed & Capretz, 2008 Ajila & Tierney, 2002 Asikaine et al., 2007 Eriksson et. al., 2005 Eriksson, 2007 Mannion & Kaindl, 2008 Rabiser et. al., 2011 Ullah, 2009 Weiss & Lai, 1999
Requirements Analysis	- Goal-oriented for NFR and stakeholder's intentions - Scenarios for functional requirements and sequence of actions - Technology neutral	Amyot et al., 2010 Clements & Bass, 2010 Galster et al., 2006 Jarke et al., 2010 Horkoff & Yu, 2011 Lamsweerde, 2001 Liu & Yu, 2001
Architecture	- Components composition with functionalities and qualities - Improving communication and understanding among stakeholders - Combining common and variable components is key	Ahmed & Capretz, 2008 Arboleda et al., 2009 Asikainen et al., 2007 Galster et al., 2006 Gruler et al., 2007 Pohl & Sikora, 2007 Schewick, 2004
Software components	- Quality of components grows faster - May help with cost and time reductions - Right combination of requirements, architecture and variability is key	George et al., 2008 Pohl & Ulfat-Bunyadi, 2005 Szyperski, 2002

Table 1: Streams from the literature review phase

Table 1 summarizes the streams were identified during the literature review phase. These streams are supporting the research to help the creation of a technique that provides a set of steps to map requirements to a software platform configuration.

Chapter 3 presents the Research Method used for this work.

Chapter 3: Research Method

The research method was defined in five steps to answer the research question. These steps provided the necessary elements to develop and validate the proposed technique to reduce the gap between requirements analysis and software development for a configurable software platform. A constructive research approach was taken for mapping the requirements to a platform configuration.

3.1 Getting the elements to create the process

During this phase, some streams were identified in the literature review which were considered relevant to answer the research question. Furthermore, a selection was made for the requirements engineering process and the tool to be used during the proof of concept.

From the literature review the following proposition was made: Stakeholder goals are fundamental to make distinction from several stakeholders' NFRs and would help to define a preliminary configuration for a software platform during the requirements analysis phase. It is hypothesized that, if *stakeholder goals are used to map requirements to create a configuration of software platform and scenarios are used to identify components variability and commonality at the domain level development phase, then platform configuration and components library set are identified in the early stage of requirements analysis.*

The User Requirements Notation (URN) method was selected because it is a recommended standard (Z.151) from International Telecommunication Union-Telecommunication (ITU-T). It has been evolving since its first issue, in 2003, and is recognized by several segments of the software industry. It is composed of Goal-oriented Requirement Language (GRL), representing stakeholders goals and User Case Map (UCM), which represents work flows and functional requirements. GRL and UCM are shown as a good fit to develop the proof of concept to answer the research question. A complementary tool called jUCMNav[6], is a plug-in for Eclipse Modelling Tools. Eclipse software is developed and supported by Eclipse Foundation.

Once the streams, requirements analysis process and tool were delimited to design the initial process, the next step was creating the initial process.

3.2 Create the initial process

The initial process to perform a set of steps to allow mapping requirements to a platform configuration is created based on goal-oriented and scenario-based approaches.

The steps of this technique were created through iterations and testing of the hypothesis presented in the Section 3.1 . The URN notation is used to support the goal-oriented and scenario-based models.

Chapter 5 shows in more details the technique designed to create the requirements documentation during the elicitation phase.

6 jUCMNav is developed by School of Electrical Engineering and Computer Science research group at University of Ottawa, Ontario, Canada

Having the first version of the proposed technique, the next step was selecting the case study.

3.3 Select Case Study

To perform the experiement, a selection of a very simple case study was made and that could represent the aspects of two or more stakeholders requesting similar features, however each stakeholder would request a small workflow variation to be satisfied.

This particular case study was selected because the author has shown interest in developing a business opportunity to provide service applications for a variety of small and medium businesses, which matches the problem to be solved. The motivation for using this case study is because during the interview process with some potential customers, they had requested variations on how the offered service should be performed in order to add more value to the service.

The case study although simple, is complete enough, it permits implementating the proof of concept in a real world situation. The selected case study is an application to generate tickets that could be drawn from a set of customers participating in a campaign promoted by the company purchasing this service offered by the software platform owner.

During the elicitation phase for the ticket generation system, it was identified that three business owners would like to use the application requesting some specific needs. Chapter 5 provides detailed information and explains the needs of each stakeholder.

The case study showed to be valuable to perform the proof of concept experiment and it was possible to apply the initial process that would solve the gap stated in the research question.

3.4 Validate process (iterate)

For this step, the proposed technique was performed and tested using the selected case study. Each requested requirement from stakeholders was registered using URN notation. The tool jUCMNav helped to model GRL graphs and UCM models through iterations for each stakeholder requirements. During this phase, several alternatives of GRL graphs were created iteratively to obtain the best set of graphs to analyze and identify potential set of parameters to create the configuration table.

Chapter 5 shows a detailed step-by-step activity to validate the proposed technique. A prototype was created as a proof of concept validating the proposition and providing a reasonable answer for the research question.

3.5 Refine process (iterate)

The modelling process of GRL graphs and UCM models for each stakeholders requirements provided some feedback to iteratively improve this proposed technique. Each iteration was evaluated by the validation phase to verify the hypotheses and helped to improve the basic architecture for the software platform and the proposed technique.

3.6 Discuss managerial implications

The research method helped to create the necessary guidelines to design the proposed technique. The selection of case study was simple and complete enough to provide the necessary elements to be tested and validated in the tickets generation application. It was shown as a valid starting point that can be easily extended for more complex systems.

This work showed a novel technique to map and create a configurable software platform from stakeholder requirements. All the aspects related to the management of software platform and reuse of software components are not considered due time and scope restrictions. A more complex software platform needs more attention in the management matter.

The core concept is created and should evolve to encompass perhaps a SPLE approach and add value both for software developers teams, software platform owners and services platform buyers. This requirements mapping technique to create a platform configuration may provide a solution that can be implemented faster, at lower cost and with good quality product.

During the elicitation process for this case study, a basic ticket generation process was presented to all three types of business owners. Feedback was collected from each of them to understand what features are common to all three and what they would like as additional features.

The modelling process of GRL and UCM was incrementally performed and the final

model incorporating all the requests was verified with each stakeholders. Perhaps the same cycle of iterations can be applied for new requests in an incremental mode.

On the other hand, the same process may not be applied for a bigger software platform and may require some adjustments to do elicitation and modelling of GRL and UCM. At this point, scalability is an issue to be investigated.

The next chapter provides the background for the proposed technique developed in this work.

Chapter 4: Background and Overview of the Proposed Process

The basic problem this research is solving relates to how it would be possible to identify a configuration for software platform during the early stages of software requirements engineering. This chapter presents a background and overview of the elements taken to support and develop a configurable software platform to satisfy different stakeholders requirements. In addition, this chapter presents a short description about each key step fitting with the background.

The approach for this proposed technique is based on the user requirements point-of-view, because there are some issues related to the development and maintenance of a software platform on how to satisfy distinct requirements from several stakeholders with a potential reduced cost and time to market. Furthermore, it provides a way to reduce the gap between the requirements analysis and software development stages (implementation) to create a configurable software platform at the domain level.

For the proposed technique, the URN method is selected for two reasons. First, there is a combined notation of Goal-oriented Requirement Language (GRL) graphs and Use Case Maps (UCM) models. The reason for using this approach is because we can document stakeholder goals and requirements through GRL and UCM notations, which makes the output fitting better for the user requirements point-of-view. Second, jUCMNav is a software tool plug-in of Eclipse modelling tool, developed by Amyot's Software Engineering team, at School of Electrical Engineering and Computer Science, University

of Ottawa, located in Ottawa, Ontario, Canada. This tool is URN adherent and under active development updating.

This chapter discuss the following topics: Requirements Analysis, Creation of Scenarios, Configuration and Software Design. The following sections describe in more detail how each topic is the supporting tool of the proposed technique. Chapter 5 provides the five steps showing how each supporting tool is applied.

4.1 Requirements Analysis Phase

The requirements analysis phase is targeting a potential solution to create configurable software based on GRL graphs. The way it is intended to be used is eliciting goals, functional requirements and NFRs from each type of stakeholder involved in this software platform.

No additional elements were needed during this stage and the main focus was to design GRL graphs that would map the set of requirements from each type of stakeholder.

The first version of the GRL graphs made possible to work with creation of scenarios to model the basic workflow that would complement and satisfy each type of stakeholders requirements. The following sub-sections are presenting a summarized description about URN and GRL as supporting modelling tool for this technique.

The designing process suggested by Liu & Yu (2004) can be adapted to these following sub-sections 4.1.1 - 4.1.3 . The proposed technique in this research adds multiple GRL

graphs showing existing differences between stakeholders or group of stakeholders.

4.1.1 User Requirements Notation

User Requirements Notation (URN) is a standard of Z.151 series of Recommendations from the International Telecommunication Union (ITU-T). Although URN was initially intended for telecommunications systems and services it is also being applied to complex reactive, distributed, and dynamic systems. It supports the development, description, and analysis of requirements (Mussbacher et al., 2008), (Amyot, 2003). The following paragraphs describe URN elements based on Amyot (2003).

URN is composed of two complementary notations: GRL and UCM. GRL provides description of business goals, non-functional requirements, alternatives, and rationales. UCM provides functional requirements description based on scenarios approach. It addresses specific needs of visualization and analysis both for functional and non-functional requirements (NFRs). These requirements can be reviewed for correctness and completeness. More detailed information about UCM is provided in the sub-section 4.1.3.

There are two objectives in using URN: the first is related to requirements engineering activities being more rigorous and predictable, and the second one is to provide clearer, more consistent, correct, and complete results. As a consequence, some potential benefits should be obtained from the output results such as, reduction of development costs, earlier delivery of products to market, and increased customer satisfaction. It allows graphic representation of user requirements and system requirements. User requirements

describe goals and functions that stakeholders expect the system to achieve. System requirements are refined descriptions from stakeholders to be incorporated while a system or application is developed.

For this research objectives there are some listed of the URN capabilities as follow:

- User requirements can be acquired with very little design details
- Dynamic refinement capability allocating scenario responsibilities to architectural components
- Detection and avoidance of undesirable interactions between services
- Managing evolving requirements
- Synthesizing feature interactions and performance trade-offs early in the design process at requirements level
- Expressing, analyzing and dealing with goals and NFRs
- Expressing relationship between goals and system requirements
- Capturing reusable analysis and design knowledge related to know-how for addressing NFRs

A complete list of capabilities is provided in Amyot (2003).

It is important to be able to analyze the processes and goal satisfaction when considering the use of URN because it supports formal modelling and analysis of user requirements. In addition, it is suitable for describing most types of reactive and distributed systems. The two complementary notations to URN, GRL and UCM have specific roles in the requirements engineering process. GRL was created to acquire business and system goals, and the alternative means and principles used to achieve goals. UCM pictures behavioural scenarios through causal flows of scenario activities. Scenario activities represent something to be performed and are called UCM responsibilities. They can be

allocated to components which may represent software or non-software entities (Weiss & Amyot, 2005).

The following sub-section is providing a summarized description of the GRL notation.

4.1.2 GRL

The goal-oriented requirements engineering facilitates the requirements analysis during the early stages of the software development cycle incorporating nonfunctional requirements (NFRs) and functional requirements along with business objectives (Liu & Yu, 2004), and the evaluation of alternatives. GRL has the core concepts of *i** and the NFR Framework and links can be created to UCM which is a type of scenario notation. A strategy containing high-level requirements provides the initial satisfaction level for intentional elements such as goals and tasks. The use of propagation algorithms propagates the satisfaction level through links that connect other intentional elements of the model. Then the effectiveness of the high level requirements contained within the strategy can be examined. URN can be applied both for proposed and evolving system to analyze discovered and specified requirements for correctness and completeness. In the URN, there are concepts for the specification of stakeholders, goals, NFRs, rationales, behaviour, scenarios, scenario participants, and high-level architectural structure. URN combines GRL with the UCM notation. GRL incorporates goal-oriented model and intentional concepts which relate to NFRs, quality attributes, and reasoning about alternatives. UCM notation is incorporating modelling scenario concepts which relate to

operational requirements, functional requirements, and performance and architectural reasoning (Amyot et al., 2010).

Amyot et al. (2010) state the following as major benefits of GRL over other notations:

> *"The GRL benefits are the integration of GRL with a scenario notation, the support for qualitative and quantitative attributes, and a clear separation of GRL model elements from their graphical representation, enabling a scalable and consistent representation of multiple views/diagrams of the same goal model."*

Liu & Yu (2001) describe Goal-oriented Requirement Language (GRL) as a language that supports goal and agent oriented modelling and deals with both functional and non-functional requirements. Completeness of requirements can be evaluated by how sufficient they are to establish the goal. GRL allows distributed goal models. The following paragraphs describe some of the elements according to Liu & Yu (2001).

There are three main categories of concepts: intentional links, elements and actors.

Intentional links have five types:

1) means-ends
2) decomposition
3) contribution
4) correlation
5) dependency

Intentional elements are goal, task, soft-goal and resource. They permit understanding:

- Why particular behaviours, informational and structural aspects were chosen
- What alternatives were considered
- What criteria and reasons were used to choose one alternative over the other

Goal is a state of affairs or condition the stakeholders would like to achieve, generally describes functional requirements and NFRs. Goal is represented by a rounded

rectangle. Alternatives on how to achieve the goal can be considered. Types of goals are business and system.

