Fraunhofer Institut Experimentelles Software Engineering

Proceedings of the

Software Product Lines Doctoral Symposium

Baltimore, MD, USA - August 22, 2006

In conjunction with the
10th Software Product Lines International Conference - SPLC

Editors
Isabel John
Fraunhofer IESE, Germany
Len Bass
Software Engineering Institute (SEI), USA
Giuseppe Lami
ISTI - Italian National Council of Researches

IESE-Report No. 104.06/E
Version 1.0
August 14, 2006

A Publication by Fraunhofer IESE

Fraunhofer IESE is an institute of the Fraunhofer Gesellschaft.
The institute transfers innovative software development techniques, methods and tools into industrial practice, assists companies in building software competencies customized to their needs, and helps them to establish a competitive market position.

Fraunhofer IESE is directed by
Prof. Dr. Dieter Rombach (Executive Director)
Prof. Dr. Peter Liggesmeyer (Director)
Fraunhofer-Platz 1
67663 Kaiserslautern

Introduction

Different from other software engineering techniques, product-line engineering arose from practical experience in industry. But as with every successful technique product line engineering can also be considered only a real standard approach when it is not only applied in practice but also widely researched and especially taught in academia. Practice and academia are the two sides of the same coin. While industry sets the requirements, academia prepares the practitioners of tomorrow. The SPLC doctoral symposium provides a platform for young researchers to present their work to an international audience and discuss it with each other and with experts in the field.

The SPLC doctoral symposium originates from the successful experience of the past two editions of the SPLYR (Software Product Lines Young Researchers) workshop held in conjunction with SPLC '04 and SPLC '05.

Experienced researchers will comment on the presented work and give feedback for further development, research goals, methods, and results to provide useful guidance in completion of the dissertation research.

This event is a unique opportunity for the presenting young researchers and doctoral students to receive invaluable expert feedback, make contact with other researchers in the field, professionally present their work, and become familiar with other approaches and future research topics.

The doctoral symposium addresses research activities in the field of software product lines (SPLs). The peculiarity of this doctoral symposium is that it is addressed specifically to young researchers with original ideas and initiatives in the SPL field.

Although it mainly addresses PhD work in progress, we also encouraged the submission of other work in progress such as master's degree or diploma theses.

Different from the standard procedure of other doctoral symposiums and workshops, we have no blind peer reviews. Each student was assigned to one or two product line experts who reviewed the proposal and discussed pros and cons of the work with the student. We would like to thank our reviewers and panelists

- Birgit Geppert - Avaya Labs, USA
- Andre van der Hoek - University of California, USA
- Kyo Kang - POSTECH, Korea
- David Weiss - Avaya Labs, USA

for reviewing the proposals and for the effort they spent. Submissions were evaluated according to the relevance, originality, and feasibility of the work.

Having no blind reviewers provides a unique opportunity for the participating young researchers to get in contact with their reviewers and to receive valuable input for their work and presentation even before the actual symposium day.

We believe that with this symposium each of the participants will get valuable feedback for the further development of their work.

Keywords : Software Product Lines, Software Product Line Young Researchers Workshop, Proceedings, SPLYR.

Organization

The doctoral symposium is associated with the 10th International Software Product Line Conference (SPLC 2006), 21-24 August 2006, Baltimore, Maryland, USA.

Workshop Chairs :

- Isabel John
  Fraunhofer IESE
  Fraunhofer-Platz 1, D-67663 Kaiserslautern
  john@iese.fraunhofer.de
- Len Bass
  Software Engineering Institute (SEI)
  Carnegie Mellon University,
  Pittsburgh, USA
  ljb@sei.cmu.edu
- Giuseppe Lami
  Istituto di Scienza e Tecnologie dell'Informazione "Alessandro Faedo"
  Area della Ricerca CNR di Pisa, Via G. Moruzzi 1
  Giuseppe.Lami@isti.cnr.it

Reviewers / Panelists:

- Birgit Geppert - Avaya Labs, USA
- Andre van der Hoek - University of California, USA
- Kyo Kang - POSTECH, Korea
- David Weiss - Avaya Labs, USA

Workshop website and email:

http://www1.isti.cnr.it/SPL-DS-2006

SPL-DS@isti.cnr.it

Table Of Contents

Symposium Program

Tuesday, 22 August 2006, 9:00 – 15:15

9:00 Introduction

9:15-10:00 Rick Rabiser

Facilitating the involvement of Non-Technicians in Product Configuration

10:00-10:30 Coffee

10:30-11:15 Timo Asikainen

Methods for Modelling the Variability in Software Product Families

11:15-12:00 Nan Frederik Mungard

Feature Model Based Product Derivation in Software Product Lines

12:00-13:00 Lunch

13:00-13:45 Marcilio Mendonca, Toacy Oliveira, Donald Cowan

Collaborative *and Coordinated Product Configuration*

13:45-14:30 Karen Cortes Verdin, Cuauhtemoc Lemus Olalde

Aspect Oriented Product Line Architecture (AOPLA)

14:30-15:15 Uirá Kulesza, Carlos José Pereira de Lucena

An Approach to Framework Implementation and Instantiation using Aspect-Oriented Programming

For each paper, 20 minutes talk and 25 minutes discussion are planned.

1 Rick Rabiser
Facilitating the involvement of Non-Technicians in Product Configuration

Facilitating the Involvement of Non-Technicians in Product Configuration

Rick Rabiser[1]

Christian Doppler Laboratory for Automated Software Engineering
Johannes Kepler University, 4040 Linz, Austria

rabiser@ase.jku.at

Abstract. A key objective of product line engineering is to accelerate the configuration of products to different customer needs. Deriving products from a software product line remains challenging, even for experienced software engineers. However, the configuration process typically also involves non-technicians such as sales people with only little understanding of the underlying technical software solution. The complexity of today's software systems makes it difficult and error-prone for non-technicians to participate in this task. In the ongoing PhD research, carried out in cooperation with our industry partner Siemens VAI, we are developing and validating a tool-supported approach facilitating the involvement of non-technicians in product configuration.

1 Introduction and Motivation

Product line engineering (PLE) aims at reducing cost and time-to-market by increasing productivity through leveraging reuse of artifacts and processes in particular domains [20]. Software product lines consist of core assets such as features, architectural elements, and solution components. Central to PLE is the explicit modeling and management of commonalities and variability [15].

When deriving a certain product the core assets need to be properly customized and configured. This process involves technical staff as well as sales staff. However, the integration of product configuration by non-technicians and technical product configuration by developers is still weak. Also, there is a lack of integrated tools combining sales knowledge and product knowledge.

Several research areas have already addressed the issue of making technical product knowledge amenable to non-technicians. For example, recommender systems [1] have been successfully applied in e-commerce [22]. Such systems demonstrate how complex products and services can be presented to non-technicians in an intuitive

[1] PhD Student, 1st year

manner. By taking decisions (e.g., about desired features) customers can customize a product to their needs.

The ongoing research is carried out in cooperation with Siemens VAI, the world's leading engineering and plant-building company for the iron, steel, and aluminum industries. The domain of interest is Siemens VAI's automation software capabilities for continuous casting in steel plants. The focus of this PhD research is to develop an approach that enables the involvement of non-technicians (i.e., sales people) in configuring products. As quite typical in industry product configuration in sales processes is only weakly integrated with the actual technical product configuration carried out by developers. This increases the effort for developers who currently derive products from the product line in a rather manual way.

Current research in PLE emphasizes support for engineers [3,20]. The ongoing PhD research focuses on supporting sales people in their interaction with customers while at the same time lowering the configuration workload of developers through automation. The aim is to develop an automated approach generating a valid product configuration based on high-level decisions of sales people and customers. The vision is that non-technicians can already do a significant part of the configuration while developers are relieved from error-prone and burdensome configuration tasks. The approach integrates concepts from PLE, recommender systems, and product configuration [21].

The remainder of this paper is structured as follows: Section 2 presents related work. Section 3 outlines the domain of interest and describes challenges in the current sales and product configuration process of our industry partner. Section 4 specifies research issues and objectives. Section 5 explains the proposed approach. Section 6 describes the PhD research method. Section 7 concludes the paper by describing the current status of work and future research to be done.

2 Related Work

A significant amount of research has been carried out in software product line engineering. A good overview is given in [20]. For example, Deelstra *et al.* [9] have developed a framework of terminology and concepts regarding product configuration. The authors present a case study carried out in two large industrial organizations. Their findings confirm that in most cases technicians are burdened with doing the configuration work because of the deep technical knowledge necessary. Halmans *et al.* [13] emphasize the need of communicating the variability of a software product line to customers and also show how to represent this information. Of particular interest for the ongoing research is also the work of Czarnecki *et al.* [8]. The authors discuss a tool-supported approach for feature modeling and feature-based configuration. One of their aims is to further improve the user interaction model and the usability for non-technicians [7]. In [16], Kang *et al.* have introduced a product line asset development method that focuses on using marketing and product plans as a key design driver. Such plans describe what features belong to certain products and how the features will be delivered to customers now and in the future. Therefore they

directly influence product configuration and further involve non-technicians in this process.

Other research areas have also addressed product derivation. For example, the field of product configuration in the artificial intelligence community aims at automating the configuration of technical products in short time and with few errors [4]. Systems supporting this automation are called product configurators. They are based on product models and well-defined rules describing how to configure individual products and how to find valid configurations. Configuration systems have been successfully applied in a wide range of industrial environments [2]. Examples are reported from mechanical or electronic systems [17, 18]. However, product configurators for traditional (non-software) products often concentrate on the back-end technical aspects and neglect the non-technician perspective [4].

In the product configuration community approaches appeared for automatically building knowledge bases from already existing product models. For example, in [12] the authors describe an approach for generating a valid knowledge base for configuration from a UML product model. Although this approach is outside the scope of software product line engineering it is interesting and relevant for our ongoing research.

Recommender systems [1] have been successfully applied in e-commerce [22] and provide a good example of how complex products and services can be presented to non-technicians. Such systems enable customers to take decisions about desired features and allow customizing a product but typically do not directly support the technical product configuration. Certain types of recommender systems, namely advisor systems, have been successfully used for generating personalized, intelligent sales advisory applications. A commercially successful example is the Advisor Suite [14].

