RESEARCH PAPERS
FACULTY OF MATERIALS SCIENCE AND TECHNOLOGY IN TRNAVA
SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

10.2478/rput-2013-0015 2013, Volume 21, Special Number

BUSINESS APPLICATIONS ARCHITECTURE MODEL BASED ON SOFTWARE PRODUCT LINE APPROACH

Zdravko ROŠKO

ABSTRACT

Software product line architecture is one of the most important artifacts defined at the early stage of a product line development process. Since the rest of the products are developed based on the initial product line architecture, it is of high importance to ensure the architecture stability by enabling the software's evolution possibilities. Industrial evidence shows that companies spend more resources on maintaining and evolving their architecture and products than on the initial development of them. Hence, there is a need for flexible software architecture that stays stable as the requirements evolve. In this paper we propose a structural model, some architecture quality metrics, case-based reasoning methodology to predict the architectural stability and a feature model for business applications. The goal of the proposed architecture model is to develop a framework for business applications development and evaluating the stability of product line architectures in the face of changes in requirements.

KEY WORDS

software product lines, feature model, architecture, case-based reasoning, metrics

INTRODUCTION TO THE PROBLEM

Software reuse is the process of creating software applications from existing artifacts rather than building them from the scratch. Effective reuse requires a strategic vision that reflects the unique power and requirements of this technique [1]. There are many software engineering technologies that involve some form of software reuse such as: application frameworks, design patterns, components, application generators, etc. Many organizations employ these technologies, and many are ready to take the next step towards more effective reuse of software.

Software product lines (SPL), in which; requirements, architecture, modeling and analysis, components, test cases, test data, test plans, documentation templates, and other software engineering artifacts, can be reused over a number of applications, is at the moment the most promising form of the software reuse [2]. SPL is defined as a set of software-intensive systems, sharing a common, managed set of features that satisfy the specific needs

Zdravko ROŠKO – Adriacom Software Inc., Stablinac 4/58, Vodice, Croatia, zrosko@gmail.com

of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way [3]. SPL development process consists of domain engineering process, (core assets development *for reuse*) and application engineering process (product development *with reuse*) that builds the final products, where construction of the reusable assets and their variability is separated from production of the product-line applications. SPL is mostly used by organizations that develop software for mobile phones, cars, electronic instruments, while information systems domain is not often considered as a potential base for developing SPL. Successful product lines have enabled organizations to capitalize on systematic reuse to achieve business goals and desired software benefits such as productivity gains, decreased development costs, improved time to market, higher reliability, and competitive advantage [4]. Considering the costs, as stated by [5] SPL offer benefits when producing at least a certain number of products. Figure 1 (partially taken from [5]) illustrates the costs and distinct stages of producing one versus multiple products from the same product line. The solid line sketches the costs of developing the products independently, while the dashed line sketches the costs of developing the products using software product line engineering approach [6]. The figure shows the case when less then four products are spawned from the same product line, where the price of product line engineering is relatively high, and the case whereas it is significantly lower for larger quantities of products being spawned from the sample product line [6]. There is a break-even point, we call it „SPL early stage end“ at which the two lines intersect. It indicates that the costs are the same for both cases. As referred in [5] recent empirical experiences have shown that this break-even point is located at around 3 or 4 systems in the particular case of software engineering.