Business goal - goals that individual or organization wishes to achieve

System goal - goals that a system should achieve

Task, represented by a hexagon, specifies a particular way of doing something

Soft goal is a state of affairs or condition that the actor would like to achieve, It is represented by a irregular curvilinear shape. It is a solution that satisfies soft goals or achieves goals and provides: operations, processes, data representation, structuring, constraints and agents. Some issues related to soft goal have no clear-cut criteria and it is up to subjective judgement and interpretation. It is used to represent NFRs and it should be addressed as early as possible in a software life-cycle.

Resource, represented by a rectangle, is a physical or informational entity.

Actor represents a stakeholder and is a holder of intentions. It is an active entity interested in goals to be achieved, tasks to be performed, resources to be available and soft goals to be satisfied (Amyot et al., 2010).

Amyot et al. (2010) describe the uses of quantitative, qualitative and hybrid algorithms that may vary on the details of satisfaction levels. While the goal model matures and more details are gathered through the development process, an evolving approach can be taken from qualitative to hybrid to quantitative algorithms. Some advantages of using

GRL are:

- It is defined as a precise language that is open enough to support numerous types of rigorous analysis
- It is integrated with the UCM notation exploring the synergy between goals and scenarios in two ways
- Key performance indicators can be defined and are useful metrics to improve process

GRL models provide a description of business and system goals, alternatives, and rationales that are refined into requirements. GRL models are useful for visualizing static relations existing between goals, the alternatives means to achieve these goals, their interactions, and accompanying rationales (Amyot 2003). An evaluation mechanism is available to measure the impact of qualitative and/or quantitative decisions on the level of satisfaction of high-level goals. During a system development process, a requirement which defines functions of a system is known as functional requirement (Amyot et al., 2010).

4.1.3 Creation of Scenarios Phase - UCM

The main idea at this phase is to identify potential software components and which part is belonging to variability or commonality sets. The commonality components are applied to satisfy similar functional requirements from two or more stakeholders. The variability components are applied to differentiate and satisfy the behaviour of distinct functional requirements requested by each stakeholder.

An initial set of diagrams are created to represent the workflow that satisfies each functional requirement. The analysis made after this stage is to identify which are

variability components or commonality components. This representation gives a high level of how the system is behaving to satisfy each stakeholder requirements.

As part of solution for this research, the *dynamic stub* is used to design the workflow and represent a commonality complement. The advantage of *dynamic stub* is because an explicit representation of component reuse can be made.

Furthermore, with the combination of the results from UCM diagrams and GRL graphs it would be possible to create an initial configuration for the software platform to satisfy all these distinct requirements. Then the application would work distinctly for each one of stakeholder involved in this software platform, based on the configuration for the platform.

A preliminary version of the software architecture can be created and a prototype can be developed with some available software components which can satisfy all or most of functional and NFRs requirements.

Scenarios have been used for human-computer interface and in software engineering. The successful use of scenarios is mainly due to their capability of stimulating thinking. Scenario can describe situated task vision and is an effective method of communication among the actors in a specific context (Leite et al., 2000).

In the early of 90's, UCM was developed by R. J. A. Buhr and his research team at the Systems and Computer Engineering Department, Carleton University, Ottawa. A detailed description, examples and case studies can be found in Buhr and Casselman (1995).

The UCM model focuses on causal relationships between responsibilities of one or more use cases and provides a view of behaviour and structure along with the flexible allocation of responsibilities to architectural structures. This combination is named grey-box and contributes to bridging the gap between requirements and design. The behavioural framework provided by UCM allows evaluation and making architectural decisions at a high level of design. It can be applied as use case capturing and elicitation, use case validation, high-level architectural design and test case generation. UCM users can make dynamic refinement for variations of scenarios and structures, and incremental development and integration of complex scenarios. Existing additional capabilities in UCM provide more benefits to express and analyze requirements for complex systems (Amyot, 2003b).

A key component for this phase is to identify where are the *dynamic stubs* because they represent a reusable software component during the requirements analysis process.

4.2 Configuration Phase

In this phase an analysis is made to identify the commonality and variability for the software platform. This analysis is considering both NFR and functional requirements to identify how variability of components is going to be combined with commonality of components.

The distinct part here is how we do the interpretation of GRL graphs compared to the approach from URN method. When a Software Developer is following the URN method,

the main objective is to run the simulation analysis for each of goal strategies and identify which strategy fits better for the desired level of customer satisfaction for the software application. Then the best fitting strategy is selected and implemented into the software development process. As a result, an overall level of satisfaction for all involved stakeholders is maintained when the software is developed and implemented.

In this proposed technique, the Software Developer follows the same steps to draw and specify the goals strategies for GRL graphs. However, the main objective here is to make sure that each of requested requirement is documented and has being satisfied for each one of involved stakeholders for the software platform.

In doing that, we are able to identify commonality and variability analyzing each GRL graph. In a complementary way, UCM models provide the workflow and functional requirements requested by all involved stakeholders. The common components are represented by using *dynamic stubs* into UCM models.

Once we have established which components are belonging to a specific group of commonality and variability, we are able to design an initial configuration for the software platform and identify the software components that can be reused from a components library or a COTS components. Considering that the GRL graphs and UCM models are approved from all involved stakeholders into the software platform, we are moving to the software design phase.

4.3 Software Design Phase

The results from requirements analysis and creation of scenarios stages allow the creation of a preliminary version of the software architecture where the two types of components are classified according to variability or commonality sets.

At this point, the development team is able to identify some existing reusable components according to their functionality and how they may satisfy the stakeholders' requirements. The advantage in using these existing reusable components is due the fact that is possible to make them run with very few changes in the interface, perform a test to see how they behave and compare the results with the current requirements documentation.

A selection process to sort the best existing reusable components is made and an initial prototype would be created from this selection of reusable software components. The prototype is created adding some glue code to the prototype's interface and a refined configuration is developed to run this preliminary version.

Once the preliminary prototype is developed and running it can be evaluated and a validation of elicited requirements can be made. In this stage, each stakeholder can verify whether each requirement is satisfied or not. After the evaluation and validation stages are finished, the next step is refining and adding missing elements to the prototype, which is done by improving it in short cycles along with stakeholders participation.

4.4 Outcomes

The outputs from each phase are smoothly integrated as inputs to the next phase. The benefits of using this technique are: creating a configuration at the domain level; satisfying multiple distinct requirements, for a service or specific application, requested by different types of stakeholders; using customer requirements approach instead of market or product-driven approach; identifying configuration for the software platform during the early stage of requirement analysis; and providing low maintenance cost due configurable software platform design.

During the maintenance phase, it is possible to analyze the impact of a new requirement over the existing software platform solution. It is also possible to identify which software components are affected and the Software Developer would be able to propose a solution that can be incorporated into the software platform configuration along with the new or modified component.

The next chapter provides detailed steps for the proposed technique applied to the case study.

Chapter 5: Detailed Process

This chapter describes the steps to be followed by using the technique for mapping requirements to the software platform configuration. This approach is supported by goal and scenario models focusing on user requirements. Some of the benefits from this approach are:

- Design is using a high level approach to create a software platform configuration at the domain level;
- Abstraction is focusing at the domain level not at the tools (programming language or implementation);
- Configuration is a selection of components of multiple sources – potentially easier to use from one tool to another;
- Reuse is across multiple applications, it is not a clone-and-own reuse (inherited from software product line engineering).

5.1 Modelling the Domain Requirements

This step is designed to provide a set of GRL graphs and UCM models from user requirements information collected during the elicitation phase. This step is related to the section 4.1 and sub-sections 4.1.2 and 4.1.3 as part of Requirements Analysis Phase.

Input: Interviews, business process descriptions, user goals and expectations, and domain expert comments.

The input elements in this step are gathered from interviews with stakeholders, documentation describing the business process including goals and expectations from all stakeholders involved in the system and further explanations from domain experts to help

in creating GRL strategies and UCM paths.

Output: GRL graphs and preliminary version of UCM models.

The use of the tool jUCMNav provides the graphs and models without any additional notation or extension.

Actors: Business owner (software platform customer), domain expert, software platform owner, and software developer.

For each potential stakeholder the information gathered during the interviews and enquiring sessions are used as input and a requirements model is drawn at the domain level to organize NFRs and functional requirements that would satisfy each customer or subscriber (stakeholder).

This step is an iterative activity balancing all user requirements for one or more provided services by the software platform and to every involved stakeholder. It is created a GRL graph for each business owner or a group of business owner who share the same functionality. For the configuration table a specific key identification has to be created. It is possible to add key performance indicators to do some measurements that may help to improve some business processes.

The UCM models are created by grouping as much as possible all the functionalities grouped by commonalities and variabilities. Links between GRL graphs and UCM models can be created making possible to perform a simulation of strategies and

workflows.

5.2 Identify Commonality and Variability in the Requirements Model

Requirements analysis is applied in this step looking for potential software components and identifying commonalities and variabilities which represent the features of the system. This step is also related to the section 4.1 as part of Requirements Analysis Phase.

Input: GRL graphs and preliminary version UCM models

The input elements in this step come from the output from the prior step.

Output: Highlighted GRL graph elements and combined version of UCM models with commonality and variability.

To create the output in this step, the software developer analyzes each GRL graph and highlights what differentiates from one GRL graph to another.

Actors: Software developer, software platform owner, and domain expert.

Each GRL graph is analyzed to identify commonalities and highlight variabilities. Each GRL graph is representing a business owner or a group of them. The commonalities are all those elements repeated on every GRL graph and variability are those distinct elements that can be highlighted as unique for the system.

In the sequence, each UCM model is analyzed to combine what are common components

in one UCM model to represent components reuse and other UCM models representing variations to be handled by the software platform. Variability is represented by using *dynamic stubs* where it would be possible to model each different workflow.

At this stage, the Software Designer is able to identify a preliminary set of parameters to be used as part of the data structure for the configurable software platform. The data structure has to be created considering each stakeholder individually. The way it is created makes more flexible in case a specific stakeholder decide to change the way a workflow should be performed.

5.3 Modelling Application Requirements

At this stage a model application is created using all the necessary elements to create the configurable software platform. This step is covered section 4.2 as part of Configuration Phase.

Input: GRL graphs, UCM models, Components Library Set for the software platform.

In this step, all GRL graphs and UCM models are those generated by the tool jUCMNav. The existing software components, both from the development team and third party developers are potential software components to reuse in the software platform. Additional software components can be searched to match the required functionalities.

Output: Application requirements are modelled and verified.

In this step, all GRL graphs and UCM models can be verified using the execution mode

from the tool jUCMNav. Potential customers and domain experts can provide some feedback to the software developer. The objective is to verify and improve all elements created for the software platform.

Actors: software developer, software platform owner, potential customers and domain expert.

This step creates a model incorporating all the stakeholder requirements to be satisfied and all functionalities expected for the configurable software platform.

Using the jUCMNav tool it is possible to run all the strategies for the GRL model and verify the workflow from each UCM model which is linked with its respective GRL graph. This stage shows a high-level version of software platform matching all the application requirements. Additional refinements can be performed iteratively in the step 5.1 until a satisfactory application requirements model is obtained.

5.4 Identifying Existing Components

The existing components inventory or components library set, which can hold both internal or external source of software components, are sorted and selected according to their features matched to the system features. The selected ones are going to be potential candidates to be implemented in a configurable software platform. This step is covering the section 4.3 as part of Software Design Phase.

Input: UCM models with common software components and components inventory.

In this step, a verified UCM models can be matched with the components inventory.

Output: Software components are selected for the software platform.

A selected set of software components from the components inventory are prepared to the next step.

Actors: software developer, software platform owner, and domain expert.

Each UCM model will provide all the information needed to identify a set of software components from the inventory. Evaluating each software component to identify their input and output data set and their features as well is part of this step.

From the component input data set one is able to find a complementary set of parameters to be included in the data structure of the configuration table. Additional data structure is created to accommodate all necessary fields to be loaded when the software platform is performing a request from a specific stakeholder.

A set of existing software components from the local components library set or 3rd party components library set or components inventory are evaluated to identify the components matching.

5.5 Binding the Variabilities to the Components

This step allows one to develop and implement the necessary source code to run the application for the configurable software platform. This step is covering the section 4.4 as

part of Outcomes.

Input: selected set of software components, application requirements model represented by executable GRL graphs and UCM models.

In this step the software developer has all need information to prepare a prototype using the selected set of software components.

Output: Source of glue code running with the selected set of software components.

A running prototype can be tested and verified along with potential customers.

Actors: software developer, business owner, and domain expert.

The glue code is created and access the configuration table to adjust a specific request recorded into the configuration. At this stage a version of configurable software platform can be executed and verified if all the requirements documented are satisfied accordingly to the GRL graphs and UCM models. The glue code should perform in a way that any request from existing stakeholder is performed accordingly.

5.6 Summary

After its implementation, more stakeholders can be added to the software platform. Configuration table and glue code are handling the application's inputs and outputs for each customer or subscriber that makes software components commonality more and more reusable while new customers or stakeholders are added to the software platform.

All those listed benefits in the beginning of this chapter can be noticed from development to maintenance phases of a configurable software platform. As a result, there is a potential to achieve the reduced costs, reduce time to the market and quality software by using this technique.

Chapter 6: Validation

This chapter presents how the proposed technique was applied for the validation phase by developing the experiment from a selected case study.

