RSPL:
A Conceptual framework for interactively building Product Line Requirements

Kabanda S.K* and Adigun M.O**
* Dept. of Computer Science, University of Zululand, sarah_kabanda@yahoo.com
** Dept. of Computer Science, University of Zululand, madigun@pan.uzulu.ac.za

Abstract – The generation of requirements during the development of a family of systems is the responsibility of domain experts. Our RSPL (Requirements Specification for a Product Line) framework proposes a strategy for eliciting, generating and documenting requirements from domain resources, including domain expert, domain models, domain vocabulary and domain features. The framework adopted the model driven architecture as a pattern for a requirements generation process that (i) transforms domain knowledge into requirements; and that (ii) automatically splits them into commonalities and variations among the products that will make the product line. In order to get requirements dynamically updated as domain sources update the knowledge repository, we have proposed to use the portlet technology implementation strategy.

Keywords – Model Driven Architecture, Product Line Requirements, Second Generation Portal Technology

I. INTRODUCTION

The generation of requirements during system development is the responsibility of domain experts who are knowledgeable enough to elicit and model requirements using various resources. Recently, tools have become available for automatic generation of consistent, current, audience-specific requirements specifications [1, 2, 3]. In spite of this development, inadequate user involvements in the generation process have been found to compromise the requirements engineering process [4]. In this work, we provide an elicitation strategy that relies on dynamic content that can be customized by users and subsequently personalized as the environment changes.

The focus of the work is on software development of a family of systems. It is therefore assumed that reusability, maintenance and assembly are issues that must drive the requirements generation process. Assembly demands that requirements must be decomposable into reusable assets. Reusability connotes that requirements must be constrained to give rise to both common and variable features of the architecture and individual products respectively. Maintenance means the requirements can dynamically change as the product features evolve and take new forms [8, 9, 10].

The framework for Requirements Specification for a Product Line (RSPL) proposes a strategy for eliciting, generating and documenting requirements from domain resources such as model, domain expert, domain vocabulary and domain feature database derived from legacy systems. RSPL adopted the model driven architecture as a pattern for a requirements generation process that (i) transforms domain knowledge into requirements; and that (ii) automatically splits them into commonalities and variations among the products that will make the product line. In order to get requirements dynamically updated as domain sources update the knowledge repository, we have proposed to use the portlet technology implementation strategy.

The contribution of this work is that requirements generation is being brought into the realm of model driven architecture. This means that in future, domain resources can be modeled as portlets and requirements generated dynamically always reflecting the current state of the target family of systems. The rest of this paper is organized as follows: Section II describes how RSPL framework derives from existing work in Requirements Engineering and Model Driven Architecture. The framework itself is the subject of discussion in section III. In order to illustrate how the framework is practiced, we present a web store example in section IV. The paper is concluded in section V.

II. RELATED WORK

A. Requirement Engineering Tools

Among existing requirements tool from which this work has drawn inspiration are: (i) the frame-based requirements engineering tool (FBRET) and (ii) Requirements Generation Markup Language Processor [1, 2, 3, 4]. They demonstrate how requirements can be elicited in an interactive manner but for only one system at a time.

One of the most used approaches in Product line development is domain modeling [7, 8] approach. In this approach a centralized library of components is created and used by a generation facility to produce the target application. Software engineers have developed the FAST and the Product Line Software Engineering Commonality and Variability Elicitation (PuLSE-CaVE) approaches in dealing with product line development [3, 9, 10, 11, 12, 13, 14, 15, 18, 19]. Most of these approaches incorporate the process of scoping and identifying commonality and variability (SCV) thereby giving

software engineers a systematic way of thinking about and identifying the product family they are creating. Among other things, helping developers create a design that contributes to reuse and ease of change, predicting how a design might fail or succeed as it evolves, and identifying opportunities for automating the creation of family members.

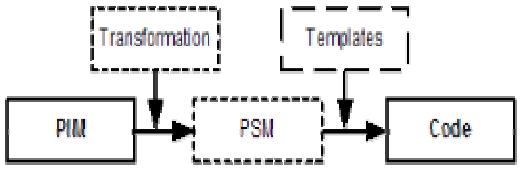

Figure 1: The flow of an MDA generator [15]

The point of departure from existing tools is that RSPL framework recognizes the contribution of domain sources such as models, expert, vocabulary and feature database to provide own perspective of the domain knowledge (see figure 3). Therefore, the framework uses the Model Driven Architecture (Figure 1) as a pattern for requirements generation.