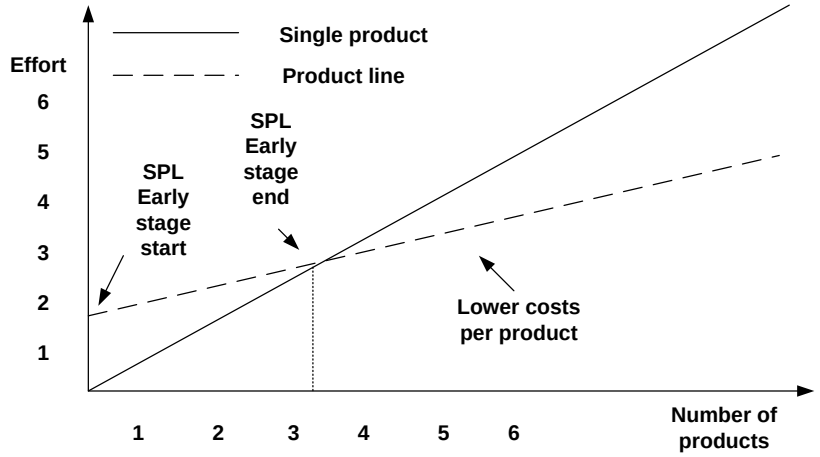

Fig. 1 *Costs of a SPL development*

Business applications are a kind of software that is used by business users to perform various business functions. Most of the business applications are interactive, they interact with a user through a user interface in order to read, process or change some persistent business data. The SPL for interactive applications defines, product line requirements, a software architecture and a set of reusable components. The existing frameworks such as Spring may sound like a solution for the problem, however it does not impose any specific programming model, it does not address all possible interfaces needed and it may lack a certain up to date features. Hence, making a product line architecture dependent on externally developed artifacts with not enough power to replace or change some of the key architecture features, is not a solution. One of the most important parts of a SPL is its architecture (PLA). The PLA plays a central role at the development of products from a SPL as it is the

abstraction of the products that can be generated, and represents similarities and variabilities of a product line [7]. The PLA must consider the needs of the complete set of products in order to provide a framework for the development and reuse of new assets. These new assets have to be conceived with the required flexibility in order to satisfy the needs of the different products in the SPL [6]. PLA consist of *frameworks* (Szyperski., 2002) as core assets, whose design captures recurring structures, connectors, and control flow in an application domain, along with the points of variation explicitly allowed among these entities [7]. In this paper we use the term „SPL platform framework“ to represent the implementation of the generic architecture and components which are not business-specific but rather generic in the sense that they can be used by more than one business domain such as: banking, insurance, manufacturing, and etc. We propose a business application architecture model which includes:

- Business applications entities structural model
- Feature model for business applications
- Some „SPL Platform Framework Responsibility“ metrics for SPL stability
- Case-based reasoning methodology used to predict the architectural stability

BUSINESS APPLICATIONS ENTITIES STRUCTURAL MODEL

Today's interactive business applications consist of the three logical layers which have a distinct and specific responsibility: presentation, business logic and data access logic. Presentation layer's function is an interaction with the application's users which includes: various rendering of the data, data edits, data validation and formatting, data inter-dependency checks, and other user initiated actions. Business logic layer function is to process data entered by a user and/or data retrieved from the persistence data source. Business logic should stay free from dependencies on various data sources and let the variability mechanism of SPL to choose among different data sources. Data access logic layer function is to handle all interactions with the persistent data sources. The layered model does not imply that each layer should be in a separate address space, even thought in today’s business application‘s environment the most of the time a three-tier model is used. Control and data can flow in both directions in layered systems. However, lower layers must not depend on functions provided by higher layers. Such a design avoids accidental structural complexity, and supports the use of lower layers in other applications independently of the higher layers [8]. Table 1 shows that business domain specific components shared among different products spawned from the same product line are not a part of the SPL platform framework, but rather are part of the business-specific components but still belong to the domain engineering process.

PROPOSED PLA STRUCTURE Table 1

Prod 1	Prod 2	Prod 3	Prod 4
Business-specific components			
SPL Platform Framework (common services)			
External Components			
OS/Language Environment			

The structural model is the framework through which components, attributes, and inter-relationships within the system are expressed [9]. The structural model enforces a consistency in the business applications structure by a set of constraints (e.g., the way a data is passed between layers, organization of the source code, the relationship between the source code

pieces). The Figure 2 shows the structural model for business applications we propose. The proposed model specifies: the kind of entities that will exist in the design (how do we package the entities), how the real world product (application) is mapped to the software entities (what is in a package) and the dependencies between the entities (how do packages relate to each another). Given a fact that most of the business applications are composed from a client part, which may be run in a separate address space, and a server part which may be run within an application server on the other address space, we assume that some of the software assets are shared between the two.