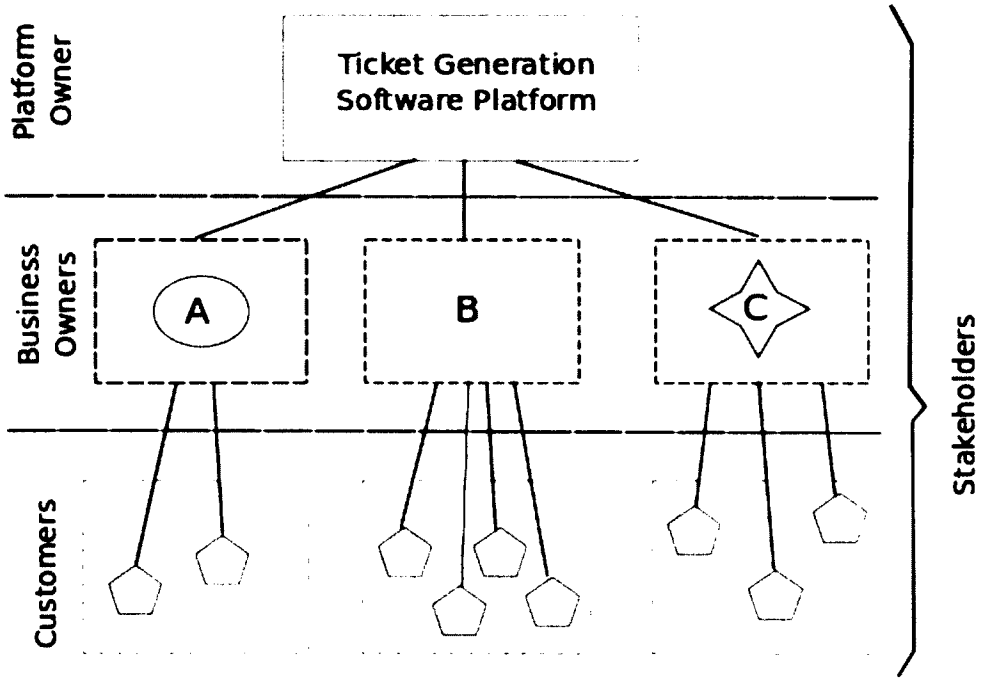


Illustration 3: Elements for the case study

For the purpose of validating the proposed technique, there are some elements in this case study to be defined such as customers, business owners, platform owner and stakeholders: customers are people accessing services or specific applications from the software platform selected by a specific business owner; business owners are people buying services or specific applications provided by the software platform; platform owner is responsible to offer matching services to business owners; and stakeholders are any entity involved in the software platform requesting requirements for a service or

specific application. For this research, there are types of stakeholders sharing features of a service or specific application which matches their unique business. These relationships are showed in Illustration 3.

6.1 Context

The case study selected is an application to generate tickets for customers. This case study, is a need for a business opportunity that has been developed by the author in creating a configurable software platform to offer a set of services for small and medium businesses. A small part of the configurable software platform was selected to run a proof-of-concept.

There are three types of stakeholders interested in using the application as part of the software platform. The first stakeholder type is interested in giving away tickets to their customers and attracting them to participate in a prize drawing. They can redeem the winning ticket at the next purchase. The second stakeholder type wants to collect emails from its business potential customers. Potential new customers and existing customers provide their email and get a ticket that is delivered through email. The stakeholder then draws winning tickets, randomly. That can also be redeemed at the next purchase. The last type of stakeholder, is interested in providing tickets for incentives such as discounts and prizes to returning registered customers, to maintain or increase the loyalty ratio.

The solution shall be implemented in the same software platform and satisfy these three distinct requirements based on which type of stakeholder's customers are coming from.

As described in Chapter 5, the URN was selected to apply the steps to create the configurable software platform during the software development process. GRL is used for the requirements analysis phase and UCM is used for the creation of scenarios. To support each stage of the development process, the jUCMNav tool was used. jUCMNav is a plug-in for the Eclipse software development platform. The following sections provide details about each stage of the software development process to implement this particular solution that must satisfy the stakeholders' requirements for their unique business.

6.2 Case Study Development

This case study software platform is designed to offer services for business owner to attract more sales offering prizes and incentives to their customers. During the survey process, the focus was to find out how business owners would be interested in using this type of service, There are three business segments, business owner A is from the food segment, business owner B is from the construction segment and business C is from the car repair segment. All three business owners were promoting their business via advertisements. For the new approach, using the software platform, each one has distinct strategies which are called business owner strategies A, B and C as illustrated in the Illustration 3.

The first type of business owner strategy (A) is interested in using the ticket generator system to promote and attract more customers to his business. His strategy is to ask the

customer after making a purchase to go at a kiosk and press a button to get a coupon printed where he or she would be participating in a draw for a prize.

The kiosk is a computer connected through the internet and able to print a randomly generated ticket from the ticket generator software platform. The ticket number is recorded into the business owner database to be drawn after the promotion period is ended. The customer has to keep his or her copy of the ticket. After the promotion period is ended, a ticket is drawn and the winning ticket is announced by the store manager on a board. The winner can come and redeem his/her ticket for the prize.

The second business owner strategy (B) is interested in using a similar process, however he is more interested in getting emails from potential customers and returning customers. All buyers are invited to provide their email address to a kiosk. The kiosk is connected to the ticket generator software platform and gets the email address. Then the customer can press submit and a ticket number is generated and sent to the email address. The business owner database keeps the ticket number and the email address till the end of the promotion. After the end of promotion a ticket number is drawn and the winner ticket is announced to the email address. The winner can return to the store and redeem his/her ticket for the prize.

The last type of business owner strategy (C) is interested in promoting more loyalty among his customers. All new customers are asked to register their name, telephone and email with a user name and a password. All registered customers are eligible to provide

the sales receipt number through their registered user name. Customers are able to connect and do the registration (new customers) or login (returning ones), and provide their sales receipt through a link from the store's web page. This link connect them with the generator ticket software platform. Once they submit their sales receipt number, the system generates an incentive ticket to be redeemed at the shop when the customer returns to make more purchases.

6.3 Modelling Process

This section is showing the process used to model GRL and UCM and how a Software Developer is able to design a configurable software platform during the RA process. The next sub-section shows GRL graph modelling.

6.3.1 Modelling the Domain Requirements - Creating GRL graphs

A first version of GRL graph and UCM models was modelled for business owner strategy A identifying the business owner needs to satisfy his requirements. In this phase, there is only a plain modelling design for the business owner strategy A. This section follows the activity suggested in the Section 5.1 as part of the Modelling the Domain Requirements. All the graphs and models generated here from the interview sessions made with business owners and domain experts. Each GRL graphs represents a set of requirements from the business owner.

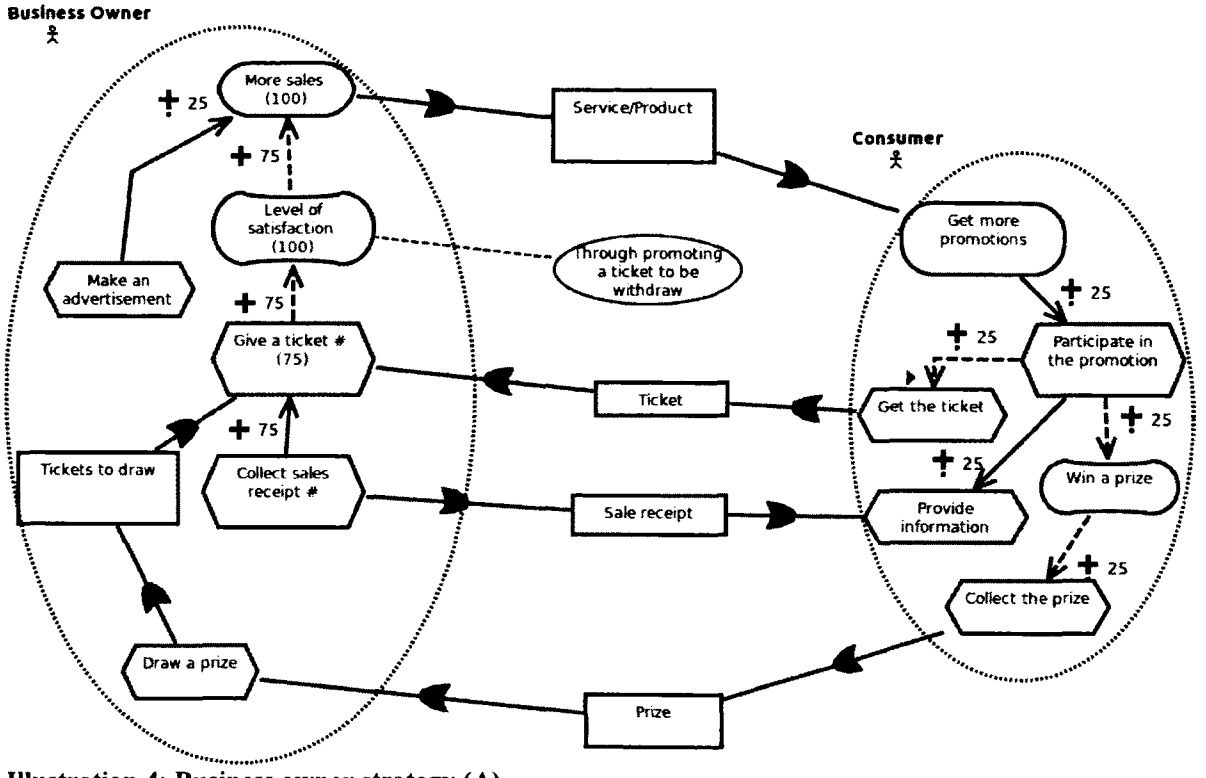


Illustration 4: Business owner strategy (A)

Illustration 4 provides a GRL graph which represents the gathered goals, actors, resources and activities with their respective links. This illustration shows both the old strategy "Make an advertisement" and the new strategy performed by the ticket generation software platform through promoting campaigns to give tickets to be drawn. For this business owner both are strategies to drive more sales, however the giving tickets campaign might be more effective to bring more customers if compared with advertisement strategy.

The next GRL graph is designed to model the business owner strategy B where Illustration 5 shows a new objective requested by the business owner which is highlighted in light green fill.

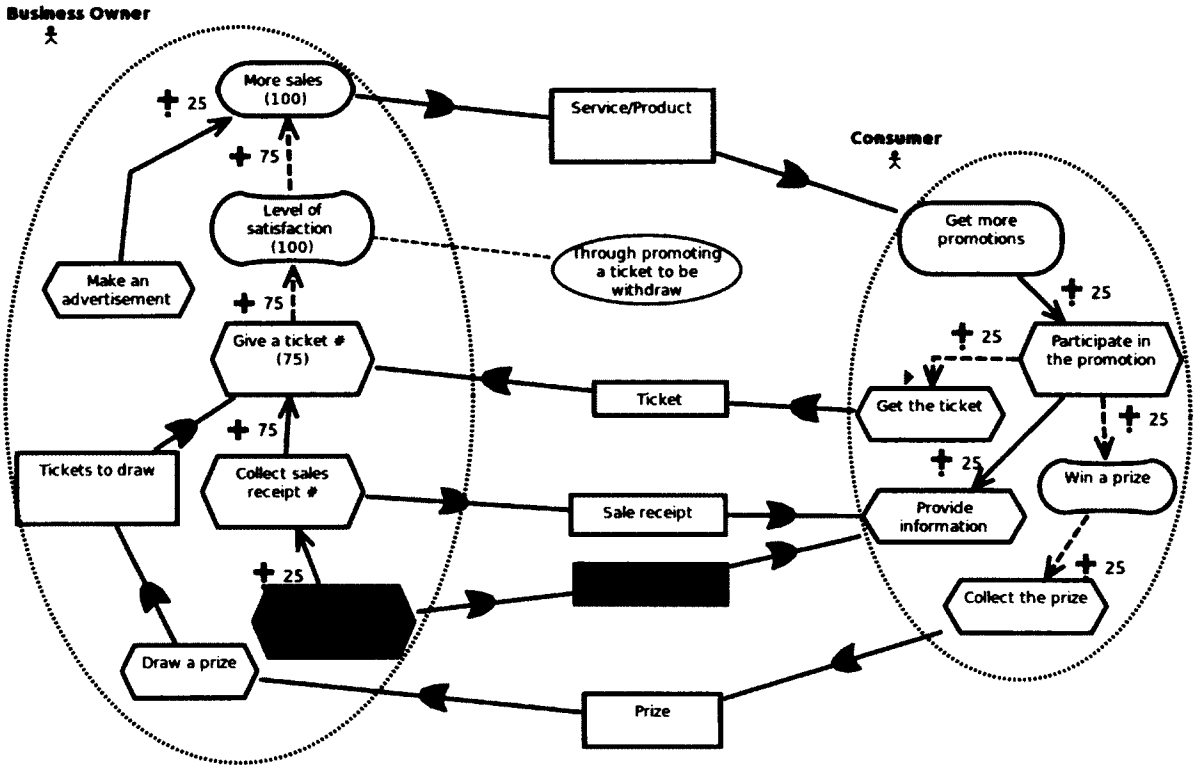


Illustration 5: Business owner strategy (B)

Mostly all the requirements shown in the Illustration 5 are similar or the same comparing from the business owner strategy (A). We can clearly see some commonalities and the variability in this stage which is considered quite early from the RA process. A visual comparison is made between these GRL graphs and the difference is highlighted to show the additional features.

The Illustration 6 shows the GRL graph modelling the goals from business owner strategy (C). It has more distinct elements when compared to the two previous GRL graphs. However, we are able to see which part is differentiating (light yellow) and which one is common.