3 Case Study

The domain of this PhD research is Siemens VAI's automation software for continuous casting, in particular the level 2 automation (cl2). This software provides capabilities for process monitoring, material tracking, and process optimization (e.g., for the cooling process). It serves as a layer between the level 1 automation (machine-oriented automation) and the level 3 enterprise resource planning. Despite its size of more than 1.3 million LOC (mainly Java) it is highly configurable, extendable, and customizable to specific customer needs through a state-of-the-art component-based architecture. The architecture consists of about 80 different subsystems which can be connected via so-called adaptors. Adaptors are developed to pass information between defined subsystems; they are often called connectors in literature, e.g., [19]. The adaptor concept leads to a great degree of variability. As adaptors can also be connected to other adaptors the number of possible connections is very high. Siemens VAI ships 20+ cl2 software solutions per year customized to specific customer needs with a staff of only 35 software engineers. A thorough analysis of the current sales and product configuration process of Siemens VAI (Fig. 1) revealed several challenges [11]:

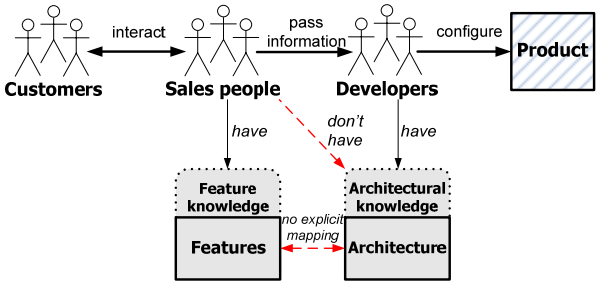

Fig. 1. Current sales and product configuration process

Knowledge is distributed. Knowledge about the customers' requirements is typically only available in the minds of sales people. Knowledge about the technical solution is only available in the minds of the developers. This makes it difficult to define and understand the complex dependencies among features and between features and architectural elements. Sales people without architectural knowledge are often unable to predict the consequences of their decisions which can result in the selection of an inconsistent set of features. Knowledge distribution is also an issue among developers. Due to the size of the cl2 software system it is impossible even for experienced developers to understand all subsystems at the same level of detail.

Weak communication links between sales and development. Sales people of our industry partner use comparably simple office tools such as spreadsheets and documents to communicate the features of the cl2 software product line to the customers or internally. However, as typical in industry, no explicit links are established between the architectural knowledge and the features of the system. It can happen that sales people do not get information about the latest features or feature modifications from the developers. There can also be delays in informing developers about customer requirements and feature requests.

Lack of tool support for product configuration. Sales people pass the information about selected features to the developers which then manually configure the product based on existing software components. Because of missing tool support for this configuration and because in many cases the sales people do not have the necessary knowledge about the technical solution errors can occur which have to be dealt with on level of the technical configuration.

4 Research Issues and Objectives

Based on a review of existing literature and the challenges reported by our industry partner we decided to address two research issues in our work:

(1) Weak integration of product configuration by non-technicians and technical product configuration by developers. The configuration of products by sales staff is mainly driven by customer needs and not based on the underlying technical solution. The technical configuration of products is mainly based on technical aspects with the risk of neglecting customer requirements.

(2) Lack of integrated tools combining sales knowledge and product knowledge. Existing tools for the configuration of complex industrial products mainly focus on product knowledge and neglect the sales perspective. Tools building on sales knowledge do not support the technical product configuration.

Our research objectives for addressing these issues are as follows:

(1) Development of interactive tools supporting non-technicians and involving them in the configuration process. Our vision is that non-technicians can largely configure a product with only little intervention necessary from the developers. Interactive wizards will be devised that support the sales process and allow sales people to provide the information required for automated configuration. The wizards will also support the sales people in their interaction with the customers during sales meetings by presenting relevant decisions to them.

(2) Integration of sales process with technical product configuration. To integrate the sales process with the technical product configuration the interactive wizards need to be integrated with product configuration tools. Thereby the generation of configurations based on information gathered with the help of the interactive wizards will be possible. This will help to relieve developers in the configuration work. In order to allow the integration all developed tools need to be based on the same knowledge.

5 Approach

In order to improve support for non-technicians and to automate product configuration we need to integrate concepts from product line engineering, recommender systems and product configuration [21]. The envisioned approach is depicted in Fig. 2.

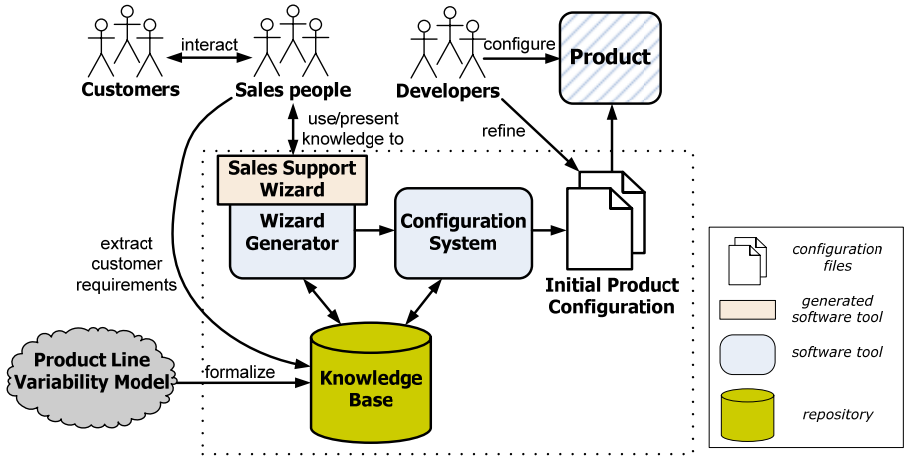

Fig. 2. Envisioned sales and product configuration process with support for non-technicians

The important knowledge (e.g., customer requirements, architectural knowledge, and feature knowledge) has to be documented explicitly in a knowledge base together

with rules of how to use and apply it to avoid knowledge gaps and redundant data entry. All tools will use this knowledge base to ensure their integration. The knowledge contained in a product line variability model – a model describing the commonalities and the variability of core assets together with the decisions that need to be taken to choose on certain variants – will be formalized and mapped onto a repository with rules describing how to use the data contained. A similar approach has already been proposed by Felfernig *et al.* [12] in a different domain. The integrated product line variability model is being built by another PhD researcher in our team [10]. Furthermore the knowledge base will contain sales knowledge such as customer properties and requirements.

Sales people will be supported in their interaction with customers through interactive, personalized wizards. A wizard generator tool will build these wizards for each sales process to address different customer properties. The wizard generator application uses the product line knowledge contained in the knowledge base. The generated sales support wizards consider the dependencies among features (e.g., the ordering of decisions) and the relationships between features and the underlying technical solution. The consequences of the decisions the sales people take will be explained by the wizards while simultaneously checking underlying technical constraints.

In a second stage information entered by sales people will be used by a configuration tool to automatically generate an initial product configuration, e.g., property files for linking certain subsystems into the product. The generated configuration can then be refined by the developers to create the final product.

6 Research Method

As this kind of research strongly relies on interaction with and feedback from non-technicians we will pursue an iterative approach informed by the spiral approach [5]. The implemented prototypes will be tested by sales people in real-world scenarios to ensure feasibility, usability, and scalability. Based on their feedback risks will be identified. A likely risk is the willingness of sales people to learn and use new tools. More specifically, our work plan includes the following tasks:

Literature and tool survey. A thorough literature review is being conducted in PLE and other relevant areas. Furthermore, already existing tools supporting similar goals are analyzed.

Analysis and modeling of existing sales process. An analysis of the sales process is necessary to understand the existing challenges. Existing documents (i.e., feature lists currently used by sales staff) are analyzed and discussed with process owners. Another plan is to participate in sales meetings to understand their dynamics. Taking into account the well-known effect that people tend to behave differently when they are under observation the results of such participation have to be handled with care. Either way, an analysis will help in validating the assumptions made in the preliminary analysis and allow explicitly defining and documenting the sales process as a prerequisite for developing proper tools.

Tool prototyping. Different tool prototypes will be devised supporting non-technicians in the sales and product configuration process. A candidate solution is a

wizard generator tool to automatically build customized, personalized wizards supporting the interaction process between sales staff and customers. These generated wizards will reflect the sales process in their user interface and present the complex product line knowledge to the customers. Furthermore, a product configuration tool will be developed to automatically generate product configurations based on the information gathered with the help of the wizards. As an alternative, we are currently exploring whether group support systems (i.e., GroupSystems[2]) can help in presenting product line knowledge to different stakeholders and gathering knowledge from them, especially from non-technicians.

Validation. It has to be assured that the approach and the developed tools are effective and efficient in real-world scenarios. It is however difficult to measure the effectiveness and efficiency of our anticipated approach. The effectiveness certainly relates to the time it takes to build the final configuration deployed to the customer. The efficiency could be compared to the efficiency of current sales processes; however, we are currently unaware whether the data needed for such analysis is available and accessible. Therefore, qualitative measurements gained in interviews with sales people are possible a more promising source.

7 Status of Work and Future Research

In this paper we presented ongoing PhD research with the goal of developing and validating an approach to facilitate the involvement of non-technicians such as sales people in product configuration. Our goal is to narrow the void between the product configuration in sales processes and the actual technical configuration of products in industrial contexts. Tools that combine sales knowledge and product knowledge can further help in bridging the gap between sales staff and technicians (i.e., developers), possibly resulting in a faster and less error-prone product configuration process.

In a first iteration a literature and tool survey and a preliminary analysis of the sales process have been conducted leading to the research issues and objectives described in Section 4. The literature and tool survey showed that the concepts used by the Advisor Suite [14] and other knowledge-based recommender systems [6] are promising in the context of the ongoing research, i.e., especially regarding the interaction process between sales staff and customers. Our analysis of the sales process, also described in [11], revealed the issues described in Section 3. The author will participate in sales meetings in the near future and analyze more feature lists to get a better understanding of the complex sales processes of our industry partner.

Based on the product line variability model devised by another PhD researcher [10] an initial knowledge base will be created together with a first prototype of a sales support wizard. Also, a first product configuration system prototype will be built that uses the knowledge base on the one hand and the information gathered by the wizard on the other hand to demonstrate the concepts of non-technician support for product configuration. The plan is to provide our industry partner with first prototypes in the next few months to receive feedback.

[2] www.groupsystems.com

Acknowledgements

I gratefully acknowledge the support of this research by Christian Doppler Forschungsgesellschaft in the Christian Doppler Laboratory for Automated Software Engineering. I also want to thank our industry partner Siemens VAI for support and feedback.