The fundamental model in the new RSPL architecture consists of what the domain sources contribute to domain knowledge. By conceptualizing MDA as a template-based transformation pattern, we imply that a model exists that needs to be transformed from a PIM form to a PSM form. Based on the vocabulary borrowed from MDA [15], the RSPL platform independent model, PIM is the domain knowledge that needs to be transformed into requirements. RSPL defines its own domain templates for scoping requirements output into either architectural commonality or family members' variability. These templates are mapped to platform-specific-model, PSM of MDA. Finally, rather than generate code, RSPL generates requirements in textual form. In other words, five different domain sources (developer, model, vocabulary, expert and feature Database) constitute the Domain Knowledge. Each of these can be represented as dynamic contents modeled as portlets in the implementation. Figure 2 captures the foregoing description of the model-driven requirements generation framework.

III. THE REQUIREMENT SPECIFICATION FOR PRODUCT LINE (RSPL) FRAMEWORK

RSPL defines four phases represented as processes 1 to 4 in the Data flow diagram of Figure 3. Each phase is further decomposed into framework activities as summarized in Table 2. The following subsections describe the phases of the framework

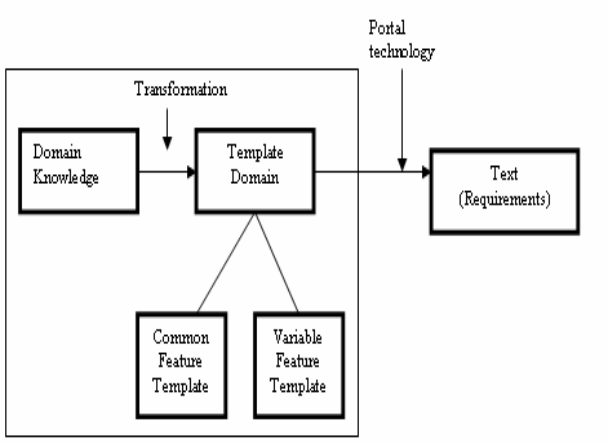

Figure 2: The flow of a Model Driven Requirements Text generator

TABLE I

MDA code generator VS MDA Textual generator

MDA CODE GENERATOR	MDA REQUIREMENTS TEXT GENERATOR
PIM	Domain Knowledge
PSM	Domain Templates
CODE	Textual Requirements

A. Requirements Elicitation

Elicitation consists of the user interacting with framework artifacts via a portal-type interface. In order to elicit requirements for a particular domain feature, the requirements engineer (playing the role of developer/domain expert) chooses from some existing domain knowledge as the starting point. These are called templates. The engineer either chooses to leave the template unchanged or introduces own perspective thus modifying the original template into a new one. Depending on the role being played by the engineer, either the elicitation database or the domain vocabulary is updated. While the developer role modifies the elicitation database, the domain expert role updates the domain vocabulary. It is possible that none of the domain repository need update in which case requirements can be generated by proceeding to phase 4. Expert knowledge can be derived from analyzing customer survey, and market and product planning data; these are then fed into the framework by the domain expert during elicitation.