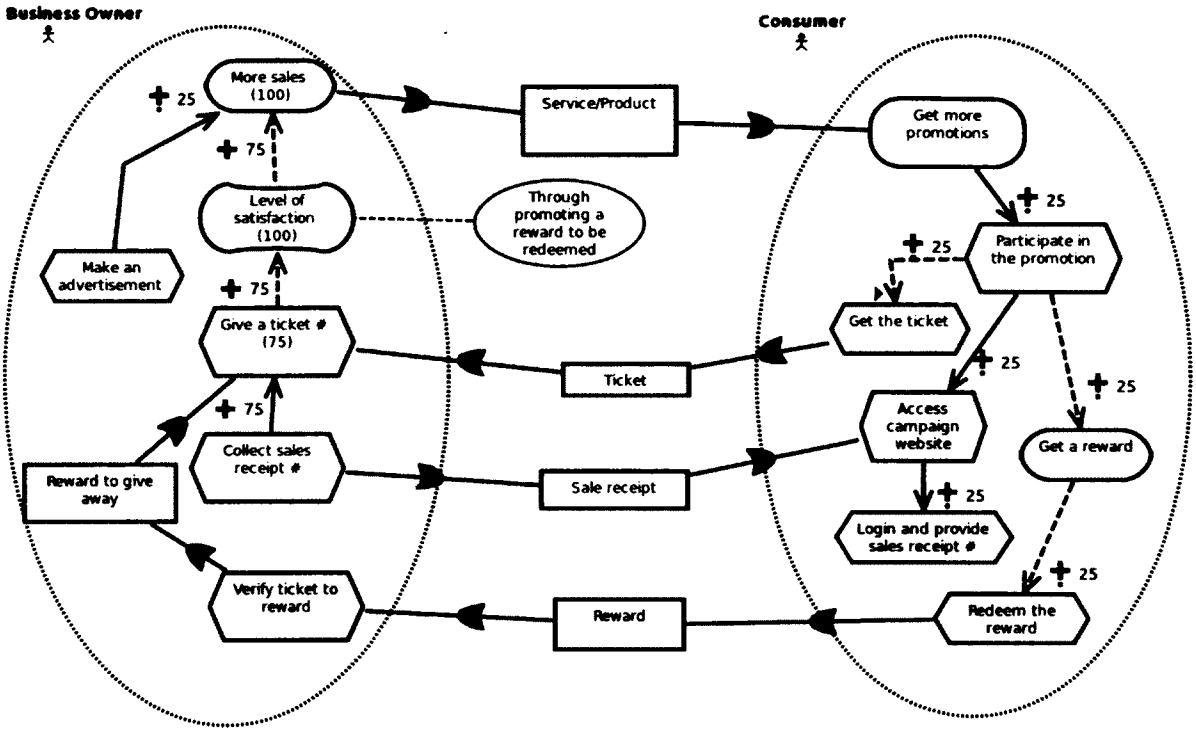


Illustration 6: Business owner strategy (C)

Illustrations 4-6 show possible GRL graphs to model a simple goal "More sales" from three different points-of-view. Every GRL graph is created to provide a specific business owner strategy expecting to increase the satisfaction of hard goal "More sales".

These GRL graphs are the starting point to design UCM models. There are common software components that would be shared with all business owner strategies and variability to make them distinct to run from each other. In the proposed technique, we are able to create UCM models identifying the most common functionalities and prepare to select possible software components available into the Developer's library or into the market. The next sub-section shows UCM models designed both for commonality and variability components for the software platform.

6.3.2 Modelling the Domain Requirements - UCM Models

The activities performed in this step are the continuation of step shown in the Section 5.1 as part of the Modelling the Domain Requirements. Using the information from Illustrations 4-6 we were able to create a series of scenarios to show the workflow and functionalities classified by commonality and variability. The notation for *dynamic stub* is used to represent variability in this technique.

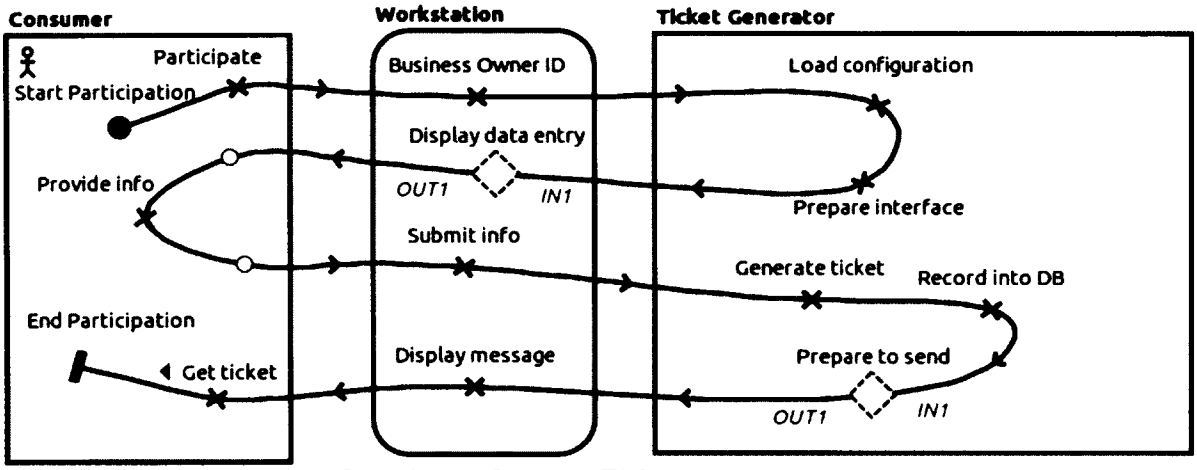


Illustration 7: UCM System Overview to Generate Ticket

Illustration 7 shows an overview on how the software platform can perform part of the requirements stated with GRL graphs. This scenario covers the commonality of the Ticket Generator system. There are two dynamic stubs to represent the variability of this particular system.

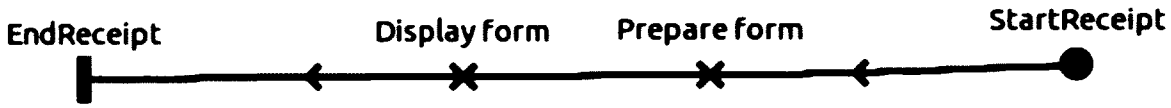


Illustration 8: Dynamic stub for Display Data Entry

Illustration 8 shows the possible solution represented by this UCM model for displaying data entry form. This stub has to manage all three strategic situations from Business

Owners A, B and C.

Illustration 9 shows a dynamic stub with a variation on how the system works for the customer from the business owner C, running a login window before providing the form to collect the receipt number as shown in the Illustration 8.

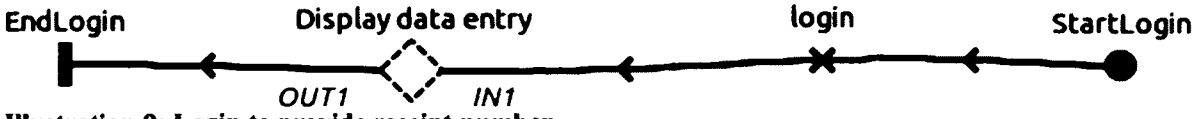


Illustration 9: Login to provide receipt number

The other dynamic stub *Prepare to Send* has two UCM models as shown in Illustration 10 and 11.

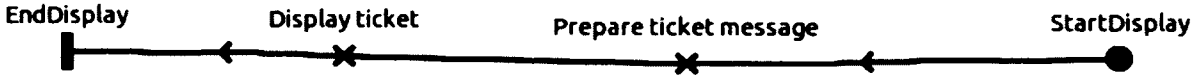


Illustration 10: UCM displaying ticket

Illustration 10 shows the UCM model that represents a way to display the generated ticket to its customer. This particular example shows that it is possible to document a functionality that can be reused.

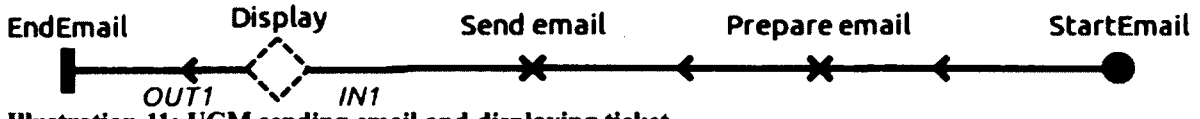


Illustration 11: UCM sending email and displaying ticket

Finally Illustration 11 shows another dynamic stub that prepares an email and sends to the customer. This UCM model can be reused for both business strategy B and C. It also shows a reuse of stub represented in the Illustration 10.

These four UCM models are representing a part of the system platform and they are good enough to show how the proposed technique can create a first version of a configurable

software platform.

6.3.3 Identifying Commonality and Variability in the Requirement Model

The previous sections 6.3.1 and 6.3.2 show the GRL graphs and UCM models respectively with identified variability through the highlighted elements in the GRL graphs. The commonality is identified using *dynamic stub* (dashed diamond) in the UCM models. This step is related to the Section 5.2 as part of the Identify Commonality and Variability in the Requirements Model.

This step creates a preliminary data structure to load the configuration table. This data structure should represent all the data needed to load the request triggered to run the glue code. A partial version of the data structured shown in the Table 2 is created from these GRL graphs and UCM models.

6.3.4 Modelling Application Requirements

This step provides the outputs as described in the Section 5.3 as part of the Modelling Application Requirements. This is made using an existing feature in the jUCMNav tool to perform some executions on the created requirements model (GRL and UCM). This allow the developer to verify if the software platform is performing accordingly to the modelled requirements during the previous steps.

The tool jUCMNav helps to identify and verify if the additional features are satisfying the requirements from each stakeholder. The software developer can provide a simulation of

how the system is behaving for each business owner strategy. Changes can be made in this stage quickly and at low cost.

6.3.5 Identifying Existing Components

This step refers to the Section 5.4 as part of the Identifying Existing Components. The software developer can start selecting the software components to be used to run the software platform. The selected components have to match the functionalities requested by the stakeholders.

Once having the software platform modelled and verified at the domain level and its features and behaviours for each customer or stakeholder represented into the model, an external search was made to identify potential software component candidates to match the features for this case study.

A preliminary configuration table can be drafted to identify the desired behaviour for each stakeholder when the glue code runs. Considering the small complexity of this case study, Table 2 shows the final version of the configuration table.

6.3.6 Binding the Variabilities to the Components

This sub-section shows how a preliminary version of a configurable software platform can be designed. This sub-section corresponds to the Section described in 5.5 as part of the Binding the Variabilities to the Components. The glue code is created and loads the configuration for each registered business owner strategy.

Illustrations 7 to 11 show a set of UCM model that may satisfy a part of the system responsible for ticket generation activities to business owner strategies A, B and C. They are representing both commonality and variability to the selected part of modelled software platform. Appendice A shows a report generated by jUCMNav tool.

Using the UCM model as the starting point to create the software platform, the Software Developer is able evaluate and select from the components library which existing software components would be reused. To design and implement the software platform there are several programming languages available in the market. PHP is selected among many programming languages for this case study where it is considered a web interface solution for the software platform, although any other programming language for web based applications could be used.

In this experiment, two software components were selected for reuse purpose: *generate ticket* and *send email*. There are some available PHP components in the market both open source and proprietary. The criteria to select a software component are components to be added in to the software platform library, updated in a regular basis no longer than one year and used by a large number of software applications

Having UCM models and some potential software components, the Software Developer is able to create a configuration table to satisfy all documented business owner strategies in this case study. Illustration 7 shows a responsibility called *Load configuration* which is inside *Ticket Generator* boundary and is one of the elements responsible for the

behaviour of the software platform according to each business owner strategy.

Field	Type	Null
strategy_ID	varchar(10)	No
members	tinyint(1)	No
send_email	tinyint(1)	No
cap_alphabets	tinyint(1)	No
small_alphabets	tinyint(1)	No
special_chars	tinyint(1)	No
user_chars	tinyint(1)	No
length	int(3)	No
expiring_date	date	No
ticket_title	varchar(50)	No
foot_msg1	varchar(50)	No
foot_msg2	varchar(50)	No
draw	tinyint(1)	No
barcode	tinyint(1)	No
barcode_type	varchar(15)	No
numbers	tinyint(1)	No

Table 2: Configurable Software Platform – data structure

Table 2 shows an example of the structure of configuration table created using MySQL database management. These fields are parameters and were selected to make the software platform configurable when the glue code loads the parameter's values for a specific stakeholder or customer. The following fieds: strategy_ID, members and send_email are parameter values identified during the requirements analysis and the remaining ones from cap_alphabets to numbers came from the selected software components (barcode and ticket generator).