References

1. Adomavicius, G. and Tuzhilin, A.: Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Transactions on Knowledge and Data Engineering, 17(6):734-749 (2005).
2. Ardissono, L., Felfernig, A., Friedrich, G., Goy, A., Jannach, D., Petrone, G., Schäfer, R., Zanker, M.: A Framework for the Development of Personalized, Distributed Web-Based Configuration Systems. AI Magazine, 24(3): 93-108 (2003).
3. Birk, A., Heller, G., John, I., Schmid, K., von der Masen, T., Müller, K.: Product Line Engineering: The State of the Practice. IEEE Software, 20(6):52-60 (2003).
4. Blecker, T., Abdelkafi, N., Kreutler, G., Friedrich, G.: Product Configuration Systems: State of the Art, Conceptualizations, and Extensions. In: Hamadou, A.B., Gargouri, F., Jmail, M. (eds.): Génie logiciel & Intelligence artificielle. Eigth Maghrebian Conference on Software Engineering and Artificial Intelligence (MCSEAI), Proceedings. Centre de Publication Universitaire, Tunis (2004), pp. 25-36.
5. Boehm, B.: Spiral Development: Experience, Principles, and Refinements. Special Report, CMU/SEI-2000-SR-008 (2000).
6. Burke, R.: Knowledge-based Recommender Systems. In: Kent, A. (ed.): Encyclopedia of Library and Information Science. Marcel Dekker, New York, USA (2000).
7. Czarnecki, K. and Kim, C.H.P.: Cardinality-Based Feature Modeling and Constraints: A Progress Report. OOPSLA'05 Workshop on Software Factories, San Diego, California, USA (2005).
8. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration Using Feature Models. In: Nord, R. (ed.): 3rd International Conference on Software Product Lines (SPLC 2004), Proceedings. Springer, Boston, Massachusetts, USA (2004), pp. 266-283.
9. Deelstra, S., Sinnema, M., Bosch, J.: Product derivation in software product families: a case study. The Journal of Systems and Software, 74:173-194 (2003).
10. Dhungana, D.: Integrated Variability Modeling of Features and Architecture in Software Product Line Engineering. 21st IEEE/ACM International Conference on Automated Software Engineering (ASE'06) Doctoral Symposium, Tokyo, Japan (2006).
11. Dhungana, D., Rabiser, R., Grünbacher, P., Prähofer, H., Federspiel, C., Lehner, K.: Architectural Knowledge in Product Line Engineering: An Industrial Case Study. 32nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Cavtat/Dubrovnik, Croatia (2006).
12. Felfernig, A., Friedrich, G., Jannach, D.: Uml as domain specific language for the construction of knowledge-based configuration systems. International Journal of Software Engineering and Knowledge Engineering, 10(4):449-469 (2000).
13. Halmans, G. and Pohl, K.: Communicating the variability of a software-product family to customers. Informatik – Forschung und Entwicklung, 18(3/4):113-131, Springer, Berlin, Germany (2004).

14. Jannach, D.: Advisor Suite – A Knowledge-Based Sales Advisory System. 16th European Conference on Artificial Intelligence (ECAI 2004), Proceedings. IOS Press (2004), pp. 720-724.
15. Kang, K.C., Lee, J., Donohoe, P.: Feature-Oriented Product Line Engineering. IEEE Software, 9(4):58-65, IEEE CS (2002).
16. Kang, K.C., Donohoe, P., Koh, E., Lee, J., Lee, K.: Using a Marketing and Product Plan as a Key Driver for Product Line Asset Development. 2nd Software Product Line Conference (SPLC2), Proceedings, San Diego, USA. Springer Lecture Notes in Computer Science, 2379:366-382 (2002).
17. Krebs, T., Hotz, L., Günter, A.: Knowledge-based Configuration for Configuring Combined Hardware/Software Systems. 16th Workshop on Planen, Sheduling und Konfigurieren, Entwerfen (PuK 2002), Proceedings. Freiburg, Germany (2002).
18. Männistö, T., Soininen, T., Sulonen, R.: Modelling configurable products and software product families. International Joint Conference on Artificial Intelligence (IJCAI 2001), Proceedings. Seattle, Washington, USA (2001).
19. Medvidovic, N. and Taylor, R.: A Classification and Comparison Framework for Software Architecture Description Languages. IEEE Transactions on Software Engineering, 26(1):70-93 (2000).
20. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer, Berlin, Germany (2005).
21. Rabiser, R., Dhungana, D., Grünbacher, P.: Integrating Knowledge-Based Product Configuration and Product Line Engineering: An Industrial Example. 17th European Conference on Artificial Intelligence (ECAI 2006) Workshop on Configuration, Riva del Garda, Italy (2006).
22. Schafer, J.B., Konstan, J., Riedl, J.: Recommender Systems in E-Commerce. 1st ACM Conference on Electronic Commerce (EC'01), Proceedings. ACM Press (1999).

2 Timo Asikainen
Methods for Modelling the Variability in Software Product Families

Methods for Modelling the Variability in Software Product Families[1]

Timo Asikainen

Helsinki University of Technology, Software Business and Engineering Institute,
P.O. Box 9210, FI-02015 TKK, Finland
timo.asikainen@tkk.fi

Abstract. Variability is the ability of a system to be efficiently extended, changed, customised or configured for use in a particular context. There is an ever-growing demand for variability of software. Software product families are an important means for implementing software variability. A software product family may contain very large numbers of individual products. Consequently, methods for representing the variability and efficiently reasoning about it are needed. This thesis studies such methods: the goal of the thesis is to define a solid conceptual basis for modelling the variability in software product families, and to provide the concepts formal semantics in such a way that reasoning on the models is possible using existing inference tools. Major parts of the work have already been completed and documented in a number of publications.

1 Introduction

Variability is the ability of a system to be efficiently extended, changed, customised or configured for use in a particular context [1]. There is a growing demand for variability of software, and a significant research interest in the topic, as exemplified by the workshops and special issues devoted to it, see, e.g., [2]. Products that incorporate variability are useful for various purposes: for example, such products can address multiple user segments, allow price categorisation, support various hardware platforms and operating systems, offer different sets of features for different needs, and cover different market areas with different languages, legislation, and market structure. Addressing these concerns without variability would be very difficult, if not impossible.

Software product families, or *software product lines*, as they are also called, have become an important means for implementing variability [3]. A software product family may contain very large numbers of individual products. Consequently, methods for representing the variability and efficiently reasoning about it are needed.

This thesis aims at developing methods for modelling the variability in software product families. The most important modelling concepts and constructs in the methods developed stem from existing methods used for representing the variability in

[1] PhD work, 4th year

software product families. However, the methods introduced in the thesis are provided with a more solid conceptual foundation and richer sets of modelling concepts.

The remainder of this position paper is structured as follows. Next, in Section 2 we provide a brief overview of the related previous work. The research problem is defined in Section 3, along with research questions and goals. Thereafter, in Section 4 we discuss the results achieved so far. An outline for further work follows in Section 5.

2 Previous work

This section provides an overview of the previous work on modelling variability in software product families and identifies an area of research in which more work is needed.

Numerous methods for modelling the variability in software product families have been proposed. In general terms, a *decision model* specifies the decisions that must be made to produce an individual product in the family and the order of these decisions [4]. Such decisions are often termed *variation points*.

A practically important class of variability modelling methods is based on modelling the common and variable *features* of a product family. An example of such a method is FODA (Feature Oriented Domain Analysis) [5]. A number of methods for modelling variability in *product family architectures* have been reported; Koalish [6] and xADL 2.0 [7] are examples of such methods. Arguably, variability models based on features or architecture can be considered to be instances of decision models: both of these span a set of decisions that must made in order to produce an individual product in the family. *Domain-specific languages* may also be used to express variability in software product families [8].

Variability has also been studied in the domain of traditional products, i.e. mechanical and electrical ones. This domain of research is called *product configuration*, or *configuration* for short, and it studies how a general design of a product can be modified in prescribed ways to produce product individuals that match the specific needs of customers [9]. In contrast to methods for modelling variability in software product families, the results achieved in product configuration domain include a number of conceptualisations of the domain [10, 11]. The conceptual work done in the domain has lead to a large number of successful applications [9, 12, 13].

Although there are a relatively large number of studies on variability of software, there is still need for further research on the topic. The conceptual foundation of the modelling methods is in many cases unclear: in many methods, the concepts and their interrelations are not defined at all, or in an unsatisfactory manner; conceptual work similar to that done in the product configuration domain could alleviate this condition. The semantics of the modelling concepts is in most cases not rigorously defined. Many practically relevant aspects have not been studied in depth. Configuration of individual systems over multiple stages [14] or *binding times* is widely acknowledged to be an important topic. Yet most existing methods for modelling variability do not account for multiple binding times. *Constraint languages* used in expressing dependencies between different decisions or variation points are either simplistic, including

only constraints of the form *A requires B* and *A excludes B*, or described cursorily, e.g., by referring to existing constraint languages, such as OCL [15], without studying the applicability of these languages to variability modelling in any detail. Also, we do not know any feature modelling methods that would account for the *evolution*, i.e., changes over time, of variability models.

3 Research Problem, Questions, and Method

The research problem is the *study and development of methods for modelling the variability and commonality in software product families*. In more detail, the thesis aims at answering the following research questions.

1. What concepts are suited to modelling variability and commonality in software product families?
2. What is the formal or rigorous semantics of these concepts?
3. What kind of languages can be built on these concepts?
4. What kind of tools can support the use of these languages and methods?

Related to the fourth point, there should be support for two tasks: the *modelling* and the *configuration task*. The former pertains to creating a model of the variability in a software product family. The latter, in turn, pertains to finding a *configuration*, i.e., a description of an individual product in the family, matching a given set of requirements at hand.

The research method applied in this thesis is a *constructive* one [16]. In short, applying the constructive research method pertains to building an artefact that solves a domain problem in order to create knowledge about how the problem can be solved and the solution artefact compares with previous solutions to the same problem.

4 Results achieved

In this section, we provide an overview of the results achieved so far.

The results achieved so far can be classified based on the underlying modelling concepts; a distinction between results on *feature modelling*, *architecture description*, and results *integrating* these two views can be made.

Forfamel is a method for modelling the variability in software product families from a feature point of view. The conceptual basis of Forfamel is defined in [17]. Forfamel includes the definition of the concepts of the method, and their informal but rigorous semantics. Forfamel synthesises a number of existing feature modelling methods, which gives it a solid foundation. Further, it previous work on features with a number of concepts and constructs from the product configuration domain. Forfamel is provided with formal semantics by translating it to Weight Constraint Rule Language (WCRL) [18], a general purpose knowledge-representation language similar to logic programs [19]. Although general-purpose, WCRL has been designed to allow the easy representation of configuration knowledge about non-software products and shown to suit this purpose [20]. This suggests that WCRL is a reasonable

choice for the knowledge representation formalism of our approach as well. Further, an inference system *smodels*[2] operating on WCRL has been shown to have a very competitive performance compared to other problem solvers, especially in the case of problems including structure [18].

Further, [21] shows how an existing prototype product configurator, WeCo-Tin [22], can be used to provide tool support for modelling and configuring the features of a software product family; it should be noticed that the feature modelling concepts studied in this paper are not those of Forfamel, but another synthesis from previous feature modelling methods. The configurator provides support for both the *modelling* and *configuration task*. The paper shows that existing tools, originally intended for describing the physical structure of non-software products, can be applied to software products.