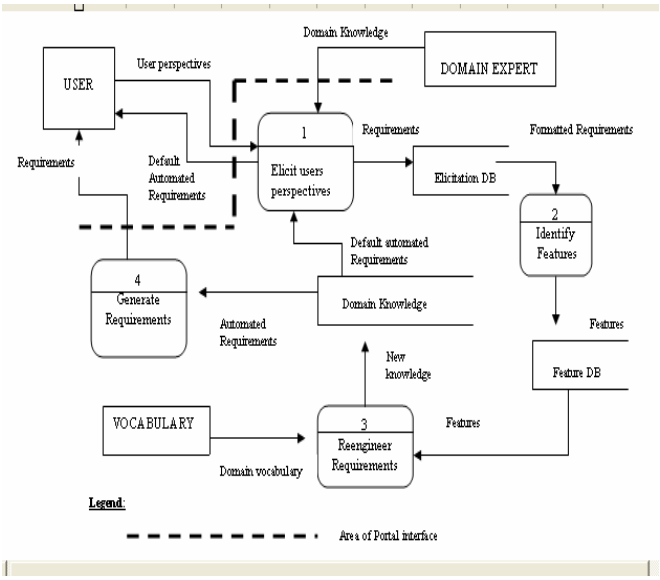

Figure 3: Product Line Requirements Generator Framework

B. Feature Identification

The *Feature identification* process is responsible for specifying commonalities and variabilities among members of the family to identify potential areas of reuse. Identified features are documented using a Product Line Requirement Specification document to represent the family's common requirements as well as the allowed variations that distinguish family members. Captured common requirements builds up to a domain language that represents the executable artifacts of the business processes, entities and relationships that become core assets stored in the domain knowledge for future use.

Once reusable components have been identified, they are further enriched with a domain vocabulary through the reengineering process that improves communication among system developers and also assuring the use of a common and controlled vocabulary in both functional and non functional requirements [21]. The domain vocabulary becomes the Knowledge Base possessing legacy system information, existing documentation, business objects and other organizational system information that form the core assets for future system development.

C. Requirements Reengineering

The essence of the requirements engineering phase is to update the domain knowledge to new knowledge that exists in the system which has not yet being used to generate requirements already. New knowledge derives from updates to domain vocabulary and modified feature database. Recently identified variable features that were elicited are incorporated into the domain vocabulary to ensure that future need for such a requirement will be met without eliciting it again. The new vocabulary becomes incorporated in the existing domain knowledge.

D. Requirements Generation

This phase is built on the premise that a model-driven requirement generation framework is superior to existing requirements generation processes. The model that drives this phase is domain knowledge derived from various domain sources such as developer, domain expert, domain vocabulary, domain feature database and elicitation database. Each source is a dynamic content repository and therefore correspond a portlet in a second generation portal. The generation process is modeled after the MDA architecture of Figure 1. The result is a template-based transformation pattern in which two elements become the driver of the process. First is the domain knowledge which is dynamically configured from sources resulting in the artifact that is transformed using templates into requirements. Templates are product constraints from which commonality and variations are constructed for the system family architecture and individual family members respectively. When a requirement document is requested, it is rendered as portal page presentation with links to both commonality and variability in the product line. Each time a hyperlink is clicked, a portlet is activated to generate that particular segment of the document. Clicking on the root node of the feature diagram generates a full document.

TABLE 2
PROCESSES IN THE PL REQUIREMENTS GENERATOR FRAMEWORK

Phase	ACTIVITY
Elicitation	1. User request for a domain feature 2. System provides default domain requirements 3. User customizes requirements by entering own perspectives. 4. Results in elicitation database
Identify Features using PuLSE-CaVE	1. Identify common features 2. Identify variation among features 3. Add to classified features to the Feature repository
Reengineer Requirements	1. Combine existing Domain Vocabulary and new identified features 2. Add new knowledge to Domain Knowledge repository.
Generate Requirements	1. Retrieve templates and activate portlets 2. Output Requirements Documentation

IV. WEB STORE EXAMPLE

To assess as to whether the RSPL framework is worth adapting to, we have applied it to a web store e-commerce scenario. The purpose was to check whether textual requirements can be produced from domain knowledge (assets) following the four phases.

We analyzed a local small and medium Enterprise (SME) that required selling its product world wide but did not have enough resources to do so. We therefore envisioned that such SMEs need a web presence as part of its requirements. The four stages of RSPL were followed in the following section in order to identify the SME requirements and to have an extensive domain understanding.

1) *Elicitation Phase*

Figure 4 provides the user with a choice of product domain. Once the user selects the domain, a requirement template feature is provided that reflects all commonalities within that domain. The user is then given the option of accepting the requirements, adding new requirements or customizing the requirements as wanted. When the customize button is pressed, the user is taken to figure 6 that represents the next phase.