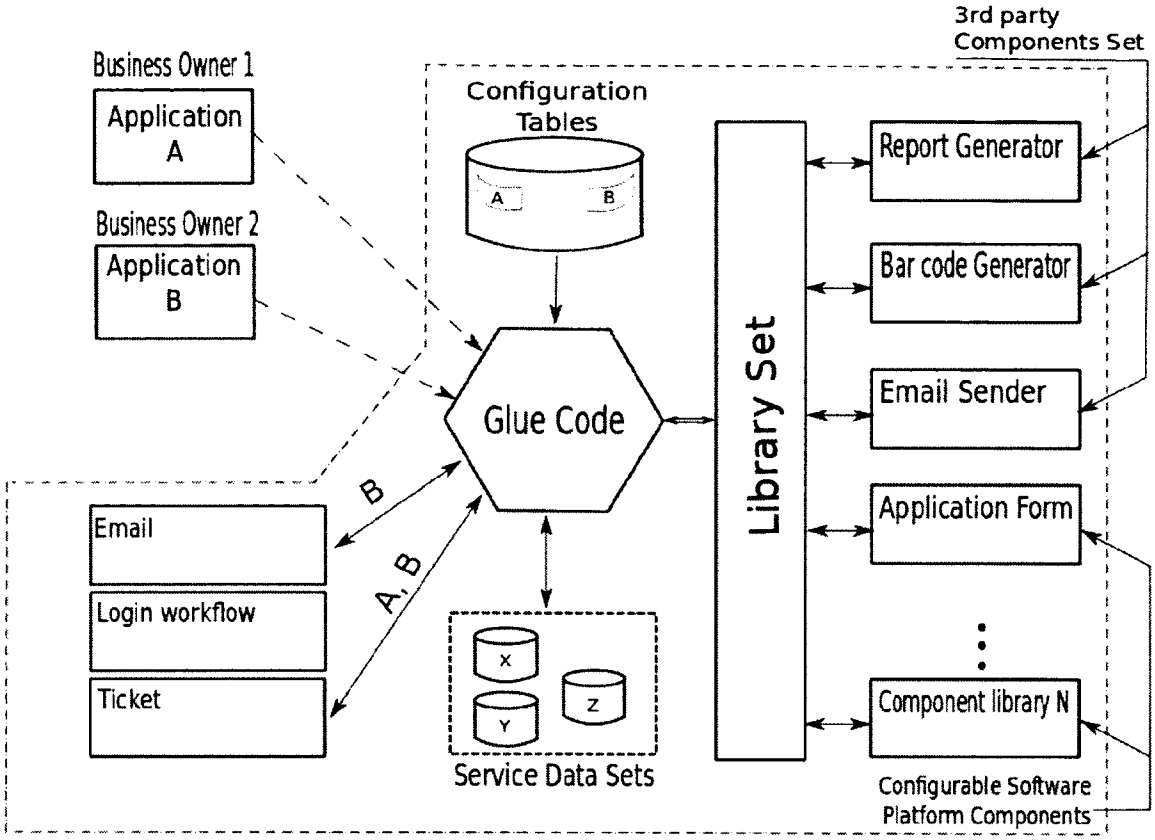


Illustration 12: Configurable Software Platform Architecture (partial)

Illustration 12 presents the partial architecture for the proposed technique where we can see the role of Configuration Table. This illustration is an overview for a generic configurable software platform. For this particular case study the Configuration Table (Table 2) has to match the requirements of each Business Owner strategy. It was created based on information from GRL graphs and selected software components source code.

rp_id	strategy _ID	cap _alphabets	small _alphabets	numbers	special _chars	user _chars	length	expiring_date	send_ email	ticket_title	foot_msg1	foot_msg2	draw	barcode	barcode _type
1	A001	1	1	1	0	0	20	2012-06-29	0	Prize drawing ticket	Please print it and check your ticket after		1	1	code128
2	B001	0	0	1	0	0	12	2012-06-18	1	Prize drawing ticket	Please print it and check your ticket after		1	1	code93
3	C001	1	0	1	0	0	8	2012-06-30	1	Thank you accept this reward!	Please print it and redeem it next time	- at www.acme.ca	1	1	code128

Table 3: Example of Configuration Table Parameters for Strategies (A), (B) and (C)

The glue code loads the configuration for each strategy based on *strategy_ID* field and adapts the performance to each requested service by the Business Owner that has been offered by the software platform.

Table 3 is an example of configuration table with values for each Business Onwer strategy. The glue code accesses this database table and loads the line for a specific strategy. The same strategy can be performed using some different type, size, type of characters and length of barcode. Title and foot messages can be customized accordingly. Each tuple in the Table 3 represents a specific strategy utilized by a business owner. This configuration allows two or more strategies from a same business owner. For example, a business owner who has stores in more than one city, would be able to run one strategy for each group of stores by location. Every store has to provide a key identification to run a specific strategy.

The next section shows some examples of the glue code used to make the system perform accordingly to the documented Business Owner requirements.

6.3.7 Implementing the Application Level

The application is running and all business owner strategies are satisfied accordingly. The code is ready to be tested and after passing the testing phase the software platform can be put in production. Maybe a few adjustments during the system integration should be done before the production phase.

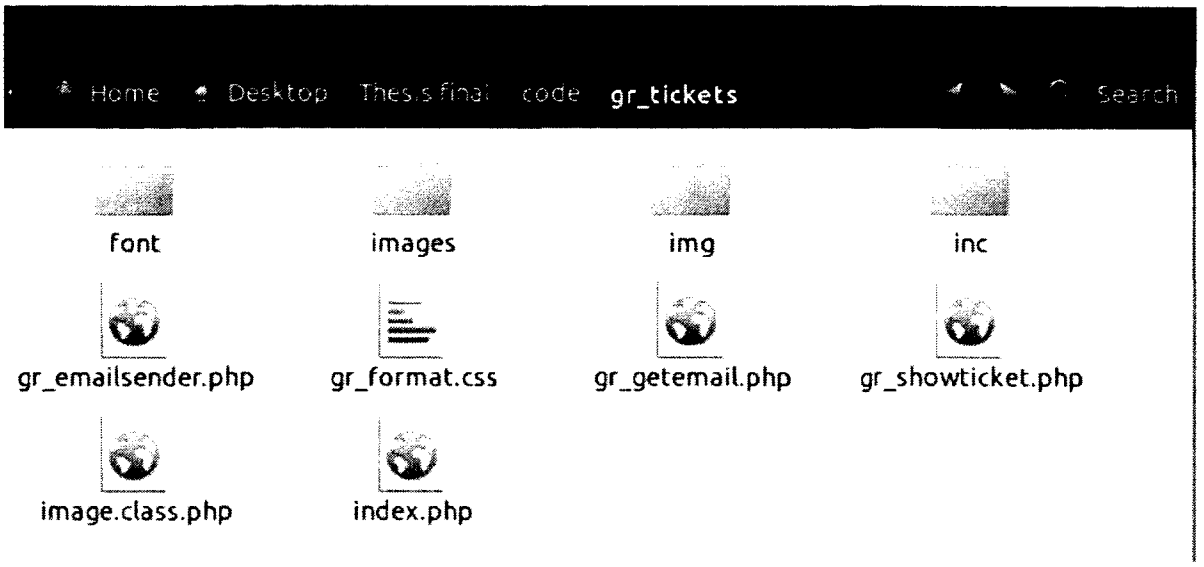


Illustration 13: Folder with some source code files for glue code

This sub-section shows some samples of the code used to create a running configurable software platform as a proof of concept.

The displayed files in Illustration 13 were created to manage the configurable software platform and satisfy the requirements from each Business Owner strategy documented on GRL graphs.

The file index.php is receiving the request from a remote workstation and loads the right configuration based on the received request. Table 4 shows some code fragment for the glue code.

```
  .
  .
  .
// Part A
  $con = MyDB::getInstance();
  $query = $con->selectFrom($table = "random_par", $columns = array('Expiring_Date','send_email'),
$where = array('strategy_ID'=>$strategy_num), $like = true, $orderby = "rp_id", $direction = "DESC",
$limit = null, $offset = null);
  $expiring_date = $query['result'][0]['Expiring_Date'];
  $b_email = $query['result'][0]['send_email'];
  $con = MyDB::getInstance();
  $query = $con->insertInto("Draw_list", $fields = array("Ticket_ID" => $value, "strategy_ID" =>
$strategy_num, "Submitted_Date" => $today, "Expiring_Date" => $expiring_date, "Winner" =>
FALSE));

// Part B
  if ($b_email) {
    echo"</br>An email will be sent";
    echo"</br>The value of b_mail is: ".$b_email;
    echo"</br>The email boolean is TRUE";
    header("Location: gr_getemail.php?code=$value&strategy_n=$strategy_num");
    exit();
  } else {
    header("Location: gr_showticket.php?code=$value&strategy_n=$strategy_num");
    exit();
  }
  .
  .
  .
```

Table 4: Code fragment to load parameters

Part A of this fragment of the code reads the configuration table from software platform code and Part B selects the action via email activiting *gr_getemail.php* code or without email activating *gr_showticket.php* code.

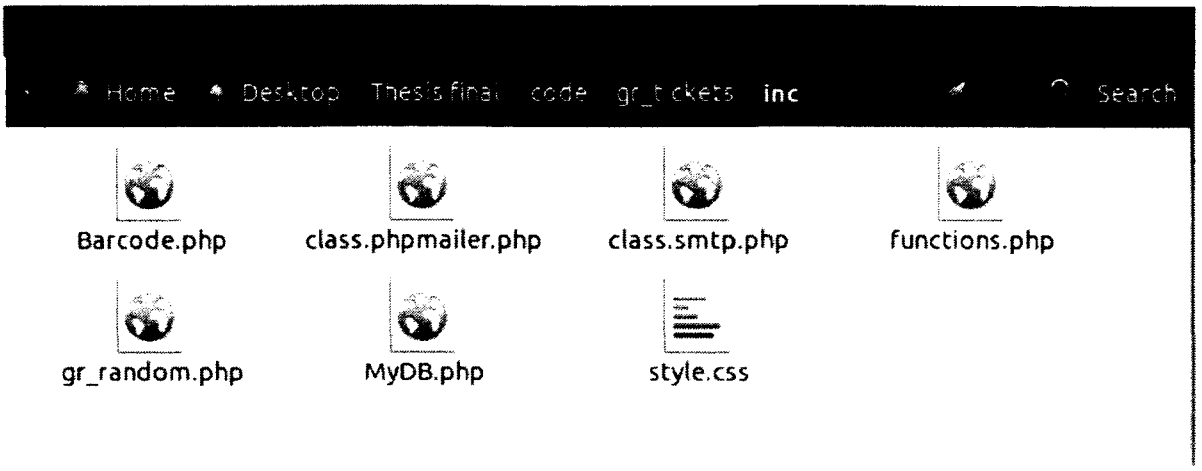


Illustration 14: Component Library Set

The files presented in Illustration 14 are the software component library from configurable software platform team and 3rd party developers. To display some examples of software components in the library set, Table 5 shows a fragment of the code to generate a random ticket number and it is created by the software platform development team.

```
<?php
// will perform all the functions required for generating a random string of user defined length and
specifications
// can be used for generating secure passwords, confirmation links, captcha verification or else.
//          define('__ROOT__',dirname(dirname(__FILE__.'/givearate')));
            require_once(__ROOT__.'/inc/MyDB.php');

class RandomString{
    var $strategy_val;
            protected static $m_pInstance;
            // randomText() - for generating the random string
            // based on user requirements.
            // NULL if nothing specified.
            // return : string
```

Table 5: Fragment of code to generate random ticket number

Table 6 shows a header of the file indicating the information about this software

component. It is a 3rd party software component selected to perform a specific task, generating barcodes as listed in the header file.

```
<?php
/*
 * BarCode Coder Library (BCC Library)
 * BCCL Version 2.0.1
 * Porting : Barcode PHP
 *         DEMONTE Jean-Baptiste
 * Date    : September 25, 2010
 * Author  : DEMONTE Jean-Baptiste (firejocker)
 *         HOUREZ Jonathan
 * Contact : jbdemonte @ gmail.com
 * Web site: http://barcode-coder.com/
 * dual licence : http://www.cecill.info/licences/Licence_CeCILL_V2-fr.html
 *                http://www.gnu.org/licenses/gpl.html
 * Managed :
 *   standard 2 of 5 (std25)                    *   interleaved 2 of 5 (int25)
 *   ean 8 (ean8)                               *   ean 13 (ean13)
 *   code 11 (code11)                           *   code 39 (code39)
 *   code 93 (code93)                           *   code 128 (code128)
 *   codabar (codabar)                          *   msi (msi)
 *   datamatrix (datamatrix)
 * Output :
 *   GD                    *   FPDF
 */
```

Table 6: Header of reused barcode software component

In this case study, the software platform receives a request from a workstation or web application and provides the output as requested by the Business Owner strategy.

The following statement from a workstation shows an example on how it works:

http://host_name_or_address/gr_tickets/index.php?strategy=A56881

The parameter *strategy=A56881*, provides the key information for the glue code to load the right set of data from the configuration table.