As for architecture description, [23] contains an analysis of three architecture description languages (ADLs), and compares them with a configuration ontology originally developed for non-software products [10]. The outcome is that the ontology is able to capture most, but not all of the concepts of the ADLs. Hence, [23] shows that configuration modelling concepts provide a basis on which architecture-based modelling methods for configurable software product families can be built on, but is not as such applicable to modelling architectures.

Further, [24] contains the definitions of a conceptualisation, i.e., a domain ontology, called *Koalish* for modelling architecture of configurable software product families. In more detail, Koalish is based on Koala [25], a component model and architecture description language (ADL), developed at Philips Consumer Electronics. Koala is, to the best of the author's knowledge, the only ADL that has been applied in the industry. Hence, its practical success gives Koalish a solid foundation. Koalish extends Koala with concepts and constructs for modelling variability. Finally, similarly as for Forfamel, Koalish is provided with formal semantics by translating it to WCRL.

The definition of a language based on Koalish is contained in [6]; this language is likewise called Koalish. In addition, an approach for managing configurable software product families is outlined. The approach is based on providing tool support for the modelling and deployment tasks. The modelling task was defined in Section 3 above. The deployment task, in turn, consists of the configuration task and the additional steps required to turn the description of an individual product into a concrete product. Together, the language and the outlined process form a solid basis on which tools supporting architecture-based configurable software product families.

Kumbang [26] is an approach integrating Forfamel and Koalish. Thus, Kumbang enables modelling variability simultaneously from a feature and an architecture point of view and the interrelations between these two views using constraints. The work includes a UML (Unified Modeling Language) stereotype illustrating the modelling concepts of Kumbang and those of UML. A number of case products inspired by real-life software product families have been modelled using Kumbang by our research team. Kumbang provides a sufficient level of support to capture the intent of

[2] See `http://www.tcs.hut.fi/Software/smodels/`

the product families. The cognitive effort required to create the models has been moderate.

Finally, Kumbang Configurator is a prototype tool that supports the configuration task for Kumbang, and hence also Forfamel and Koalish, models [27]. The configurator includes an implementation of the Kumbang. The configurator has performed well when applied to the case software product families mentioned in the previous paragraph.

5 Further work

This section discusses further work needed to complete the thesis. It is still unclear which extensions will be included in the thesis; it is unlikely that all of them would be included.

Further work should take place in three main areas. First, it is possible to extend the conceptual basis with new modelling concepts and constructs. Second, theoretical studies can be carried out to add rigour to the possibly extended modelling concepts. Finally, empirical studies can be carried out to demonstrate the practical applicability of Kumbang.

There are a number of possible ways to extend the conceptual basis of Kumbang. An essential extension is to define a constraint language to be used with Kumbang: constraints are needed to specify dependencies both within a single view and between views. Such a constraint language should resemble existing languages such as the Object Constraint Language (OCL) [15] or xPath (see http://www.w3.org/TR/xpath) and should be an integral part of the modelling method in the sense that it is both possible to check the constraints and efficiently search for a configuration that satisfies the constraints in the configuration model.

It is also possible to extend Kumbang with concepts and constructs for modelling the evolution of software product families, similarly as has been done in xADL 2.0 [7].

An issue often discussed in conjunction with variability are *binding times*: a configuration is not produced during a single step but during multiple steps where the output of the previous step serves as an input for the following steps [14]. However, the notion of binding times and their semantics has not yet been thoroughly studied or understood. Hence, augmenting the modelling methods developed in the thesis could both improve their usefulness and contribute to the area of research.

Another possibility is to extend the modelling concepts in such a way that the user would define the views used in a particular model. That is, the set of views in the modelling method would not be fixed to, e.g., a combination of feature and architectural views. Instead, the number of views, the properties of each view, and the possible interrelations between views could be specified to match the particular requirements of the domain at hand. This extension is motivated by the fact that the number and characteristics of the views required to model the variability in a software product family depends on the particular domain and family at hand. It seems that no single set of views suits all domains.

An example of a practical domain with more than two views is car periphery systems at Robert Bosch GmbH [28]. In this domain, four views are used: the environment in which the device is located, the features of the software, the architecture of the physical device in which the software is embedded in, and the architecture of the software itself and how it is deployed to physical components.

To make the theoretical foundation of Kumbang, or an extended method more solid, the method could be provided with even more rigorous formal semantics than has been done so far for Kumbang. Such semantics could also be used to perform theoretical complexity analysis and other relevant properties of the methods.

Demonstrating the practical applicability of the results requires testing the methods empirically with real software product families in real software development contexts. The tests should concern both their expressive power and usability. The same method, applied to a sufficiently wide range of different kinds of configurable software product families, can also be used to analyse their scope of applicability.

References

1. Svahnberg, M., van Gurp, J., Bosch, J.: A Taxonomy of Variability Realization Techniques. Software - Practice and Experience 35(8) (2006) 705-754
2. Bosch, J.: Software Variability Management (introduction to special issue on software variability management). Science of Computer Programming 53(5) (2004) 255-258
3. Clements, P. C. and Northrop, L.: Software Product Lines - Practices and Patterns. Addison-Wesley, Boston (MA) (2001)
4. Weiss, D. and Lai, C. T. R.: Software Product Line Engineering: A Family Based Software Development Process. Addison-Wesley, Boston (MA) (1999)
5. Kang, K. C., Cohen, S. G., Hess, J. A., Novak, W. E., Peterson, S. A.: Feature-Oriented Domain Analysis (FODA) - Feasibility Study. CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University (1990)
6. Asikainen, T., Soininen, T., Männistö, T.: A Koala-Based Approach for Modelling and Deploying Configurable Software Product Families. In: Proceedings of the 5th International Workshop on Product Family Engineering (PFE-5). Lecture Notes in Computer Science 3014. Springer (2003) 225-249
7. Dashofy, E., van der Hoek, A., Taylor, R. M.: A Comprehensive Approach for the Development of Modular Software Architecture Description Languages. ACM Transactions on Software Engineering and Methodology 14(2) (2005) 199-245
8. Tolvanen, J.-P., Kelly, S.: Defining Domain-Specific Modeling Language to Automate Product Derivation: Collected Experiences. In: Obbink, J. Henk and Pohl, Klaus (eds.): Proceedings of the 9th International Software Product Line Conference (SPLC 2005) (2005) 198-206
9. Soininen, T., Stumptner, M.: Introduction to Special Issue on Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 17(1-2) (2003) 1-2
10. Soininen, T., Tiihonen, J., Männistö, T., Sulonen, R.: Towards a General Ontology of Configuration. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4) (1998) 357-372
11. Felfernig, A., Friedrich, G., Jannach, D.: Conceptual Modeling for Configuration of Mass-Customizable Products. Artificial Intelligence in Engineering 15(2) (2001) 165-176
12. Faltings, B., Freuder, E. C.: Special Issue on Configuration. IEEE Intelligent Systems 14(4) (1998) 29-85

13. Darr, T., Klein, M., McGuinness, D. L.: Special Issue on Configuration Design. Artificial Intelligence for Engineering Design, Analysis and Manufacturing 12(4) (1998) 293-397
14. Czarnecki, K., Helsen, S., Eisenecker, U. W.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practices 10(2) (2005) 143-169
15. Object Management Group: OCL 2.0 Specification. ptc/2005-06-06 (2005)
16. Kasanen, E., Lukka, K., Siitonen, A.: The Constructive Approach in Management Accounting Research. Journal of Management Accounting Research 5(1993) 243-264
17. Asikainen, T., Männistö, T., Soininen, T.: A Unified Conceptual Foundation for Feature Modelling. In: Proceedings of the 10th International Software Product Line Conference (SPLC 2006) (2006)
18. Simons, P., Niemelä, I., Soininen, T.: Extending and Implementing the Stable Model Semantics. Artificial Intelligence 138(1-2) (2002) 181-234
19. Asikainen, T. Modelling Methods for Managing Variability of Configurable Software Product Families. Licentiate thesis, Helsinki University of Technology, Department of Computer Science and Engineering. (2004)
20. Soininen, T., Niemelä, I., Tiihonen, J., Sulonen, R.: Representing Configuration Knowledge with Weight Constraint Rules. In: Proceedings of the AAAI Spring 2001 Symposium on Answer Set Programming: Towards Efficient and Scalable Knowledge Representation and Reasoning (2001)
21. Asikainen, T., Männistö, T., Soininen, T.: Using a Configurator for Modelling and Configuring Software Product Lines Based on Feature Models. In: Männistö, Tomi and Bosch, Jan (eds.): Proceedings of Software Variability Management for Product Derivation - Towards Tool Support, a workshop in SPLC 2004. Helsinki University of Technology, Espoo, Finland (2004) 24-35
22. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A Practical Tool for Mass-Customising Configurable Products. In: Proceedings of the International Conference on Engineering Design (ICED'03), Stockholm, Sweden (2003)
23. Asikainen, T., Soininen, T., Männistö, T.: Towards Managing Variability Using Software Product Family Architecture Models and Product Configurators. In: van Gurp, Jilles and Bosch, Jan (eds.): Proceedings of Software Variability Management Workshop. IWI preprint 2003-7-01. University of Groningen, Groningen, The Netherlands (2003) 84-93
24. Asikainen, T., Soininen, T., Männistö, T.: A Koala-Based Ontology fro Configurable Software Product Families. In: Configuration Workshop of 18th International Conference on Artificial Intelligence (IJCAI) (2003)
25. van Ommering, R., van der Linden, F., Kramer, J., Magee, J.: The Koala Component Model for Consumer Electronics Software. IEEE Computer 33(3) (2000) 78-85
26. Asikainen, T., Männistö, T., Soininen, T.: Kumbang: A Domain Ontology for Modelling the Variability in Software Product Families. Advanced Engineering Informatics (accepted for publication) (2006)
27. Myllärniemi, V., Asikainen, T., Männistö, T., Soininen, T.: Tool for Configuring Product Individuals from Configurable Software Product Families. In: Männistö, Tomi and Bosch, Jan (eds.): Proceedings of Software Variability Management for Product Derivation - Towards Tool Support, a workshop in SPLC 2004. Helsinki University of Technology, Espoo, Finland (2004) 106-109
28. Thiel, S., Ferber, S., Fischer, T., Hein, A., Schlick, M.: A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems. In: Proceedings of In-Vehicle Software 2001 (SP-1587) (2001) 43-55

3 Nan Frederik Mungard
Feature Model Based Product Derivation in Software Product Lines

Feature Model Based Product Derivation in Software Product Line Engineering

Nan Frederik Mungard

Software Systems Engineering
University of Duisburg-Essen
45117 Essen, Germany
nan.mungard@sse.uni-due.de

Abstract. Given a feature model, the initial activity of application engineering is to determine a set of features that form the configuration of the application. With very large feature models this process becomes tedious, as a great amount of selections have to be made. This paper discusses the challenges to product derivation and lays out requirements for product derivation. As a solution a new model is proposed and realized as a prototype. The model defines a clear terminology for the process and is adaptible to different application scenarios; e.g. staged configuration.