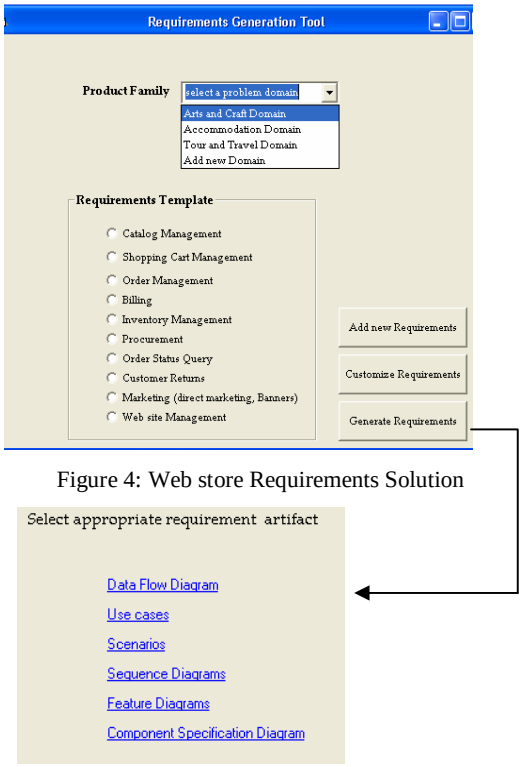

Figure 4: Web store Requirements Solution

Figure 5: Requirements artifacts

2) *Feature Identification*

The identified requirements were grouped into various features which are characterized as either common or variable depending on the domain knowledge. Identified features are: Web Site Management, Customer Relationship Management, eSales and Procurement (see figure 6). Each time a hyperlink is clicked, a portlet is activated to generate that particular segment of the document. Due to space limitation, we only illustrate the eSales Feature. When the eSales hyperlink Feature is clicked, the eSales Feature template is generated and can be modified or customized as one wishes. New features can be added using the *Add new Feature* button. Adding a new feature requires the addition of both common and variable elements (see figure 7) as well as constraints that the feature is restricted to.

Figure 6: Components of the Web Store (depicting eSales Features)

Figure 7: Adding a new feature

3) *Reengineering and Generating Requirements*

New features are added to the domain vocabulary and domain knowledge (see figure 8). For example, Report is a new vocabulary name that will have to be incorporated into the vocabulary repository. When a user selects the Generate Requirements in figure 4, he/she will have to choose which requirement artifacts is required (figure 5). Due to limitation of space, we are unable to show all generated templates but have shown a snapshot of the Scenario Requirement artifact in figure 9.

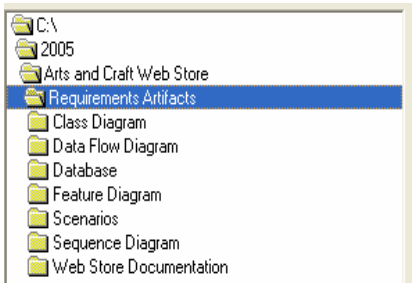

Figure 8: Domain Knowledge Repository

Figure 9 presents a customer-order scenario. Making an order incorporates the process of Billing, hence the need to

show the feature *Bill To*. Once the billing is done, the order has to be shipped to either the customer or another person whom the product is bought for. This creates the *ShipTo* feature that reflects the Address element with Name as an optional feature (Optional features are reflected using dotted rectangle in the address element). Each generated requirements is documented in the domain repository. A template is presented to the user in the form of a portal interface.

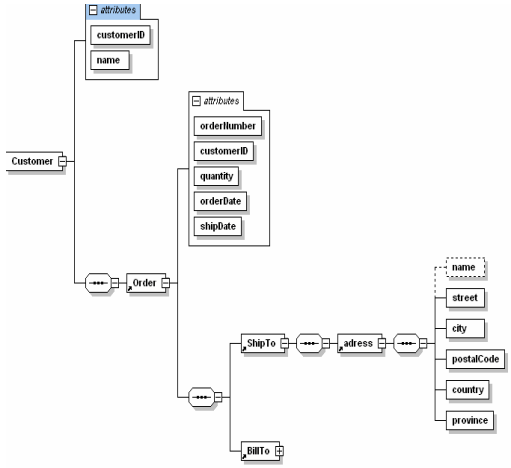

Figure 9: An example of generated Requirement Template

V. CONCLUSION AND FUTURE WORK

A conceptual framework for generating requirements dynamically via a portlet-type user interface has been described. The whole concept is still under development and is being actively used in a live development project at the moment. The application of the framework in the development of a family of web store system for a group of Arts and Crafts small and medium enterprises operating under a cooperative agreement has produced very interesting results.