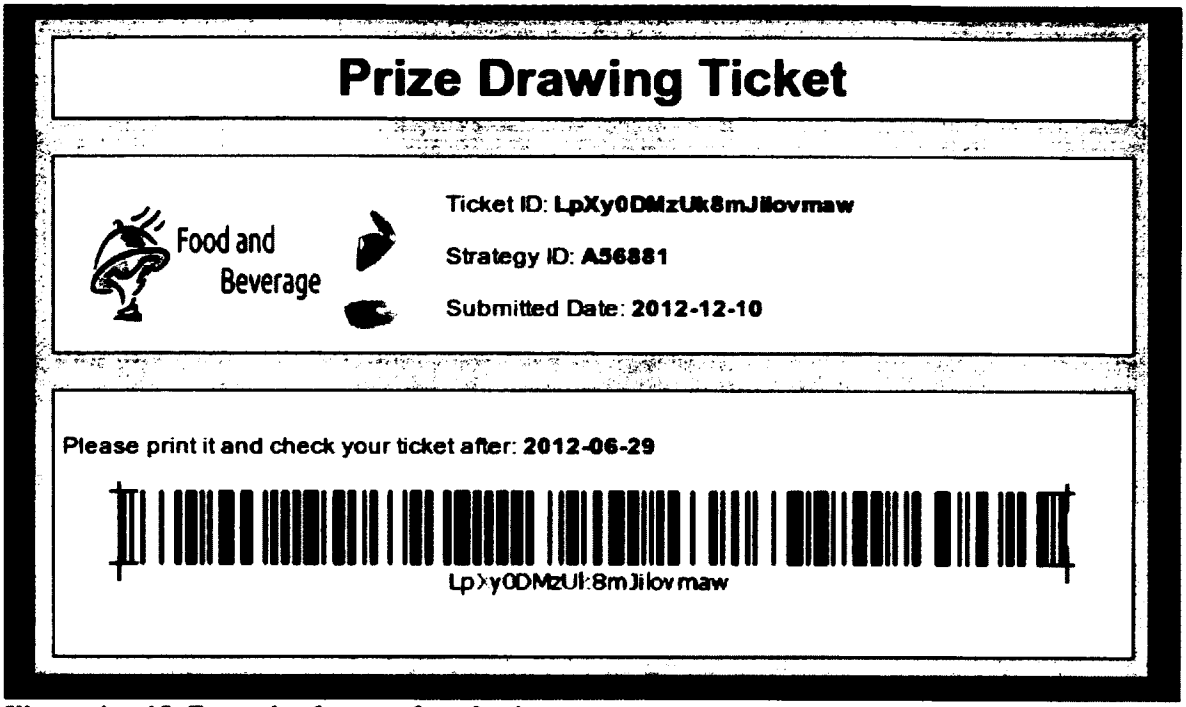


Illustration 15: Example of output for a business owner strategy A

The output is presented in the Illustration 15 showing the generated ticket based only on the strategy A identified as A56881, for this particular case study.

The other two Business Owner strategies (B) and (C) will also send an email to the Business Owner customer after displaying the generated ticket along with its customized text fields.

6.4 Validation

The experiment created for this case study shows that it is possible to create a configuration during the early stage of Requirements Analysis process using URN method.

Perhaps, to satisfy some requirements a 3rd party software component may be necessary to run the software platform application. The combination of the glue code and software components provides a reasonable solution for the software platform development process.

The amount of code as part of the glue code represents 12.31% of the total number of lines of code and 3rd party software development code is representing about 87.69% as shown in Table 7. These percentages are representing the amount of software code as contribution for the software platform implementation process. These percentages do not show the amount of code reused in this case study.

The majority of the code is represented by 3rd party source code (87.69%) for this particular case study. That represents a significant amount of code that does not need to be created from scratch. The other part (12.31%) is code created from scratch.

This sample represents the potential of source code from each component that can be explored in terms of reusing software components and how much would be possible to gain in terms of costs reduction, reduced development time and improved product quality.

Code file	Lines of code approx.	Percentage %	Sub-total %	Component Type
Barcode.php	1228	20.53		Component 3rd party
Class.phpmailer.php	2535	42.38	*87.69*	Component 3rd party
Class.smtp.php	818	13.68		Component 3rd party
MyDB	664	11.1		Component 3rd party
gr_random.php	94	1.57		Local team
Style.css	72	1.2		Local team
gr_emailsender.php	69	1.15		Local team
gr_format.css	47	0.8	*12.31*	Local team
gr_getemail.php	31	0.5		Local team
gr_showticket.php	135	2.26		Local team
image.class.php	288	4.82		Local team
Total	**5981**	**100**	**100**	

Table 7: Distribution between Glue Code and 3rd Party components

From this work we can not claim that there is going to be this ratio of source code for every customizable software platform. However, it is possible to analyze how much of glue code represents the total amount of lines of code for the software platform. The majority of the 3rd party software components are mature enough and have a good quality.

Table 8 shows another way to see the reuse aspect where increasing the amount of customers or stakeholders for the configurable software platform provide several possible combinations and also takes advantage of the potential components reuse demonstrated on Table 7.

If a higher amount of reused code can be implemented in the software platform, it would be expected to get a reduced time to put the product in the market and saved a

considerable amount of time during the software development phase.

Analyzing the glue code, there is reuse of a majority of the existing glue code. That happens because each Business Owner Strategy triggers the glue code and the glue code loads the specific parameters for the Business Owner Strategy.

Reused component by the Configurable Software Platform			
Business Owner Strategy	**Ticket**	**Email**	**Login (Loyalty)**
A	yes	no	no
B	yes	yes	no
C	yes	yes	yes

Table 8: Reuse of Groups of Components

Furthermore, most of components' behaviour are customized throught the configuration table, which does not affect others behaviour when we make a change in one specific owner's strategy. The parameters affect only one specific customer or stakeholder requirements.

The use of jUCMNav as modelling tool makes the requirements analysis process more consistent to create the requirements model (GRL and UCM) and when one is adding additional stakeholders does make the modelling and analysis activities more efficient because the tool manages all input data very well.

Althougether, adding a new customer or stakeholder provides a smooth sequence of activities both for requirements modelling and configuration.

The next chapter discusses the results of this research.

Chapter 7: Discussions

There are several works related to reducing the gap between requirements phase and development process stage as shown in the Chapter 2. Existing research on goal-oriented and scenarios provides some alternatives to solve or reduce this gap. However, they are focusing the software application with market or technology-driven approaches, giving less alternatives for users who need some operational business customization.

7.1 Research Results

The software providing service to business can also offer a less restricted solution for the service buyers to their business process. This technique helps the software developers to adapt the solution to their service buyer business process adding more value to the product.

Illustration 16 presents a possible solution to make a software platform performing a variety of features to satisfy a larger number of requirements for the service buyer. The question is how to design this software platform during the early stage of the RA process. For this architecture (partial) the problem resides on how to develop a configuration table and the glue code that does not increase the complexity of the system.

Considering the amount of research found in the literature using goal-oriented and scenarios to develop solutions for SPL, it looks valuable that the same approach could be applied for the configurable software platform as presented in this work.

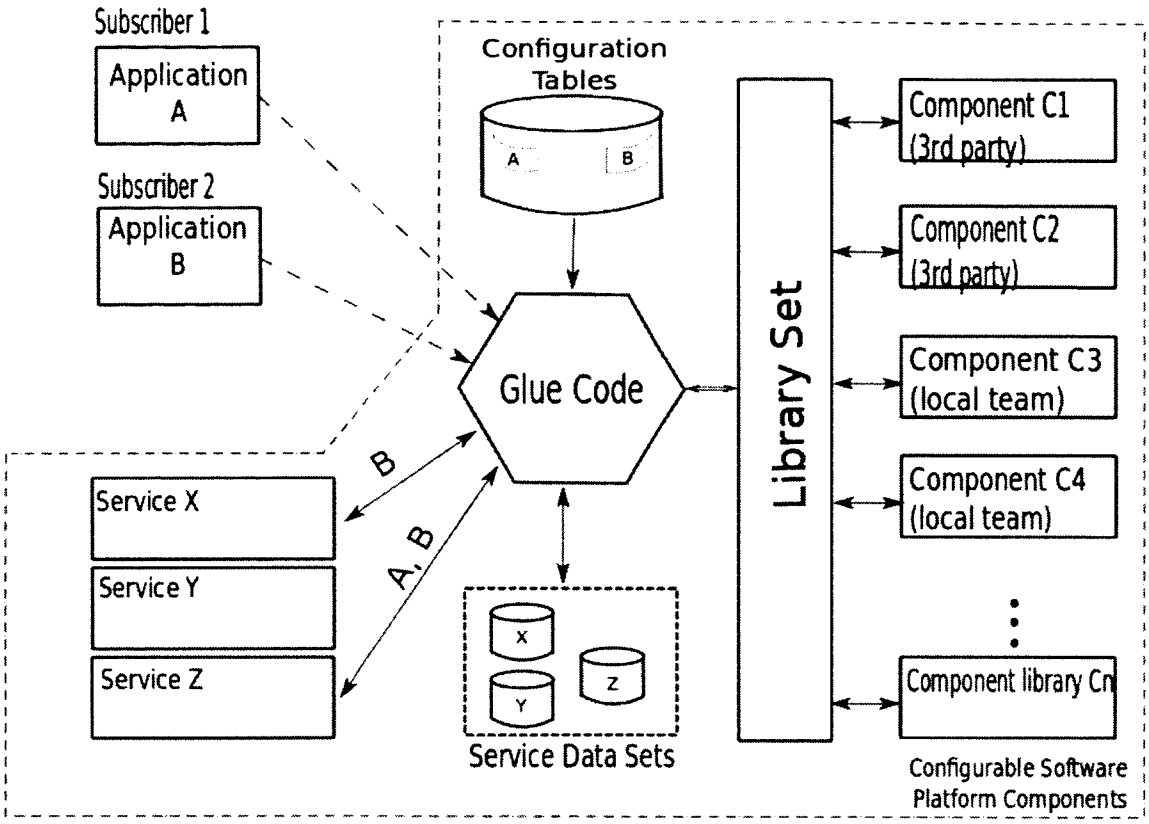


Illustration 16: General Architecture for Configurable Software Platform

Some approaches are focusing more in terms of a more formal design which make them more accurate to perform the analysis. However, the majority of software developers will not be interested in creating a more formal design for a more accurate analysis, especially if that is a task that should be done manually providing all formal specifications.

During the work in this research, a premise was established to provide a solution that would attract software developers to implement the technique without a formal approach. URN method is a potential candidate because it has been used by the telecommunications industry for almost a decade to design a variety of systems.

A software tool is a valuable asset to be considered because it provides a better and faster way to design a system. The software tool jUCMNav is up-to-date and covers most of features from the URN method. It is relatively easy to learn and use. Furthermore, GRL is known as modelling tool to document NFRs and UCM as modelling tool to document functional requirements.

The output from both models provides a way to identify variability and commonality in the RA process. It helps to define what is part of the configuration table (variability) and what is part of the library set (commonality), see Illustration 16.

From GRL graphs and UCM models, it is possible to perform an analysis to identify and match existing library of components and provide a potential solution to create a configurable software platform design. Evaluation of these potential reusable components lead to create the configuration table to satisfy all the strategies from GRL graphs and to design the glue code to satisfy all UCM models. The reusable components matching activity was done manually during the experiment. There is not enough dataset at this point to provide some automation for this activity.

Illustration 17 shows a general idea of how the proposed technique works in solving the research question through GRL graphs, UCM models, existing library components analysis, configuration table creation and glue code design.

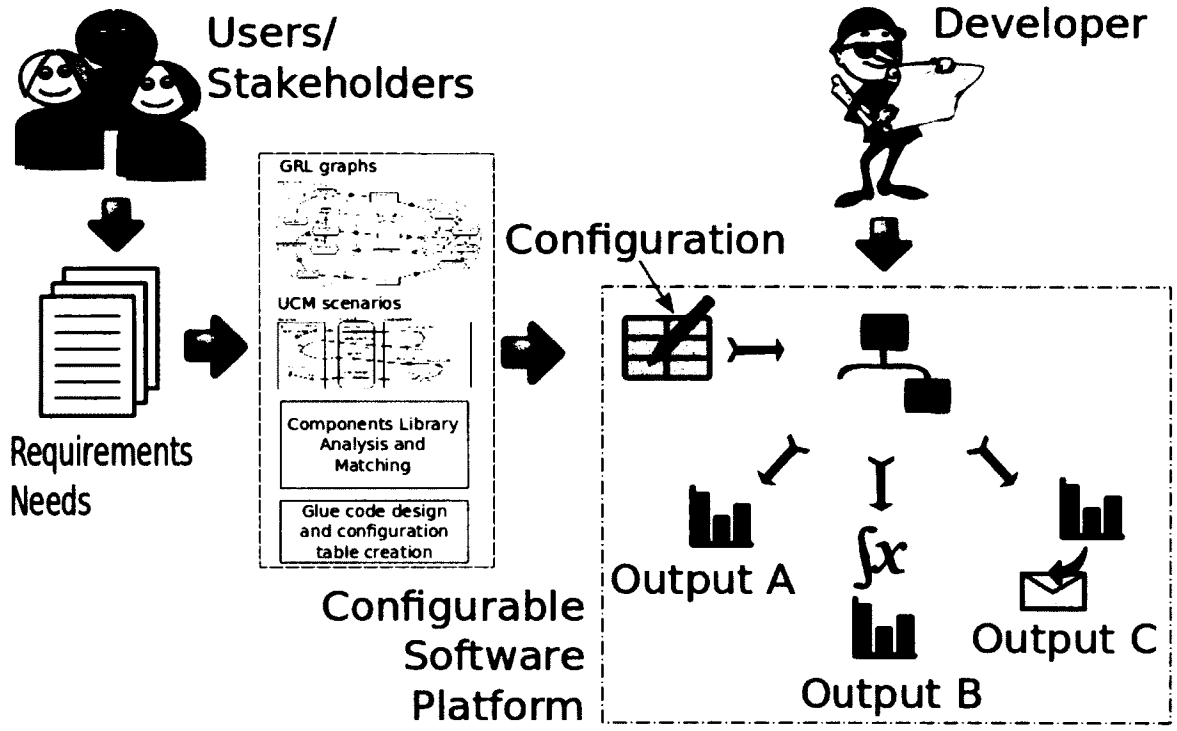


Illustration 17: Proposed Technique to solve the gap question

7.2 Business Opportunity

In the software development industry, the requirements analysis technique presented here can make some contribution for software platform developers and owners. It may provide potential opportunities for business innovation when combining business process modelling and requirements analysis process.

As a business opportunity, the resulting outcome from this research is offering a novel way to provide a service that could be easily adapted to a specific business owner process might attract other business owners to buy services from the software platform once the offered service of software platform process would adapt its service matching better to the business owner strategy.