Client resources include the entities which are used by client part of an application while server resources include the entities used by server part of an application. Shared resources are the entities which are shared by client and server parts of an application. This structure does not impose a separation of client and server to the two separate address spaces, but indeed represent a variation point which can be used to compose an application as a one part to be run in one address space or as a two separate parts to be run in two distinct address spaces. The Figure 2 shows 13 distinct dependency relationships among different SPL structural entities. As we will show later some of them will be used as elements of the new proposed metric for stability of SPL platform framework.

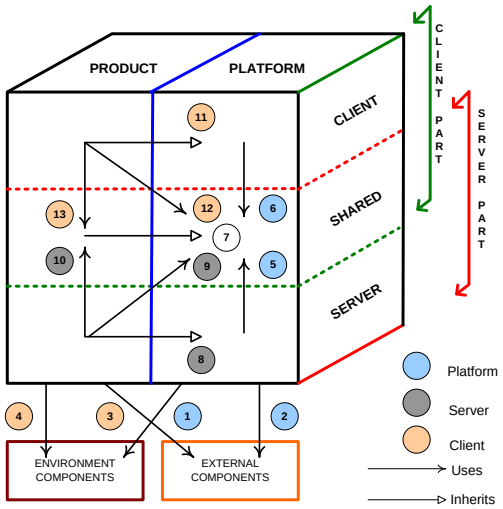

Fig. 2 *Proposed structural model and dependencies*

FEATURE MODEL FOR BUSINESS APPLICATIONS

Features are important distinguishing aspects, qualities, or characteristics of a family of systems [10]. Features are use to depict the shared structure and behaviour of a set of similar products. Feature model for business applications is used for representing the possible configuration space of all the products of a product line in terms of its features. Business applications feature model is composed from the client and server models. Feature model for client (Figure 3) captures variability and commonality between the features of the different products available in a given domain.

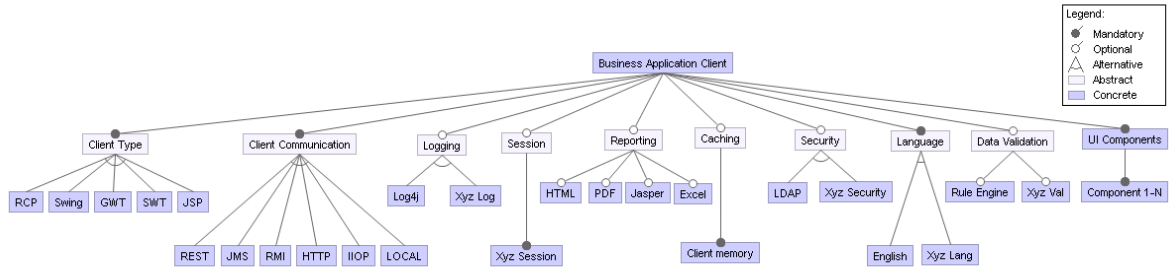

Fig. 3 *Client feature model*

Figure 4 shows server feature model. Not all possible configurations of the server features produce a valid server part of an application. For instance, a configuration of server part of an application that uses EJB as a type of business objects cannot use a non EJB transaction feature. Such restrictions are expressed in the form of integrity constraints. An example of these constraints is: *Business Object EJB EXCLUDES XYZ Transaction*. These constraints ensure the correct composition of product features in the various final business applications developed from this feature model.