1 Introduction

A widely used notation for modelling domain knowledge of software product lines (SPLs) is feature modelling, first introduced by Kang et al. [KCH+90]. A feature model uses features, domain relations, and dependencies to express the concepts and the variability of the domain. Based on the feature model the application engineers have to derive a consistent and fully bound selection of features that form the configuration. Consistent means that the semantic of the feature model is not violated. Fully bound means that the variability of the feature model is completely resolved.

In my diploma thesis at the Research Group Software Construction, RWTH Aachen, I focussed on the process of deriving a configuration from a feature model. My research interest was to describe and analyze this process. In the following, I present my work.

Limitations The evolution of feature models is not considered in this paper. Also only a very basic feature modelling notation is used which does not encompass cardinalities, attributes, and complex dependencies.

1.1 Outline

This paper is organized as follows. In Section 2 the challenges of product derivation in general and with regard to feature models are discussed. In Section 3 a model for product derivation is introduced. In Section 4 different applications of

the model are proposed. Section 5 describes this work's approach to validation and the realized prototype. An overview of related work is given in Section 6. Section 7 summarizes the work.

2 Motivation

At the Software Product Lines Conference 2005 Jan Bosch described the key challenges product derivation faces today. "The basic problem of product derivation is the enormous size of software product lines in industrial settings. [...] The number of variation points easily ranges in the thousands and may even exceed ten-thousand in some product families."[Bos05]. Additionally, he criticized that the process too heavily depends on experts.

The obvious consequence of large SPLs is that product derivation cannot be considered as a single user problem. Instead a scenario where several stakeholders cooperate, each contributing their expertise to specific parts of the derivation, is far more realistic. In order to cooperate, responsibilities, roles, and rights are needed; a process has to be defined. A second consequence of large SPLs is that the importance of efficiency increases. E.g.: While selecting features in a list of some 100 entries might be considered reasonable, the same is not true for a list of 10.000 entries.

The problem of the dependency on experts might simply be discarded as being another consequence of the size of current SPLs. Clearly, the size of a SPL influences its complexity, as the number of possible configurations grows exponentially. However, the question has to be raised whether all of the experts' knowledge has been captured. Especially procedural knowledge, i.e. *how* to do something, is often overlooked.

The challenges identified above apply to product derivation using feature models, too. Furthermore it is important to note that a feature model cannot capture procedural knowledge.

I identified further challenges that apply to feature model based product derivation and current tools:

- **Terminology** The *terminology* of the product derivation process is not defined. The most prominent example is the use of the word *configuration* which refers either to the process or the result of the process. Another example is that most papers do not state who is responsible for what. E.g.: Who sets up the stages or product sets?
- **Product Hierarchy and Context** While in [KCH+90] great effort is placed on creating a complete hierarchy of the features of the domain, the modelling of the hierarchy of the products of the domain is neglected. Of course, this is done on purpose as the commonalitites and differences of products are expressed through the feature selection. But as pointed out by [RW05] similar feature selections do not have to have similiar rationales.
- **Not a Uniform Process** Feature model based product derivation is not a uniform process. E.g.: A sales person should have different rights than a

domain engineer. A software product line can or cannot allow for a delta in the derived configurations.

– **Conflict and Inference** Conflict prevention and inference are useful mechanisms, but they should be handled with care. Inference may lead a user to overlook problems in the configuration, when the system informs him, that the configuration is *done*. Conflict prevention "only conceal[s] the basic issues of disagreement rather than resolve them"[Mar99]. In general, the question has to be raised whether a product derivation tool for software product lines is an expert system. Or whether the user is the expert whom the system supports (*decision support system*).
– **Not enough abstraction** When several users cooperate an additional abstraction layer is needed to separate the user decisions from the configuration values. This way, redundant or conflicting decisions of differenzt users can be documented without being lost.

3 Product Derivation Model

To address the above mentioned challenges I developped a model for product derivation. The main task was to define a terminology for the artefacts of product derivation. In the following, the artefacts as well as their relations and dynamics are described.

Basically, the product derivation process consists of *derivation decisions* where a user defines *assignments* for one or more features (see Figure 1). He can either *include* or *exclude* a feature. By applying the given assignments to the semantics of the feature model a *mapping* of each feature to a *status* can be determined. A feature can be *included* or *excluded* from the configuration. *Open* means that no assigments cause either the inclusion or exclusion of the feature. If different assignments include and exclude a feature, the status of the feature is *inconsistent*.

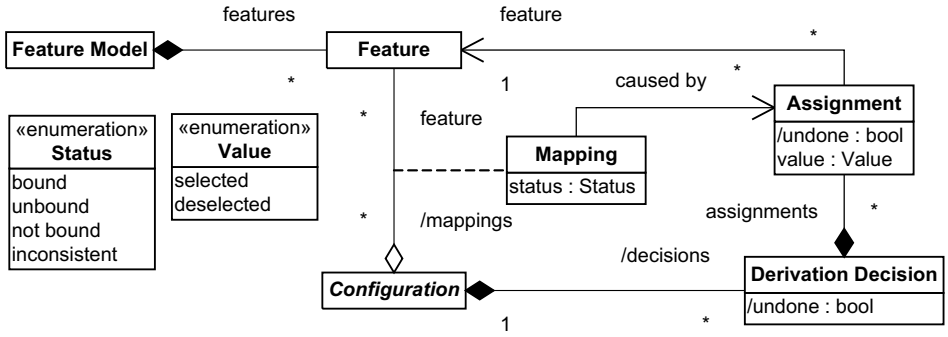

Fig. 1. Storing of Values

A configuration is edited by performing *derivation steps* (s. Figure 2). The basic step is the *feature step*. It represents the choice to assign a value to the associated feature. Through *collections* several steps can be grouped together. If the grouping is ordered it is a *sequence*, if not it is a *selection*.

When a derivation step is *performed* the choices represented by the derivation step can be made. A performed feature step allows to either include or exclude a feature from the configuration. Performing a sequence means that the elements of the sequence are performed sequentially. A performed selection allows to simultaneously perform all its elements.

Afterwards the derivation step has to be *finalized* in order to apply the new assignments to the configuration. This is only possible if the *target* of the derivation step was met. The target can either be *unchanged* or *fully specialized*. A feature step is fully specialized if the feature is either included or excluded. A collection is fully specialized if all the targets of its elements are satisfied. The result of a finalized derivation step is a *derivation decision*. The decision stores all assignments made.

Derivation decisions can be undone. An undone derivation decision is not removed from the configuration, as it still represents an expressed user second type of undo is to undo a value for a feature. Undoing a value means that each assignment of the value to the given feature is ignored. An undone decision or assignment is signalled by the *undone* attribute.

Configuration Types In this model two general types of configurations are differentiated (see Figure 3). *Workpieces* are configurations that can be edited, i.e. on which derivation steps can be performed. *Baselines* are configurations that represent a lasting result of the derivation process and that cannot be altered. Baselines again fall into two types. *Products* are consistent and fully bound configurations. They form the basis of the application design and implementation. A *master* is a branching point in the derivation process. The most prominent master configuration is the *platform*.

Each configuration holds a set of derivation steps. Every editor of a configuration can perform its derivation steps. This allows for a configuration based reuse of derivation steps.

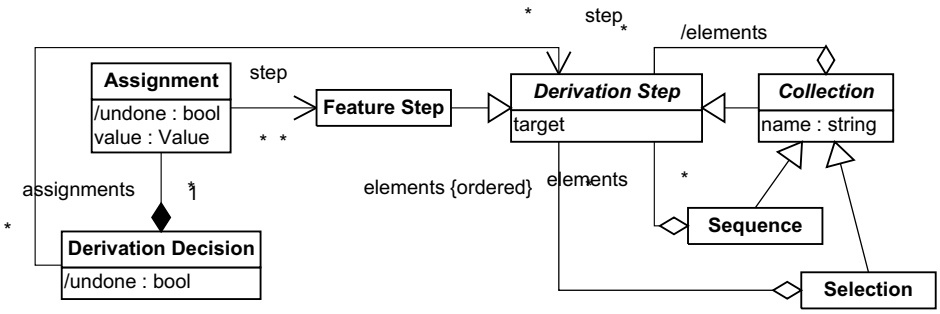

Fig. 2. Derivation Steps

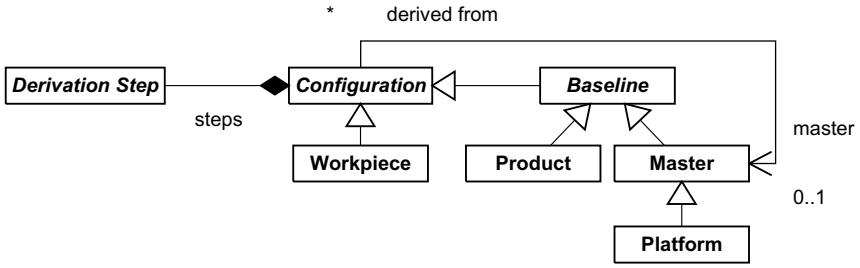

Fig. 3. Hierarchy of configurations

In Figure 4 the transformations of configurations are shown. A workpiece is created by deriving it from a master. It is a copy of the master and can be edited. When an important point in the derivation has been reached a snapshot of the workpiece can be taken. The snapshot is again a copy. It is always possible to create another master. This way a master can be used to store temporary results and create new starting points for product derivation. To create a new product, however, the workpiece has to be fully bound and it has to be consistent.

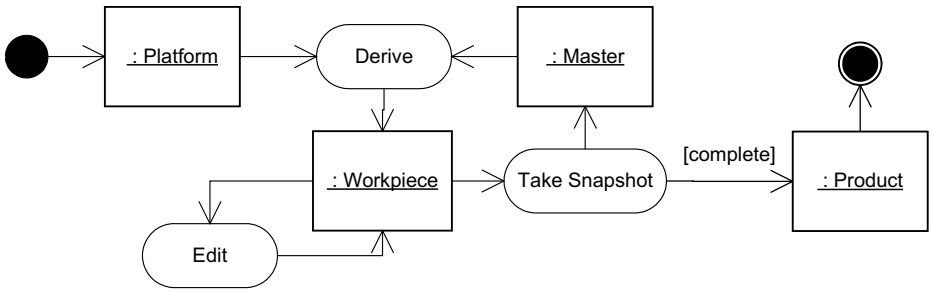

Fig. 4. Transformations of Configurations

Roles During product derivation a user can play several roles. Each user who participates in the product derivation is a *participant*. Each configuration has a *derivation manager* who administrates the configuration. This encompasses adding and removing the participants and defining derivation steps for the configuration. Workpieces can have an *editor* who is able to perform derivation steps leading to derivation decisions (see Figure 5). Every participant of a master configuration can derive new workpieces. The deriving participant is then automatically assigned as editor of the workpiece. Afterwards the derivation manager of the workpiece assigns the following editors.

