RunTime Variability in Domain Engineering for Post-Deployment of User-Centric Software Functional Completion

Alexandre Manuel Tavares Bragança

Research fields: domain engineering, domain-specific languages, feature modeling, component composition, and runtime execution environments.

Introduction

Product families and product lines aim at promote reusability within a given set of software products [Bosch 2000]. Software product lines have achieved substantial adoption by the software industry. The adoption of Product line software development approaches has enabled a wide variety of companies to substantially decrease the cost of software development, maintenance, and time to market and increased the quality of their software products [Bosch 2002].

To accomplish reusability among various software products there must be common characteristics among them. Normally this means that the various software products must share the same domain. Therefore, an organization that has built several software systems in a domain also has acquired very good knowledge of such a domain. This knowledge can be used when building new software systems in the same domain. A fundamental technical requirement for achieving successful software reuse is the systematic discovery and exploitation of commonality across related software systems [Prieto-Diaz 1990].

So we can say that reuse has to do with finding commonalities among software systems within a domain. Nonetheless, to build diverse software systems within a domain we also need to specify variability. Domain engineering focuses on supporting systematic and large-scale reuse by capturing both the commonalities and the variability of systems within a domain to improve the efficiency of development and maintenance of those systems. As such variability is one of the key aspects of domain engineering.

Variability in software is achieved fundamentally by pre-runtime techniques [Svahnberg and Bosch 2000; Gurp 2003].

There are also more recent techniques and methods that came from the academia but have limited adoption in the industry, such as: aspect oriented programming [Kiczales *et al.* 1997], subject oriented programming [Ossher *et al.* 1994] or generative programming [Czarnecki 1998]. These techniques tend all to resolve the variability issues at compilation-time. Thus variability is solved mostly at pre-deployment and deployment (installation) time.

Research Problem

There are some cases in witch solving variability at compilation-time and even at deployment time is not a satisfactory solution.

Let's take, for instance, the case of a system aimed at the insurance industry. One common situation in this domain is the need that the insurance company has to market new insurance products. Products in the insurance market are very complex. A new insurance product may imply that new and variable data may be necessary to register for each new insurance policy of that product. New rules for risk assessment and claim processing may also be needed. These are some of the many variability points that such a system may need to cope in the case of the market of a new insurance product. We can say

that such a software system may have significant change in behavior for each insurance product. Insurance companies also need to market new products very quickly in order to respond to market needs.

In the described scenario if we have, for instance, 10 insurance companies using the software system and an average of 20 insurance products for company in the case of software variability solved at compilation-time we must have 10*20=200 variations of the software system.

Even if technically one could manage such a number of diverse versions of the system one significant problem remains: when an insurance company needs to market a new product it is necessary for the software engineer (in the software house) to build the new variation point in the system (i.e., a new application). This will encompass the engineering process that takes time, and also time to deploy the new application. Such a time frame may not be feasible because of the constraints of the time to market for the new product.

In the study we made so far, it is possible to identify and resume the limitations regarding variability in domain engineering in the following three problems:

(1) Variability is mostly taken into account before or during application deployment. As observed, this is not always the best solution.

(2) At the same time, and also resulting from the previous problem, the domain engineering methodologies do not take into account the possibility of the application customer having a special role in a pos-deployment phase. This special role could enable the variable points of the application to be resolved at runtime.

(3) Finally, the techniques used to achieve variability do not take into account the technology heterogeneity of most real cases. For instance, in most techniques it is implicit the adoption of object-oriented languages.

In this thesis we will face the dynamic runtime software variability problem, in particular the three major problems described above. For such we propose the adoption of a domain-specific language and framework.

Thesis statement

The adoption of domain-specific languages in domain software engineering can significantly improve product line variability and flexibility by (a) enabling an effective technique for implementing runtime variability; (b) enabling the possibility of a pos-deployment kind of software engineering; and (c) enabling variability in a technically heterogeneous product line.

Proposed Approach

Regarding the first problem, we will evaluate to what extent it is feasible to specify the variability of an application at runtime using the proposed technique. In order to do so a domain-specific language and framework will be evaluated in a real case of an insurance software product line. A pragmatic approach will be used because this evaluation will be done in a real case. Feedback from end-users, software engineers and domain experts will be registered and results from this real case will provide further insight into the viability of our propose regarding runtime software variability. These results can then be compared with documented results from other techniques of software variability.

Our approach raises the second problem mentioned above. Who will solve the runtime variability? The end-user? The end-user in a special role of *a kind of* software engineer? Our approach should try to answer these questions. In order to do so it

is necessary to revise the software engineering methods, particularly the domain engineering methods, so that dynamic variability and its method's implications can be "formally" integrated in a methodology. Regarding this problem, we propose the adoption of existing and proved methodologies with the necessary adaptations.

In our approach to the runtime variability issue, we see two (almost) orthogonal aspects:

(1) how to combine features at runtime, i.e., how do end-users select features at runtime;

(2) how to enable behavior specification holes in the applications for the end-users to fill at runtime.

Static runtime variability. Our research so far indicates that the first aspect is similar to feature composition techniques used at pre-runtime, like GenVoca [Batory *et al.* 2002]. At this initial exploring phase of our work, we propose that the operating runtime environment should be extended with: (a) metadata; (b) reflection; and (c) runtime compilation. We assume that for the first aspect of our approach to runtime variability the end-user will only do feature selection and composition and so there isn't a crucial need for a high-level feature composition language. At the present, it seems satisfactory to have a composition language similar to the one used with GenVoca.