This technique is also adherent with business modelling process and allows Business Analysts and Software Developers work along with a solution that needs refinement levels based on distinct types of software platform customers. As an example, it has a good potential to be used along with business modelling process as created by Ries (2011).

As a complementary technique/tool for business modelling process, it brings an advantage of allowing system developers to try two or more alternatives at the same time. A dynamic business modelling process needs a complementary technique/tool to help Business Analysts to create experiments through by using a test trial approach. A similar approach is currently used by some Marketing companies to evaluate how a customer can behave in a website environment (i.e. http://www.gnim.ca[7]).

Weiss & Amyot (2007) present URN method as a lightweight business modelling approach where business managers would be able to manage business model process in a systematic and incremental way and make a quick evaluation of alternative business models. In this case, the proposed technique can be easily adapted to provide means for software developers to analyze and evaluate a software implementation for the selected alternative of business model.

The next chapter provides some insights about limitations and future work from this research.

7 This company is in its 3rd phase as part of Lead To Win (LTW) program (http://www.leadtowin.ca/). LTW is a business development program and encompasses all growth-oriented businesses in the National Capital Region, Ottawa, Canada.

Chapter 8: Conclusions, Limitations and Suggestions for Future research

The positive result from this experiment provides a valuable technique to uncover configuration for software platform during the early stage of the requirements analysis process. This experiment is also proving that one who is interested in designing and implementing a customizable software platform can do it with a shorter learning curve using well know software requirements modelling tools from the software industry. There are some potential contributions for business modelling process and software development related to SPL, software components and architecture. The following sections present conclusions, limitations and suggestions for future research.

8.1 Conclusions

This research showed a possible solution to solve the gap between Requirements Analysis process and software implementation phase. The case study is a proof of concept to show how is possible to create a configuration for configurable software platforms during the early stage of RA and potentially may impact increase quality, reduce costs and time to the market.

The outcome from this research shows that a software platform may be designed in a more flexible way accommodating distinct requirements from business owners interested in buying some services from the software platform. Consequently, it allows the software

platform owner to serve more customers, at a lower development and maintenance costs, to the offered services because the service is more customizable in terms of business process and may generate a higher revenue.

The proposed technique uses the current notation for GRL graphs and UCM models as a modelling tool along with the use of jUCMNav tool which incorporates many features from URN method supporting the early requirements stage that the proposed technique requires. As an additional advantage, having jUCMNav developed on the Eclipse modelling platform, is a key element because other software tool developers can add new plug-ins to the URN modelling tool to generate source code based on design templates or frameworks.

The contribution this research brings to the software development industry is to demonstrate how a software developer can interpret the models and provide a reasonable solution for the configurable software platform creating a component library set for commonality and designing specific glue code for variability.

This experiment shows enough potential to answer the research question, it has been shown as a potential use for configurable software platform development. As a result, it may be fast and flexible to be incorporated in software product line development approach in performing two or more alternatives at the same time.

This work shows how that would be possible for practitioners to apply requirements analysis and get a preliminary design for a configurable software platform. It also may fit

with some SPL methods and may be incorporated along with business modelling process using a common notation for both processes. This opens a potential opportunity for business analysts and software developers to share a common notation and tool to create a more integrated process to develop new systems or maintain existing ones. This combination has potential to provide benefits such as cost reduction, error prone reduction by missed requirements, fast time to the market and improved management process.

The next section presents some limitations from this work imposed by time restrictions and trimmed scope range.

8.2 Limitations

This research is limited in terms of a proof of concept and it was not intended to cover all the aspects of elicitation, analysis and implementation of a configurable software platform development process. It scratches the surface of this complex matter and provides a potential research opportunity to identify, analyze and evaluate an identified configuration during the requirements analysis stage.

The selected case study is also limited in terms of amount and variety of software components. However, it does not diminish the potential of using this technique in a larger software platform environment. Furthermore, it is using some similar approach for demonstrating the modelling process as proposed by other researchers.

A supporting layer for management and administration set of activities is not conceived

in this research. All tasks to create, modify or delete variability parameters were made manually. The selection of potential components for reuse were analyzed from scratch. An administration layer should be developed to automate configuration tasks for the software platform.

This work does not cover the components and configuration management because it is not part of the original scope of the research project. Other methods similar to URN may or may not produce the same results to identify a configuration table for configurable software platform.

Finally, the next section provides some suggestions for future work relating business innovation and software development technology.

8.3 Suggestions for Future Research

For future research, there are some suggestions that can be made such as comparative study for similar approaches using goal-oriented and scenarios with more formal approach; evaluation of this technique considering a more complex system like Content Management Systems (CMS), Enterprise Resource Planning (ERP) or Customer Relationship Management (CRM); creating a combined approach using Business Process Modelling (BPM) and Software Development Process (SDP) supported by the URN method to evaluate and implement business process changes.

For the business innovation perspective, there are some opportunities to evaluate which benefits can be identified both for configurable software platform owners and business

owners interested in buying offered services by the software platform. In the same perspective, considering a cloned software platform how much of its glue code can be reused from the original one, along with its components library set interchangeability between them.

In addition, there is a potential opportunity to combine the proposed technique with business model process, as proposed by Weiss & Amyot (2007). This combination could perform the whole process from business analysis to software implementation in an integrated fashion, bringing some benefits for business managers.

The missing layer for management and administration of configurable software platform is a work to be developed to increase usability, maintenance and control over reusable components and elements of variability dimension in the platform. Furthermore, a project showing a GRL and UCM models for the administration layer could be developed and implementated to improve the concept presented in this work.

Bibliography

Ahmed, F, and L Capretz. 2008. "The Software Product Line Architecture: An Empirical Investigation of Key Process Activities." Information and Software Technology 50 (11) (October): 1098–1113.

Ajila, Samuel a., and Ali B. Kaba. 2008. "Evolution Support Mechanisms for Software Product Line Process." Journal of Systems and Software 81 (10) (October): 1784–1801.

Ajila, S.A., and P.J. Tierney. 2002. "The FOOM Method-modeling Software Product Lines in Industrial Settings." In Proceedings of the 2002 International Conference on Software Engineering Research and Practice (SERP02), Las Vegas, Nevada, USA: CSREA Press.

Amyot, Daniel. 2003. "Introduction to the User Requirements Notation: Learning by Example." Computer Networks 42 (3) (June 21): 285–301.

Amyot, Daniel, Sepideh Ghanavati, Jennifer Horkoff, Gunter Mussbacher, Liam Peyton, and Eric Yu. 2010. "Evaluating Goal Models Within the Goal-Oriented Requirement Language." International Journal of Intelligent Systems 25: 841–877.

Amyot, D. & Mussbacher, G. 2001. Bridging the requirements/design gap in dynamic systems with use case maps (UCMs), *in* 'Proceedings of the 23rd International Conference on Software Engineering', *IEEE Computer Society,* Washington, DC, USA, pp. 743–744.

Amyot, Daniel, and Gunter Mussbacher. 2001. "Bridging the Requirements/design Gap in Dynamic Systems with Use Case Maps (UCMs)." *in* 'Proceedings of the 23rd International Conference on Software Engineering', *IEEE Computer Society,* Washington, DC, USA: 743–744.

Arbab, Farhad, and Sun Meng. 2008. "Synthesis of Connectors from Scenario-Based Interaction Specifications." CBSE 2008, LNCS 5282: 114–129.

Arboleda, Hugo, R. Casallas, and J.C. Royer. 2009. "Dealing with Fine-grained Configurations in Model-driven Spls." In Proceedings of the 13th International Software Product Line Conference, 1–10. Carnegie Mellon University.

Asikainen, Timo, Tomi Männistö, and Timo Soininen. 2007. "Kumbang: A Domain Ontology for Modelling Variability in Software Product Families." Advanced Engineering Informatics 21 (1) (January): 23–40.

Bosch, J., 2000. "Design and use of software architectures: adopting and evolving a product-line approach." *Addison-Wesley Professional.*

Buhr, R. J.A. & Casselman, Ron S., 1995. "Use Case Maps for Object-Oriented Systems." *Prentice Hall,* 302 pages.

Clements, P., & Bass, L. 2010. "Business goals as architectural knowledge." *Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge - SHARK '10*, 9–12.

Clements, P., & Northrop, L., 2001. "Software Product Lines: Patterns and Practice." *Addison Wesley.*

Dahiya, D., & Sachdeva, R. 2006. "Understanding Requirements: Aspect Oriented Software Development." *30th Annual International Computer Software and Applications Conference (COMPSAC'06)*, 303–308.

Eriksson, M. 2007. "Engineering Families of Software- Intensive Systems using Features, Goals and Scenarios." *Science*. Umeå universitet.

Eriksson, M., Börstler, J., & Borg, K. 2005. "The PLUSS approach–domain modeling with features, use cases and use case realizations." *Software Product Lines*, 33–44.

Galster, M., Eberlein, a., & Moussavi, M. 2006. "Transition from Requirements to Architecture: A Review and Future Perspective." *Seventh ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD'06)*, 9–16.

George, B., Fleurquin, R., & Sadou, S. 2008. "A Component Selection Framework for COTS Libraries." *CBSE 2008, LNCS*, 5282: 286–301.

Gruler, A., Harhurin, A., & Hartmann, J. 2007. "Development and Configuration of Service-based Product Lines." *11th International Software Product Line Conference (SPLC 2007)*, 107–116.

Horkoff, J., & Yu, E. 2011. "Analyzing goal models: different approaches and how to choose among them." *Proceedings of the 2011 ACM Symposium on Applied*, 675–682.

ITU-T – International Telecommunications Union: Recommendation Z.151 (11/08). 2008. "User Requirements Notation (URN) - Language Definition". Geneva, Switzerland, 190 pages.

Jarke, M., Klamma, R., Pohl, K., & Sikora, E. 2010. "Requirements Engineering in Complex Domains." *LNCS - Lecture Notes in Computer Science*, 5765: 602–620.

Lamsweerde, A. V. 2001. "Goal-oriented requirements engineering: A guided tour." Requirements Engineering.

Leite, J. C. S. P., Hadad, G. D. S., Doorn, J. H., & Kaplan, G. N. 2000. "A Scenario Construction Process." *Requirements Engineering*, 5(1), 38–61.

Liu, L., & Yu, E. 2001. "From Requirements to Architectural Design – Using Goals and Scenarios." University of Toronto. ftp://ftp.white.toronto.edu/cs/ftp/pub/eric/STRAW01-R2A.pdf, November 30, 2011, Toronto.

Liu, L., & Yu, E. 2004. "Designing Information Systems in Social Context: a Goal and Scenario Modelling Approach." Information Systems 29 (2) (April): 187–203. doi:10.1016/S0306-4379(03)00052-8.

Mannion, M., & Kaindl, H. 2008. "Using parameters and discriminants for product line requirements." *Systems Engineering*, 11(1): 61–80.

Niehaus, E., Pohl, K., & Böckle, G. 2005. "Product Management. Software Product Line Engineering": 163–192. Springer Berlin Heidelberg.

Pohl, K., & Sikora, E. 2007. "The co-development of system requirements and functional architecture." *Conceptual Modelling in Information Systems Engineering*, 229–246.

Pohl, K., & Ulfat-Bunyadi, N. 2005. "Selecting High- Level COTS Components." *Software Product Line Engineering*: 285–301. Springer Berlin Heidelberg.

Rabiser, R., O'Leary, P., & Richardson, I. 2011. "Key activities for product derivation in software product lines." *Journal of Systems and Software*, 84(2): 285–300.

Ries, E., 2011. "The Lean Startup". Crown Business, 320 pages.

Schewick, B. Van. 2004. "Architecture & Innovation The Role of the End-to-End Arguments in the Original Internet." Technical University of Berlin.

Szyperski, C. 2002. "Component Software: beyond object-oriented programming." (C. Szyperski, D. Gruntz, & S. Murer, Eds.) (Second Edi.): 589 pages, Addison-Wesley Professional.

Ullah, M. I. 2009. "COPE + : A method for design and evaluation of product variants": 1–35, University of Calgary, Calgary.

Weiss, Michael, and Daniel Amyot. 2005. "Business Process Modeling with URN." International Journal of E-Business Research 1 (3): 63–90.

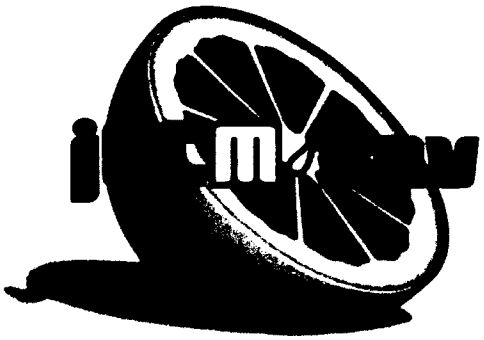
Weiss, Michael, and Daniel Amyot. 2007. "Chapter 13 Business Model Design and Evolution." In Challenges in the Management of New Technologies - Vol 1, ed. Marianne Hörlesberger, Mohamed El-Nawawi, and Tarek Khalil, 183–194. Hackensack, NJ: World Scientific Publishing Co. Pte. Ltd.

Weiss, D., & Lai, C. T. R. 1999. "Software Product-Line Engineering: A Family-Based Software Development Process": 448. *Addison Wesley*.