Our future plan is to use the fully automated RSPL to maintain the requirements for building the individual members of the family according to the needs of cooperative members. This work adds value to the system development process in the telecommunication services, by taking advantage of reuse of common assets and also by providing automated rapid production of textual requirements that would have proven costly and time consuming otherwise.

REFERENCE

[1] W. Scot and S.C. Cook, "An Architecture for an Intelligent Requirements Elicitation and Assessment Assistant", INCOSE 2003-13th Annual International Symposium Proceedings.

[2] Greenfield and K. Short, "Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools", Wiley Publishing,Inc.

[3] K.Kang, S.Cohen, J.Hess, W.Novak and S.Peterson, "Feature Oriented Domain Analysis (FODA) Feasibility study", Technical report CMU/SEI-90-TR-21, Software Engineering Institute, November 1990

[4] S.A. Sidky and J. D. Arthur "RGML: A specification Language that Supports the Characterization of Requirements Generation Process. 28th Annual NASA Goddard Software Engineering Workshop (SEW'03).

[5] ARTiSAN Software Tools "Code Generation From UML For Small Systems", Retrieved from http://www.artisansw.com/news/codegen2.asp

[6] A.W. Brown, "An introduction to Model Driven Architecture" Retrieved from http://www-106.ibm.com/developerworks/rational/library/3100.html

[7] J. Dorr and I. John, "Elicitation of Requirements from User Documentation" Retrieved from http://www.empress-itea.org/publications/EMPRESS_22.pdf

[8] Gomaa H. Reusable software requirements and architectures for families of systems. Journal of Systems and Software, 25(3):189–202, August 1995.

[9] Prieto-Diaz R. Domain analysis: An introduction. ACM SIGSOFT Software Engineering Notes, 15(2):47–54, 1990.

[10] Weiss D.M and Chi Tau Robert Lai. Software Product Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[11] Ardis M A and Weiss D. M. Defining families: The commonality analysis. In Nineteenth International Conference on Software Engineering (ICSE'97), pages 649–650, 1997.

[12] Neighbors J. The draco approach to constructing software from reusable components. IEEE Transactions on Software Engineering, 10(5):564–574, 1984.

[13] Bentley J.L. Programming pearls: Little languages. Communications of the ACM, 29(8):711–721, August 1986.

[14] Coplien J, Hoffman D, and Weiss D, Bell Labs, Commonality and Variability in Software Engineering. November/ December 1998 IEEE Software

[15] J. Herrington, "Generation Techniques for Java" Retrieved from http://www.onjava.com/pub/a/onjava/2003/09/03/generation.html

[16] A.Cockburn, "Writing effective Use Cases" Addison Wesley, 2001

[17] B.Paech and K.Kohler, "Task-driven Requirements in Object-oriented Development". In Leite, J.,Doorn, J.,(eds) Perspectives on Requirements Engineering, Kluver Academic Publishers, 2003, to appear.

[18] D. Batory and S. O'Mally, "The design and implementation of hierarchical software systems with reusable components" ACM Transactions on Software Engineering and Methodology, 1(4):355–398, October 1992.

[19] D.L. Parnas, "On the Design and Development of Program Families", IEEE Transactions on Software Engineering, SE-2:1-9, 1976

[20] S. Faulk, "Product Line Requirement Specification (PRS): an Approach and Case Study", Proceedings of the 5th International Symposium of Requirements Engineering (RE'01).

[21] Parnas, D.L., "On the Design and Development of Program Families", IEEE Transactions on Software Engineering, SE-2:1-9,1976

Kabanda S.K holds a BCOM. Honors (Information Systems) and is currently doing her Masters of Science degree in the Department of Computer Science at the University of Zululand. Her research title is "Requirement Specification for a Product Line using 2nd Generation Portal Technology".