Dynamic runtime variability. Regarding the second aspect of our approach to the runtime variability issue the context in witch feature specification is done suffers a very significant change. In fact, what we are advocating in this aspect of runtime variability is that domain engineers and also application engineers will leave holes in their specification for the end-user to fill. This is a scenario where one or more features don't have possible implementations specified before runtime. This only occurs in very dynamic domains like the example of the insurance domain. n this scenario end-users will have to play a special role because they will have to fill the holes left in application engineering. This filling of holes corresponds to implementing features. In order to achieve this, the end-user will need to do some kind of program specification. This entails several implications: the end-user needs to assume a special role; development methodology has to be adapted to support such need; feature implementation/specification language has to be adapted to this special end-user.

Work Plan. Our plan clearly adopts an applied research methodology. In fact, we will be integrated in a research department team of a company specialized in software solutions for the insurance industry. As such, we will interact with and work within the context of this team. Such a research method has its risks but also as many advantages. The advantages start with the possibility to have a realistic case study and experiment and validation context for our research. One major risk is the eventually impossibility to full control the development of the research actions. This case will enable the possibility of a real environment for the verification and validation of our work.

After our first year work documented in a report [Bragança 2003] we plan to carry out the future two years work with the following four major phases or activities: a) static runtime variability (generalization); b) static runtime variability (domain case study); c) dynamic runtime variability (domain case study); and d) dynamic runtime variability (generalization).

In the first phase of our work plan, we will focus our research on static runtime variability. This phase will be developed on the first half of 2004. In the first four months of second half of 2004, we plan to use the results from the previous phase in the case study. In this phase, we plan on focus our research on validating the company proprietary runtime operational environment for proper support of static runtime variability. After validating static runtime variability on the case study we plan to continue our work on the case study, this time with the focus on dynamic runtime variability. This phase of our work will have eight months duration, from November 2004 until June 2005. In this phase we will explore our propose for implementing dynamic runtime variability in applications. As already exposed, our propose is based on the adoption of

domain-specific languages. We will start our work on dynamic runtime variability in the case study. The enterprise case study we will be working on already uses a domain-specific language and runtime environment in its product line. We plan to validate our work on this phase using applications from the enterprise product line and (if possible) end-users and the enterprise software engineers. During the last half of 2005, in the last phase of our work, we plan on generalizing the results of our research. Particularly we plan on generalize the knowledge resulting from the previous phase using general available technology. The work done on the previous phase is based on proprietary technology and thus this phase will adapt it to a generic runtime framework. We also plan on sharing the experiences and knowledge resulting from our work with the scientific community in the field of study. This will be done mainly through paper publications in relevant scientific conferences of the field. We will plan on doing so in each phase of our work plan. One such example is the paper submitted for the ICSR'04 conference [Bragança *et al.* 2004].

References

[Batory *et al.* 2002] D. Batory, Roberto E. Lope-Herrejon, Jean-Philippe Martin, *Generating Product-Lines of Product-Families*, Proceedings of ASE'02, pp. 81-92, 2002.

[Bosch 2000] Jan Bosch, *Design and Use of Software Architectures Adopting and Evolving a Product-Line Approach*, Addison-Wesley, 2000.

[Bosch 2002] Jan Bosh, *Maturity and Evolution in Software Product Lines: Approaches, Artifacts and Organization*, Proceedings of the Second Software Product Line Conference (SPLC2), Lecture Notes in Computer Science, vol. 2379, pp. 257-271, Springer-Verlag, 2002.

[Bragança 2003] Alexandre Bragança, *Runtime Variability in Domain Engineering for Post-Deployment of User-Centric Software Functional Completion*, 1st Year Ph.D. Report, Universidade do Minho, November 2003.

[Bragança *et al.* 2004] Alexandre Bragança and Ricardo Machado, *Runtime Variability Issues in Software Product Lines*, submitted to ICSR'04, 2004.

[Czarnecki 1998] Krzysztof Czarnecki, *Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models*, Ph.D. Thesis, Department of Computer Science and Automation, Technical University of Ilmenau, 1998.

[Gurp 2003] Jilles van Gurp, *On the Design & Preservation of Software Systems*, Ph.D. Thesis, Computer Science Department, University of Groningen, 2003.

[Kiczales *et al.* 1997] Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Videira Lopes, Jean-Marc Loingtier and John Irwin, *Aspect oriented programming*, Proceedings of the European Conference on Object-Oriented Programming (ECOOP), Finland, Lecture Notes in Computer Science, vol. 1241, Springer-Verlag, June 1997.

[Ossher *et al.* 1994] Harold Ossher, William Harrison, Frank Budinsky, Ian Simmonds, *Subject-Oriented Programming: Supporting Decentralized Development of Objects*, Proceedings of the 7th IBM Conference on Object-Oriented Technology, July 1994.

[Prieto-Diaz 1990] Ruben Prieto-Diaz, *Domain Analysis: An Introduction*, ACM SIGSOFT Software Engineering Notes, vol. 15, no. 2, pp. 47-54, April, 1990.

[Svahnberg and Bosch 2000] Mikael Svahnberg and Jan Bosch, *Issues Concerning Variability in Software Product Line*, Proceedings of the Third International Workshop on Software Architectures for Product Families, Springer Verlag, 2000.