Each user has a set of derivation steps for personal use, that he/she can apply to different workpieces. The derivation steps of the user allow for a user based reuse of derivation steps.

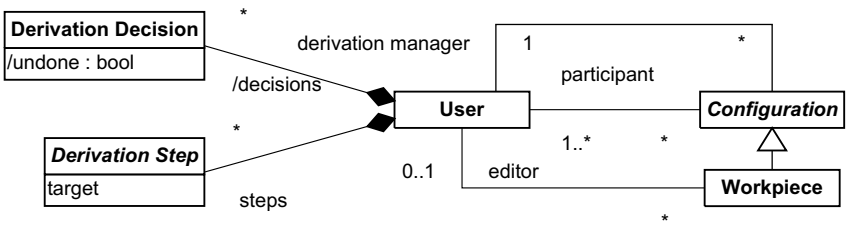

Fig. 5. Roles

To taylor the process to the specific requirements of a given situation the model relies on *derivation parameters*. A derivation parameter characterizes a user's role and access during product derivation. Besides the following six derivation parameters, more differences between derivation processes are thinkable and will be considered in my future work.

- **Access Region**: The access region refers to the features for which the user can execute derivation steps. Possible values are: derivation manager, partial, empty. For partial access a set of features has to be defined for which the user has access. The value derivation manager implies that the user has the access region of the derivation manager.
- **Undo**: *Undo* determines what decisions a user can undo. Values are: own, team, derivation manager. Team refers to all decisions of current participants. Derivation manager is analogous to access region.
- **Conflict Prevention**: *Conflict prevention* specifies if the system should actively prevent the user from making assginments that lead to conflicts or not. Possible values are: full, own, none. Own means that conflicts are only prevented between decisions the user has taken himself.
- **Inference Information**: *Inference Information* defines what information the user receives during product derivation about the computed state of the configuration. Possible values are: full, own, none. Own means that the user is only informed about inferred values resulting from his own decisions.
- **Own Steps**: If *own steps* is enabled a user can freely select between steps provided for by the configuration and his own steps. If not, the user is limited to the steps provided for by the configuration.
- **Instance Manager**: A derivation manager of a master configuration can specify who becomes *instance manager* if a new workpiece configuration is instantiated by a participant.

The derivation manager has to set all derivation parameters for each participant of the product derivation other than himself. The derivation parameters of the derivation manager are set by his precursor and cannot be changed.

4 Application Scenarios

In this section a few application scenarios that can be supported with the model are introduced. They show different approaches to product derivation.

- **Toolbox**: Experts store the derivation steps they consider useful in order to reuse them. If they have a specific order in which they resolve the variability of the feature model, they create an appropriate sequence. If they are interested in a specific view on the feature model they sets up a corresponding selection.
- **Prescription**: The domain experts want to make sure that the sales staff performs specific steps when contacting the client for the first time. They set up a master for the sales staff with a sequence containing all relevant derivation steps. As target he chooses *fully bound* for the derivation step and its elements. Furthermore they forbid own steps for the sales staff. Whenever the sales staff instantiates a new workpiece they are limited to using the given sequence.
- **Staged Configuration**: A derivation manager can perform a staged configuration simply by setting the undo and access region derivation parameters of the participants accordingly.
- **Voting**: Several users are asked to record their selections for a specific set of features. The set of features is given in the form of a sequence. They are not informed about inferred values at all. Conflict prevention is also disabled. Afterwards all records are compared. Voting is a classical approach for finding a decision in a group.
- **Joined Optimization**: Several experts cooperate during the derivation. They are responsible for different aspects and their decisions have the same priority. They are only informed about the system state resulting from their own decisions. Afterwards existing conflicts in the configuration are solved together.
- **Flat Hierarchy**: A flat hierarchy can also be realized in the given model. All users are participants of the platform configuration. They are assigned as instance managers and all other derivation parameters are set to the least restrictive value.

The flat hierarchy scenario on the one hand and e.g. the joined optimization szenario on the other hand, indicate the ability of the process to scale.

5 Approach to Validation

In order to demonstrate the concepts I realized a prototype. In the following the prototype is described and the next steps for validation are discussed.

Prototype The prototype was created within RequiLine [vdML03] using the C# programming language. It differentiates between master, product, and workpiece configuration. Through derivation parameters and the definition of derivation steps, the process can be adapted to specific situations. Edit and undo are seperated. If a user can use own steps, the system provides a generic derivation step that corresponds to his access region. The undo can be performed on a decision and a value base. In Figure 6 a screenshot of the edit window is shown.

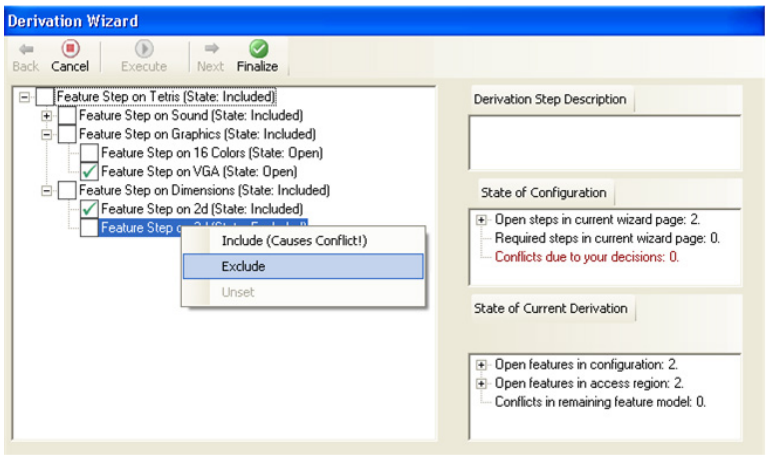

Fig. 6. Screenshots of the Prototype: Performing Derivation Steps

Inference Engine The inference engine was implemented according to the algorithms described in [Bat05]. However, some modifications were necessary. An additional feature state *inconsistent* was introduced to encompass conflicting assignments. Any clause that contains an inconsistent feature is not evaluated by the engine. In order to allow for undo a reset operation was needed. It operates by recalculating all values based solely on the stored premises.

A problem I identified was *conflict resolution*. In my model a feature can have multiple assignments, resulting in multiple causes for a given mapping. Furthermore the derivation parameters can severely restrict a user in resolving a given conflict. Therefore the task of conflict identification and resolution is substantially more complicated.

Next steps On basis of the prototype extensive user studies are due. The initial focus should be the evaluation and improvement of the GUI and of the functionality of the prototype. If the initial phase is successful, the improved prototype should be applied in an industrial setting to demonstrate its usefulness.

6 Related Work

To cope with large SPLs Reiser and Weber propose the usage of *product sets* [RW05]. A product set represents a property and has sets of included and excluded features associated to it. By assigning a configuration to a product set the features are included and excluded according to the associated feature sets. The authors point out, that while product sets might contain the same features, there could still be the need to differentiate between them based on their rationales.

Another contribution comes from the ConIPF project [Con06]. Although the ConIPF approach to product derivation is not based on feature models they raise

important points, especially the importance of procedural knowledge. They propose the usage of *strategies* which contain configuration steps [KWR03]. Each strategy is annotated with a *priority* and a *precondition*. During product derivation the strategy with the highest priority whose precondition is fulfilled is executed.

Czarnecki, Helsen and Eisenecker introduce the concept of *staged configuration* [CHE04]. In staged configuration the product derivation consists of stages performed by different users. In each stage the user specializes the feature model, that is, the user reduces its variability. The decisions taken in former stages cannot be undone in later stages. This process is repeated until all variability has been eliminated. *Multi-level configuration* is an extension of staged configuration [CHE05]. Here the product derivation consists of executing levels in a predefined order. Each level has a feature model that describes the choices a user can take. The choices of earlier levels additionally limit the choices in later ones.

Existing Tools Most current tools in the field of feature model product derivation, e.g. [CAK+05,psG04], offer a tree view that corresponds to the feature model. The user interactively includes and excludes features. The system then calculates the inferred values and displays them. They differentiate between user and system decisions, but not between different user decisions. Undo is performed directly on the values of the configuration. Procedural knowledge cannot be stored. Furthermore most settings are global and not user or configuration specific. [psG04] has a conflict resolver and inference is optional. Additionally it offers filters that can be used to store specific views of the feature model. [CAK+05] supports staged configuration.

7 Conclusion

In this paper the case for new requirements for feature based product derivation has been made. Product derivation in large software product lines is a multiuser process. It should be supported by a well defined process with clear roles and rights. The process should be adaptable to the needs of specific projects. Configurations should not be edited directly, but computed based on the given user input. Also the question of reuse should be handled systematically.

To support this claim a process and prototype have been realized and an approach to validation has been layed out. Mappings are computed based on decisions. Collections are a simple and intuitive way to store procedural knowledge. Reuse has been defined on a user and on a configuration level. The derivation parameters are examples of potential and important differences in product derivation processes. Additionally, a clear vocabulary for the process has been presented.

Future Work The derivation decisions could be used to allow for a *merge* approach as proposed in [CHE05]. Evolution could be supported by the derivation

hierarchy which represents a clear path to propagate changes. If and how evolution decisions taken for a master configuration apply to its derived configurations could be included in the model as derivation parameter.

Acknowledgements I would like to thank Thomas von der Maßen, Alexander Nyssen, and Prof. Lichter of the Research Group Software Construction, RWTH Aachen, for supporting me with my diploma thesis.

References

[Bat05] Don S. Batory. Feature models, grammars, and propositional formulas. In Obbink and Pohl [OP05], pages 7–20.

[Bos05] Jan Bosch. Software Product Families in Nokia. In Obbink and Pohl [OP05], pages 2–6.

[CAK+05] Krzysztof Czarnecki, Michal Antkiewicz, Chang Hwan Peter Kim, Sean Lau, and Krzysztof Pietroszek. fmp and fmp2rsm: eclipse plug-ins for modeling features using model templates. In Ralph Johnson and Richard P. Gabriel, editors, *OOPSLA Companion*, pages 200–201. ACM, 2005.

[CHE04] Krzysztof Czarnecki, Simon Helsen, and Ulrich W. Eisenecker. Staged configuration using feature models. In Robert L. Nord, editor, *SPLC*, volume 3154 of *Lecture Notes in Computer Science*, pages 266–283. Springer, 2004.

[CHE05] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process: Improvement and Practice*, 10(2):143–169, 2005.

[Con06] Conipf - configuration in industrial product families, 2006. http://www.conipf.org/ [Online; accessed 20-July-2006].

[KCH+90] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, November 1990.