Appendices

Appendix A: Report generated by jUCMNav tool

jUCMNav Report



http://softwareengineering.ca/jucmnav

Title:	case_study-thesis
URN Model Name:	URNspec
Description:	
Author:	Antonio Misaka
Creation Date:	December 3, 2012 10:02:59 AM EST
Modification Date:	January 16, 2013 05:52:57 PM EST
Report Generation Date:	January 18, 2013 12:08:47 PM EST
Specification Version:	41

UCM Scenario Groups documentation

- ScenarioGroup6:
  1. ScenarioDef7

Intentional Elements

1. More sales
2. Collect sales receipt #
3. Give a ticket #
4. Draw a prize
5. Level of satisfaction
6. Sale receipt
7. Prize
8. Service/Product
9. Get more promotions
10. Win a prize
11. Participate in the promotion
12. Collect the prize
13. Make an advertisement
14. Ticket
15. Tickets to draw
16. Email address
17. Collect email address
18. Get the ticket
19. Provide information
20. Access campaign website
21. Login and provide sales receipt #
22. Redeem the reward
23. Verify ticket to reward
24. Get a reward
25. Reward
26. Reward to give away

Actors

1. Business Owner
2. Consumer

System Login

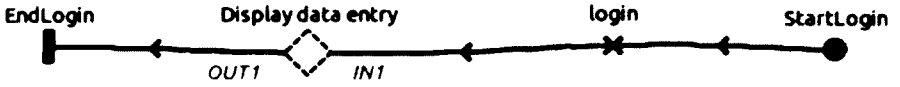


Figure 1 - System Login

Responsibilities

login

Stubs

Dynamic Stub - Display data entry

Plugin Map - Get receipt form

Input Bindings:

IN 1 <-> StartReceipt

Output Bindings:

OUT 1 <-> EndReceipt

Precondition:

Expression: true

Transaction: false

Probability: 1.0

Email ticket

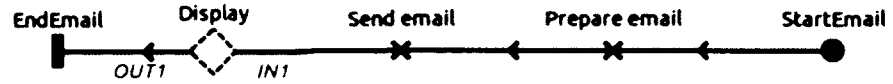


Figure 2 - Email ticket

Responsibilities

Prepare email

Send email

Stubs

Dynamic Stub - Display

Metadata: " hits" = "0"

Plugin Map - Display ticket

Precondition:

Expression: true

Transaction: false

Probability: 1.0

Start Point

StartEmail

Metadata: " hits" = "0"

End Points

EndEmail

Metadata: " hits" = "0"

Get receipt form

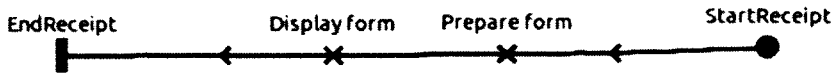


Figure 3 - Get receipt form

Responsibilities

Prepare form

Display form

Start Point

StartReceipt

Metadata " hits" = "0"

End Points

EndReceipt

Metadata " hits" = "0"

GRLGraph - strategy B

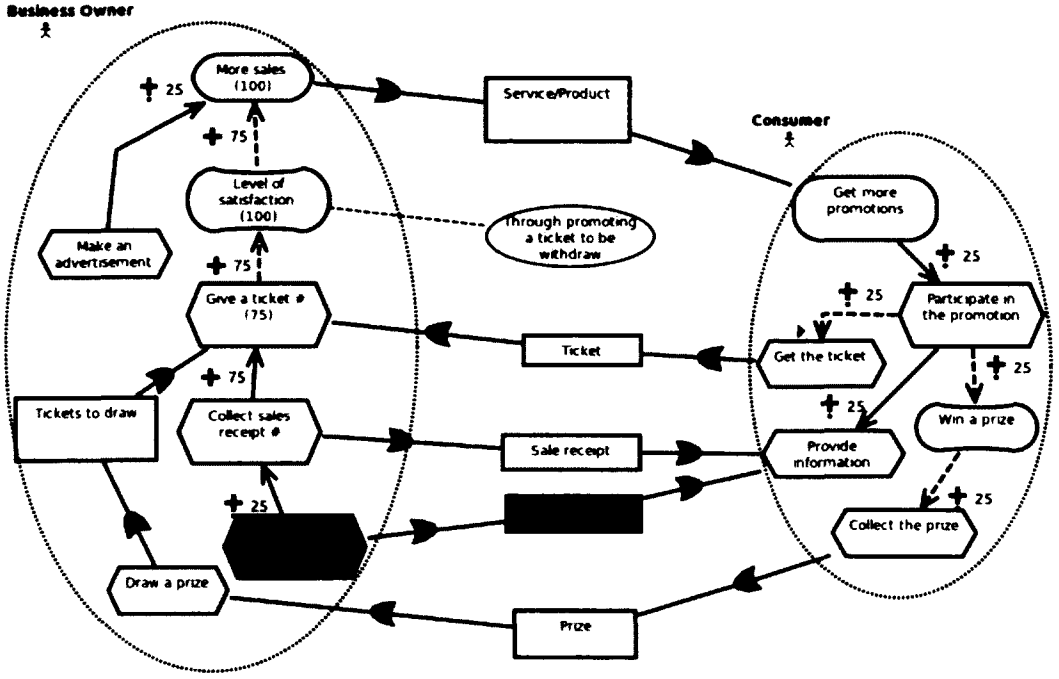


Figure 4 - GRLGraph - strategy B

Beliefs

Belief91: Through promoting a ticket to be withdraw

Metadata

" width" = "150"

" height" = "58"

Intentional Element URN Links

Get the ticket ==> (UCMmap)Creates ticket

Link Type: Refine

Display ticket

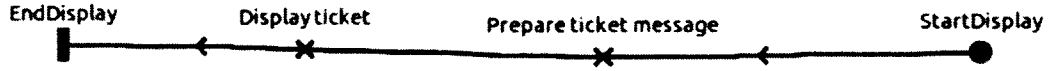


Figure 5 - Display ticket

Responsibilities

Prepare ticket message

Display ticket

Start Point

StartDisplay

Metadata " hits" = "0"

End Points

EndDisplay

Metadata " hits" = "0"

Creates ticket

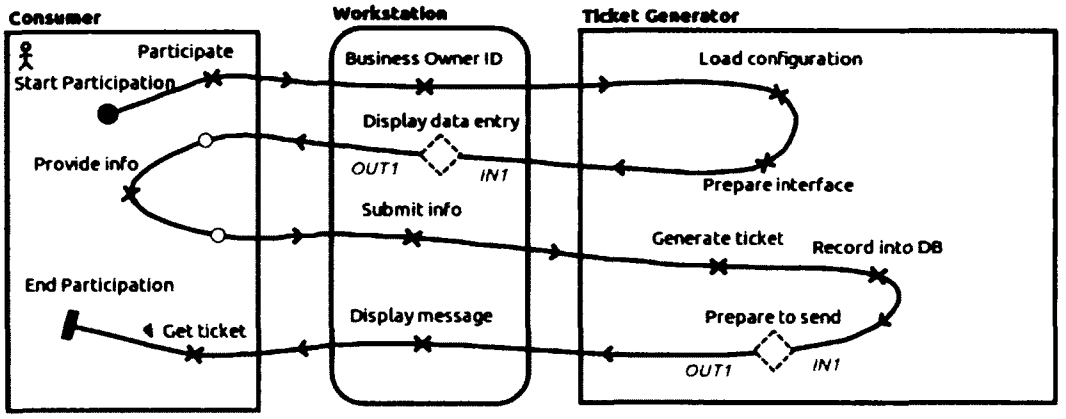


Figure 6 - Creates ticket

Description

It provides an overall representation about the ticket generation software platform.

Map URN Links

Get the ticket (Task)

Responsibilities

Participate

Business Owner ID

Load configuration

Prepare interface

Provide info

Submit info

Generate ticket

Record into DB

Display message

Get ticket

Stubs

Dynamic Stub - Display data entry
Metadata: " hits" = "0"

Plugin Map - Get receipt form
Input Bindings:
IN 1 <-> StartReceipt
Output Bindings:
OUT 1 <-> EndReceipt
Precondition:
Expression: true
Transaction: false
Probability: 1.0

Plugin Map - System Login
Input Bindings:
IN 1 <-> StartLogin
Output Bindings:
OUT 1 <-> EndLogin
Precondition:
Expression: true
Transaction: false
Probability: 1.0

Dynamic Stub - Prepare to send
Metadata: " hits" = "0"

Plugin Map - Display ticket
Precondition:
Expression: true
Transaction: false
Probability: 1.0

Plugin Map - Email ticket

Precondition:

Expression: true

Transaction: false

Probability: 1.0

Start Point

Start Participation

Metadata: " hits" = "0"

End Points

End Participation

Metadata: " hits" = "0"

GRLGraph - strategy C

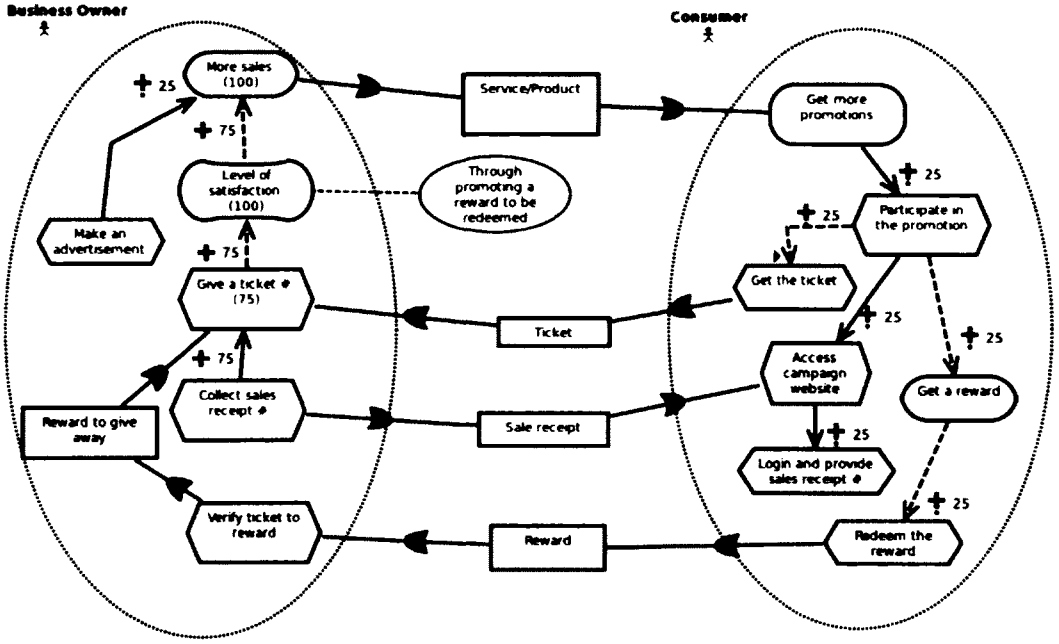


Figure 7 - GRLGraph - strategy C

Beliefs

Belief274: Through promoting a reward to be redeemed

Metadata

- "width" = "143"
- "height" = "76"

Intentional Element URN Links

Get the ticket ==> (UCMmap)Creates ticket

Link Type: Refine

GRLGraph - strategy A

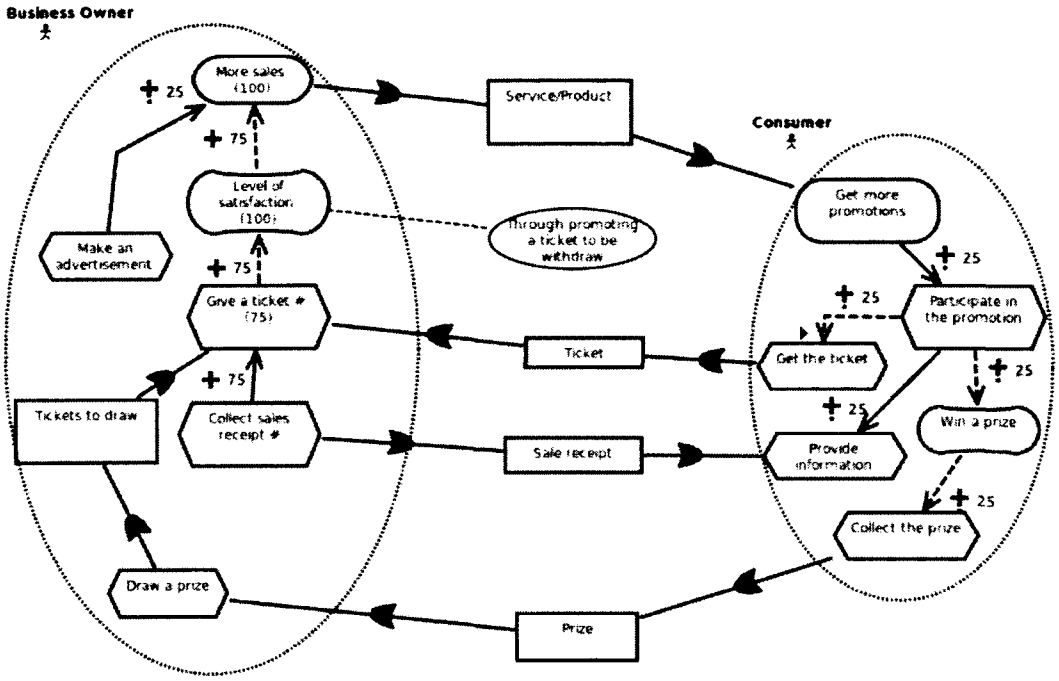


Figure 8 - GRLGraph - strategy A

Beliefs

Belief - promoting ticket to be withdraw: Through promoting a ticket to be withdraw

Metadata

" width" = "150"

" height" = "58"

Intentional Element URN Links

Get the ticket ==> (UCMmap)Creates ticket

Link Type: Refine