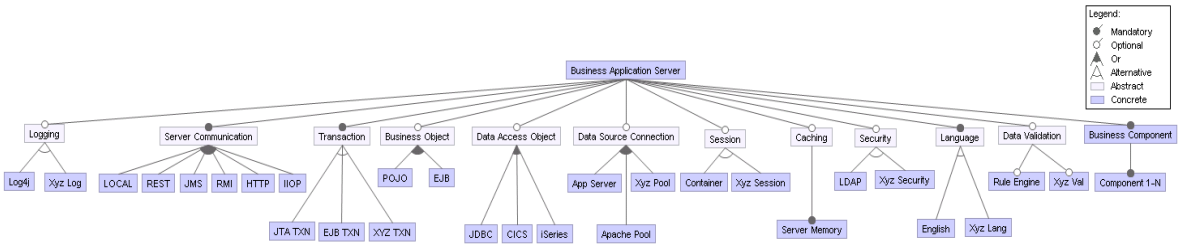

Fig. 4 *Server feature model*

PLATFORM FRAMEWORK RESPONSIBILITY METRICS

Software metrics to measure quality attributes of an architecture such as “Design Quality” metrics [11], metrics to measure structural soundness of product line architecture [12], PLA metrics [13], and complexity metrics for software product line architectures [7] do not address the quality of SPL platform framework responsibility. Within the context of SPL for business applications which is based on generic components, early indicators of the software product line architecture (PLA) quality attributes can be used in order to avoid low-quality products during the later stages of product development [14]. We propose a „SPL Platform Framework Responsibility“ metrics which can be used as an early indicator of the future product's quality. A platform framework, is a group of components and services that provide a coherent set of functionalities through inheritance, interfaces and specific design patterns. The application development process should be concerned with the business requirements rather than with the low level APIs or external component's interaction rules. Platform framework needs to ensure the application development process independence by taking the responsibility to interact with external third-party components. By external components we refer to a non-development components developed by a third party organizations and used by the SPL platform framework or by a products spawned from it, illustrated in Figure 5. Referencing an external component directly from a business application product, makes the product less stable and harder to develop or change. The more external components a product relies on, the larger the likelihood to misunderstand or misuse

some of these services. Therefore, the product is more difficult to understand and develop, and thus likely to be more fault-prone. The product line platform framework should take as much as possible of the responsibility to interact with external components. We propose a five simple and intuitive architectural metrics as a measurement for SPL platform framework quality based on architectural elements dependency [14].

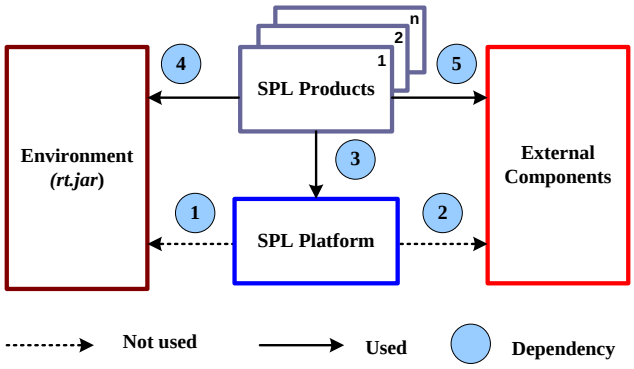

Fig. 5 *SPL Platform Framework Metrics*

As illustrated in Figure 5 there are 5 distinct high level dependency metrics of an SPL for business applications. SPL platform depends on its environment such as Java or .NET and on a number of external third-party components, while SPL products depend on its platform framework its environment and on a third-party external components. The proposed „SPL Platform Responsibility“ metric use the three dependencies metrics (Figure 5): D3: „Platform Afferent Coupling“ - the number of distinct references outside the platform that depend upon classes within the platform, D4: „Product Efferent Coupling“ - the number of distinct references inside the product that depend upon classes within environment components (e.g. Java RTE), D5: „Product Efferent Coupling“ - the number of distinct references inside the product that depend upon classes within external components. We can calculate the *Platform Responsibility* (PR) for a product line platform framework through the following equation:

$$PR = \sum_{i=1}^{n} \frac{D5_i + D4_i}{D3_i + D5_i + D4_i}.$$

The *PR* can be calculated for each product or for all of products spawned from the product line. *PR* = (D4+D5) / (D3+D4+D5): The range for this metric is from 0 to 1, where PR=0 indicates that SPL platform used by product makes the product more stable and protected from frequent changes to the external third party components, while the SPL platform serves the product by taking the responsibility to interact with external components. PR=1 indicates a completely irresponsible SPL platform.Table 4 shows the calculation of the *PR* for three products (P1, P2, P3).

MULTIPLE PRODUCT PR CALCULATION

Table 2

	D3	D4	D5	PR
P1	4	3	3	0,60
P2	4	3	0	0,43
P3	4	0	0	0,00
Total	12	6	3	0,43

The proposed metrics may be analyzed within the framework of measurement theory such as the Distance framework [15] and framework based on desirable properties which serves guidance provided to define proper measures for specific problem [16].