[KWR03] Thorsten Krebs, Thomas Wagner, and Wolfgang Runte. Recognizing user intentions in incremental configuration processes. In *Proceedings of Configuration - IJCAI 2003 Workshop*, pages 44–50, August 2003.

[Mar99] George M. Marakas. *Decision support systems in the twenty-first century.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.

[OP05] J. Henk Obbink and Klaus Pohl, editors. *Software Product Lines, 9th International Conference, SPLC 2005, Rennes, France, September 26-29, 2005, Proceedings*, volume 3714 of *Lecture Notes in Computer Science*. Springer, 2005.

[psG04] pure-systems GmbH. Technical White Paper: Variantenmanagement mit pure::variants, 2004.

[RW05] Mark-Oliver Reiser and Matthias Weber. Using product sets to define complex product decisions. In Obbink and Pohl [OP05], pages 21–32.

[vdML03] Thomas von der Maßen and Horst Lichter. Requiline: A requirements engineering tool for software product lines. In Frank van der Linden, editor, *PFE*, volume 3014 of *Lecture Notes in Computer Science*, pages 168–180. Springer, 2003.

4 Marcilio Mendonca, Toacy Oliveira, Donald Cowan
Collaborative and Coordinated Product Configuration

Collaborative and Coordinated Product Configuration

Marcilio Mendonca[1], Toacy Oliveira[2], Donald Cowan[1]

[1]School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada
[2] Departamento de Computação, PUC-RS
Porto Alegre, RS, Brasil

{marcilio,dcowan}@csg.uwaterloo.ca, toacy@inf.pucrs.br

Abstract. Product configuration is a key activity of product engineering that regards the constrained combination and parameterization of product line assets as a means to achieve correct software specification. Current product configuration approaches frequently rely on the application engineer to translate user requirements into correct configuration choices. This process is error-prone and risky as requirements may lead to conflicting decisions at configuration time. Indeed, we deem that an important aspect of product configuration has long been neglected: its collaborative nature. In our research, we advocate that product configuration is enhanced by a collaborative perspective, providing that conflicting scenarios are properly handled. We propose an approach to support collaborative and coordinated product configuration by promoting processes to first-order elements for the explicit guidance of configuration decisions. We provide insights on important coordination issues and introduce an algorithm to derive process models from annotated feature models to illustrate the approach's feasibility.

Classification: Ph.D., 3rd year.

1 Introduction

Product configuration is a key activity of product engineering that regards the constrained combination and parameterization of product line assets as a means to achieve correct software specification. As configurability is a critical issue in product family approaches proper variability management is required. Feature modeling [5] has been well accepted as a technique to capture and represent commonalities and variabilities of product families. Since its inception in 1990, feature models have been enhanced and widely supported [1][4][9][6] motivated by the need for improved automation of production processes. Today, it is common practice to make use of mappings to link features to components of domain-specific languages as means to support automated product generation [1][4].

However, as feature models are experienced in practical scenarios important shortcomings start to arise. First, product configuration is turning into a complex process requiring people with different knowledge, skills and authority to coordinate efforts towards a common goal, i.e., the specification of a valid software

configuration. In [2], some contexts in which product configuration is performed in stages (or collaboratively) are depicted. Nonetheless, current approaches to product configuration mostly rely on the role of the product engineer to properly interpret and translate user requirements into configuration choices. This process is error-prone and may also lead to decision conflicts as the requirements of different stakeholders may be found incompatible at configuration time. Second, although feature models are normally regarded as hierarchical structures with a fairly simple semantic, in practice they resemble *graphs* as opposed to *trees* as a consequence of complex feature dependencies. For instance, semantic feature dependencies have been largely exploited in the realm of *feature interactions* [18][9]. In practice, major consequences are the increased complexity of product configuration and the need for proper coordination of configuration decisions especially when a collaborative perspective is envisioned.

In this paper, we present our research on product configuration. The research aims at investigating alternatives to enhance the configuration process. In particular, motivated by the problems earlier mentioned, we are interested in enabling a collaborative and coordinated product configuration scenario. In such scenario, product configuration is achieved when a group of decision makers (e.g. stakeholders) coordinate their (sometimes conflicting) decisions towards a commonly agreed *configuration model*[1]. We incorporated analysis of such conflicting scenarios in order to properly address coordination issues. The approach relies on process models to describe configuration steps and their order of execution, and also includes an algorithm to derive process models from *annotated feature models*[2]. We expect our approach to be fully applicable as process engines can be used as a runtime environment allowing for the execution of generated process model.

The main contributions of our research include: a new perspective on product configuration that promotes collaboration and coordination throughout the configuration process; various insights on important product configuration issues including *decision conflicts* and *decision propagation*; an algorithm to derive process models from annotated feature models; the development of a support tool that demonstrates the feasibility of the approach in a practical context.

The remainder of this paper is organized as follows. Section 2 provides background and related work on product configuration and software processes. In section 3, we present our approach to collaborative and coordinate product configuration. Section 4 discusses the current status of our research. We conclude the paper and discuss the next steps in our research in section 5 and provide references in section 6.

2 Background and Related Work

Product Configuration: various approaches to product family engineering have recognized the importance of feature models in supporting product engineering activities, in special, product configuration [5][7][2][1][4]. Kang et al. [5] introduced feature modeling as a domain analysis technique in FODA to represent variability in

[1] The *configuration model* stores the *product decisions* [20] made in the configuration process.
[2] *Annotated feature models* can also be seen as *decision models* [20].

product families. Since then, various enhancements have been proposed to feature modeling in an attempt to boost software automation [1][4]. For instance, in FArM [4] and in generative programming [1], mappings to link configuration models to domain-specific languages are suggested as a means to improve automated code generation. In [8], Griss acknowledged that product configuration can be a complex and coordination-demanding process by stating that "*…as a product is defined by selecting a group of features, a carefully coordinated and complicated mixture of parts of different components are involved*". The complexity of product configuration caused by feature interaction problems was extensively discussed in the literature [18] [9]. Gurp et al. [9] addressed *feature interaction* as a problem of decomposition in which "*…the sum of parts is larger than the individual parts"*, i.e., features may overlap and expose complex dependencies. Thus, for Gurp it is natural to refer to feature models as *feature graphs*. Calder et al. [18] made a comprehensive survey on feature interaction problems using telecommunication systems as motivational examples. Czarnecki et al. [2] points out various contexts in which product configuration is achieved collaboratively (the author named it *staged configuration*). In staged configuration, mechanisms such as specialization and multi-level configuration are used to progressively eliminate configuration options. After a certain number of stages a configuration model is derived reflecting the collaborative decisions made. How conflicting decisions are handled is left open.

Software Processes: the idea of software processes as a means to reduce costs and raise software quality is relatively old. In 1987, Osterweil stated that "*software processes are software too*" [14] suggesting that similarly to software applications processes could be modeled, implemented, tested and more importantly executed. In this sense, executable process models allow not only for the description of collaborative scenarios but also for their automation. When applied to the realm of business, processes are referred to as business processes. Business processes generalize the notion of software processes [13] thus developed technology might also fit well in the software process world. For instance, BPMN [16] is a business process modeling notation that can be used to describe process models. BPMN models can be executed when transformed to other formats such as BPEL [15] models. In our approach, we plan to use BPMN to describe derived process models and BPEL-related technology as an execution environment for such models.

3 Approach

In the next section, we provide a set of definitions and concepts that might prove useful in understanding our approach.

3.1 Concepts and Definitions

Decision: During product configuration, a decision is made when an originally undecided feature, i.e., without any decision state defined, is voluntarily selected or unselected. In principle, decisions are to be made in a top-down fashion following the hierarchical structure of feature models. Thus, decisions made on level-1 enable or disable decisions on subsequent levels. In Fig. 1-A a feature diagram is shown

containing a concept (C), mandatory (F11, F12) and optional features (F13, F14) as well as alternative (F23, F24), inclusive-or (F21, F22), and exclusive-or features (F25, F26). The diagram follows the notation described in [1] and also includes group cardinalities to facilitate understanding. If feature F13 is unselected then level-2 features F23 and F24 will be unselected and consequently not present in the final configuration. Features may also expose constraining dependencies such as *requires* and *excludes*. If a feature A *requires* a feature B it means that if A is selected then the selection of B is also required. It also means that if B is unselected then A must be unselected too. In the example, if feature F22 is selected then so will feature F23 and if feature F23 is unselected then feature F22 should also be unselected.

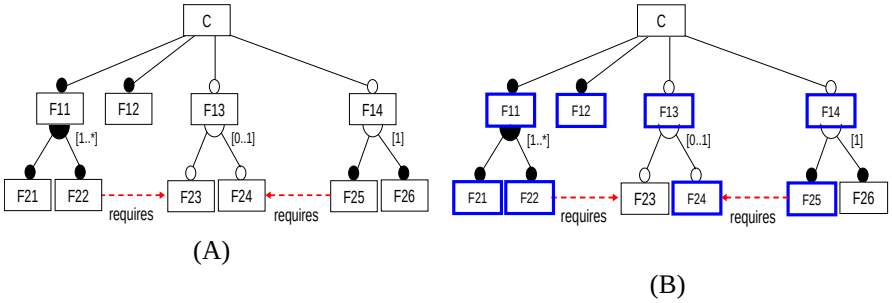

Fig. 1. Example of a Feature Model (A) and a Configuration Model (B)

Decision Conflicts: Because feature models can become *graphs* as opposed to *trees* they are likely to contain conflicting decisions. We say a *decision conflict* occurs when two or more features contain explicit or implicit dependencies that make them rely on the decision state (e.g. selected, unselected) of each other. For instance, in Fig. 1-B, a decision conflict occurred when feature F22 was selected but feature F23 was intentionally unselected. In this case, the conflict can be resolved either by selecting feature F23 or by unselecting feature F22. However, a careful examination will reveal that the problem is much more complex than it appears because of implicit dependencies. Features F23 and F24 are alternative features thus only one can be selected. However, unselecting feature F24 also means unselecting feature F25 because of the *require* dependency. Therefore, features F22 and F24 as well as F22 and F25 are mutually exclusive even though this dependency is not shown explicitly but rather was derived from other dependencies. In general, decision conflicts occur when dependent features hold inconsistent decision states.