CASE-BASED REASONING USED TO PREDICT THE STABILITY

Predicting product design stability of software product lines for business applications, i.e., the ease with which a product evolves while it's design remains stable, can be used in order to plan product maintenance activities during the later stages of product's existence. A well designed product spawned from a software product line inherits most of the characteristics from the SPL platform framework but it also shares many similarities between other products. Product stability is a complex measure and its prediction is of high importance for any software maintenance planning. We propose an approach that uses the case-based reasoning (CBR) and k-nearest neighbour (k-NN) technique to predict the product stability. The application engineering process that uses and apply the stability prediction will help ensure that final product's maintenance is planned by using the most closest and similar cases from the historical case-library. Since there is a lack of knowledge about software evolution, we believe that CBR is an appropriate approach to the business application stability prediction problem. We hypothesize that two products (business applications) which show same or similar characteristics will also evolve in a similar way. Case repository for applications and its versions needs to have an appropriate structure which will enable the stability prediction. We propose to use the dependencies metrics explained earlier and a set of structural software metrics. Each metrics may be assigned a weight calculated by assigning the importance factor to each metric.

CONCLUSION

In this paper we propose some parts of an architecture model for software product lines in the field of information systems. We propose an entities structural model, feature model for business applications, a new metrics for measuring the „responsibility“ of a common platform framework and a case-based reasoning approach for predicting the stability of an architecture. Our future research is directed at the design of a complete architecture model based on a case study to help reduce the effort to maintain business applications.

REFERENCES

1. The Institute of Electrical and Electronics Engineers, „Guide to the Software Engineering Body of Knowledge,“ 2004. p. 120.
2. Z. ROŠKO. 2012. Strategy Pattern as a Variability Enabling Mechanism in Product Line Architecture.
3. L. N. Paul Clements, Software Product Lines: Practices and Patterns, 3 ed., Westford, MA: Addison-Wesley Professional, 2001, p. 608.
4. V. S. S. P. Kyo C. 2011. Kang. Applied Software Product Line Engeenering. *Taylor and Francis Group,* p.
5. G. B. a. F. J. v. d. L. Klaus Pohl. 2005. „Software Product Line Engineering:Foundations, Principles and Techniques,“ *Springer-Verlag,* pp. 12, 14, 15, 46, 62, 156, 157.

6. C. PARRA. 2011. Towards Dynamic Software Product Lines: Unifying Design and Runtime Adaptations.
7. M. G. J. C. M. Edson A Oliveira Junior. 2001. Empirical Validation of Variability-based Complexity Metrics for Software Product Line Architecture.
8. K. H. D. c. S. Frank Buschmann. 2007. Pattern-Oriented Software Architecture, zv. 4, p. 187.
9. J. Robert G. Crispen and Lynn D. Stuckey. 1994. STRUCTURAL MODEL: Architecture for Software Designers.
10. K. K. K. &. L. J. Lee. 2002. Concepts and guidelines of feature modeling for product line software engineering. *Lecture Notes in Computer Science*, pp. 62,77.
11. R. MARTIN. 1994. OO design quality metrics. An analysis of dependencies.
12. Rahman. 2004. Metrics for the Structural Assessment of Product Line Architecture.
13. N. M. a. A. v. d. H. Ebru Dincel. 2002. Measuring Product Line Architectures. *Software Product-Family Engineering*, pp. 151-170.
14. Z. ROŠKO. 2013. Assessing the Responsibility of Software Product Line Platform Framework for Business Applications. *CECIIS*.
15. G. a. G. D. ". a. f. f. s. m. c. D. R. R. 9. (. 1.-4. Poels, DISTANCE: a framework for software measure construction. *DTEW Research Report 9937*, pp. 1-47, 1999.
16. L. C. S. M. a. V. R. B. Briand. 1996. Property-based software engineering measurement. *Software Engineering, IEEE Transactions on 22.1*, pp. 68-86.
17. „An Industrial Case Study of Product Family Development Using a Component Framework“.
18. K. S. E. R. Frank van der Linden. 2010. Software product lines in action, the best industrial practice in product line engineering. *Springer*, p. 8.
19. D. G. Ebrahim Bagheri. 2011. Assessing the maintainability of software product line feature models using structural metrics.