Decision Propagation: Decision propagation is the process of propagating a decision throughout the feature model based on feature dependencies. For example, the decision to select feature F22 in Fig. 1-B should be propagated allowing features F23 and F13 to be selected as well as features F24 and F25 to be unselected. As expected, decision propagation only occurs in feature models containing feature dependencies otherwise decisions can always be made in a top-down fashion. Decision propagation is a recursive process. For every feature where a decision has to be made automatically by means of decision propagation it is necessary to identify which other features may also be affected. We identified at least three scenarios in which decisions propagate: i) within a group of alternative, inclusive-or, and exclusive-or features depending on the group cardinality; ii) the ancestor's path of a feature; iii)

the descendants of a feature. For example, when a decision is propagated to select a feature within a group of alternative or exclusive-or features all other features will be automatically unselected. When propagation occurs in an inclusive-or feature group the cardinality of the group has to be taken into account and be deducted by one. In the case of ancestors, all features that are at lower levels and in the path of a feature where decision propagation was applied will also apply decision propagation. In the case of descendants, only mandatory features may apply decision propagation as optional features remain as open decisions. An example of a group decision propagation occurred when feature F24 was automatically selected (because of feature F25 selection) demanding feature F23 to be unselected. As another example, ancestor feature propagation may occur when feature F24 selection triggers feature F13 selection. In this case, decision propagation follows a bottom-up approach which is opposite to the regular top-down flow of decisions in a feature model.

Decision sets: A decision set (DS) encompasses a group of features that will be decided by decision makers playing specific roles (see examples of DSs in Fig. 2). The union of all DSs forms the feature model. DSs are key components to enable collaboration throughout product configuration. A valid DS must comply with the following rules: i) contains at least one open decision; ii) contains a single root node; iii) contains all grouped features of the same feature group; iv) do not overlap open decisions with other decision sets

Decision roles: Decision roles (DR) are the means to assign configuration decisions to different people involved in the product configuration process. DRs are linked to one or more decision sets. The person playing a particular decision role is responsible for making decisions on all attached decision sets (see Fig. 2).

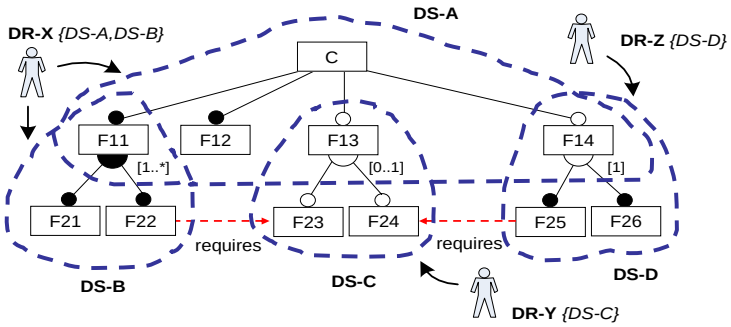

Fig. 2. Feature model annotated with different decision sets and decision roles

Conflict Resolution: Conflict resolution relates to the strategy adopted for resolving decision conflicts. In a *proactive* perspective potential conflicts are anticipated and solved beforehand (pessimistic approach) whereas in a *reactive* perspective conflicts are solved iff they occur (optimistic approach).

Priority Scheme for Decision Conflicts: In order to properly solve decision conflicts a priority scheme has to be specified. For each potential conflict scenario priorities are assigned to participant decision roles in order to define their decision-power. For instance, for a web portal system product line the *project manager* role can be assigned the highest priority followed by the *database manager* role in a conflict

scenario that involves the selection of the database system for a particular web portal instance. Therefore, if there is a disagreement on the database system chosen the project manager's decision will prevail.

3.2 Collaborative Product Configuration

Our approach is depicted in Fig. 3. The first step (Fig. 3, top arrow) indicates the derivation of software processes from annotated feature models. That is, the feature model is decorated with decision sets and decision roles, and then a transformation process takes place producing a process model. As mentioned in the previous section, process derivation may result in decision conflicts that require decision makers to define the precedence of the conflicting decision sets. Decision sets without dependencies will be forked while conflicting sets will be ordered sequentially according to a specified precedence.

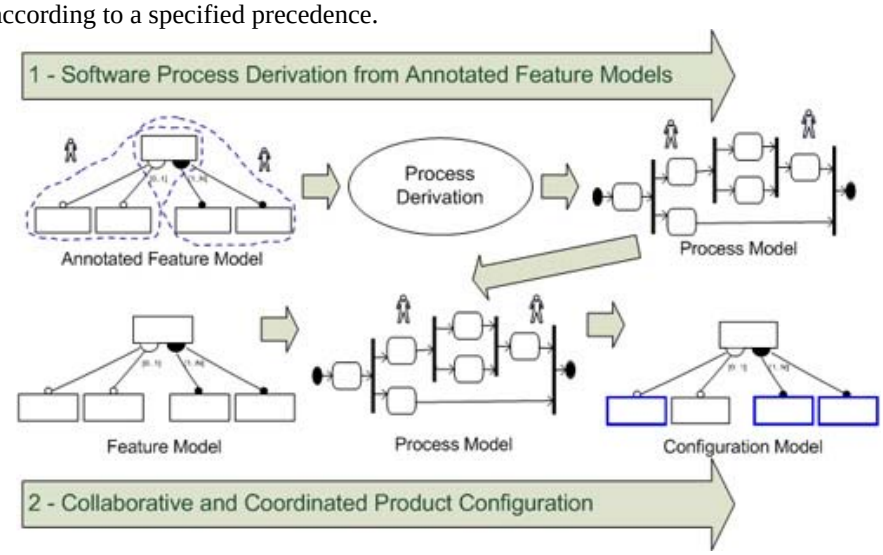

Fig. 3. Overview of the Approach

The second step (Fig. 3, bottom arrow) represents product configuration, in the case, as a collaborative and coordinated process. The process model produced in step 1 can be executed by a process engine allowing decision makers to operate simultaneously over feature models yet in a coordinated manner. In the end, a valid configuration model is produced since all feature model constraints were enforced in the process model. In the following, we provide an overview of the transformation algorithm to derive process models (represented by the *process derivation* ellipse in Fig. 3).

Algorithm: From annotated feature models to process models

1. Reads and validates input data (feature model, decision sets, and decision roles).
2. Identify and resolves decision conflicts.
    a. Identifies conflicting decision sets.

   b. Applies decision propagation to expand conflicting decision sets.
   c. Shows the list of conflicting sets to the user.
   d. Updates decision sets precedence's table based on the *priority scheme*.
3. Builds the process model.
   a. Navigates hierarchically over the feature model (top-down). If found decision sets have no conflicting decisions, specifies precedence: upper-level precedes lower-level; builds a process step for each decision set found. Otherwise: specifies precedence according to user inputs and builds a process step for each decision set.
   b. Builds transitions between decision sets: *fork* for independent sets and *sequence* for dependent sets.
   c. Specify pre/post conditions for each step. Pre-condition: for each process step checks whether the corresponding decision sets still have open decisions. Post-condition: for each process step makes sure that corresponding decision sets have no open decisions left.
   d. Assign decision sets to decision roles
   e. Generate the process model
4. Validates generated process model.

The algorithm begins by reading and validating the inputs, i.e., the feature model, the decision sets and the decision roles. Following this step, decision conflicts are searched and if decision sets are found conflicting, the user is presented with necessary information to specify the desired precedence. At this time, the user is prompted to indicate which decisions should prevail. Then, the process model starts to be assembled. A hierarchical navigation over the decision sets is performed in order to determine sequential and parallel sets. Conflicting decision sets are ordered to reflect the precedence indicated by the user. Then, transitions are specified with pre and post-conditions, and decision sets are assigned to decision roles. Finally, a process model is produced and validated, and the process is finalized. The users in this context are the decision makers involved in conflicting decisions.

Example: applying our approach on the annotated feature model of Fig. 2

In Fig. 2, a feature model is decorated with decision sets DS-A, DS-B, DS-C, and DS-D and the decision roles DR-X, DR-Y and DR-Z. Decision sets were properly assigned to decision roles (represented by the curly brackets in the figure). It is important to notice that the specification of decision sets and decision roles is flexible allowing organizations to (re)arrange the sets in a way that is appropriate to their needs. Let us now discuss step-by-step how a process model is derived using the annotated feature model described in Fig. 2 as the input.

First, the feature model and all decision sets and decision roles are validated. Then, conflicting features and decision sets are discovered as illustrated in Table 1 (first and second columns). The features F22 and F23 as well as features F24 and F25 expose dependencies. Hence, decision sets DS-B and DS-C as well as DS-C and DS-D represent conflicting sets.

Table 1. Decision conflicts and decision propagation

Conflicting Features	Conflicting Decision Sets	Propagated Features	Propagated Decision Sets
F22, F23	DS-B, DS-C	F13, F24, F14, F25, F26	DS-D, DS-A
F24, F25	DS-C, DS-D	F14, F26, F13, F23, F22	DS-A, DS-B

In the next step, decision propagation is applied to find implicit feature dependencies. As shown in Table 1 (third and forth columns), the dependency between features F22 and F23 is propagated and features F13, F24, F14, F25, and F26 are found implicitly connected. The same process of decision propagation is applied to features F24 and F25 and an expanded conflicting list is found as also shown in Table 1. Notice that feature F11 is left out since it is mandatory for all family members thus there's no need to propagate a decision to select or unselect this feature. The user is then presented with a high-level interface[3] for conflict resolution. Based on the user choices a precedence list is defined. In our example, the six possible precedence lists are: {DS-A,DS-B,[DS-C],(DS-D)}, {DS-A,DS-B,[DS-D],[DS-C]}, {DS-A, [DS-C], DS-B, [DS-D]}, {DS-A,[DS-C],[DS-D],DS-B}, {DS-A,[DS-D],DS-B,[DS-C]}, and {DS-A,[DS-D],[DS-C],DS-B}. Square brackets indicate optionality, i.e., previous decisions may automatically resolve subsequent open decisions. Parentheses indicate that the decision set contains no open decisions as a consequence of previous decisions made.

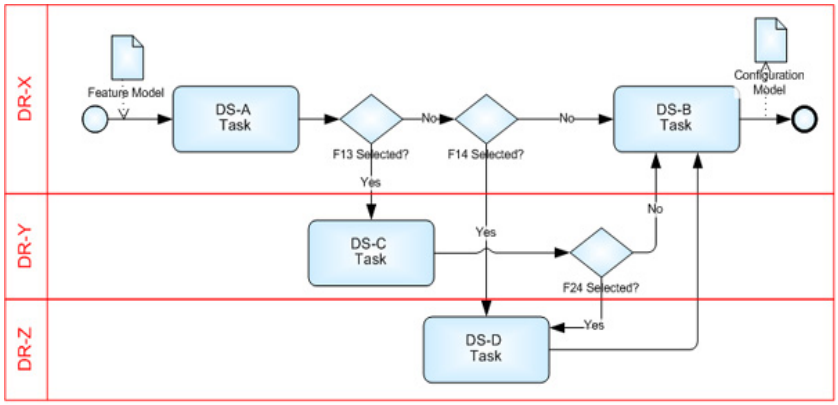

Fig.4. BPMN Process Model for Collaborative Product Configuration

Finally, note that decision set DS-A decisions precedes all others as the set is in the same tree of decisions as the others yet in a higher level in the feature model hierarchy. Fig. 4 illustrates an output BPMN process model representing a collaborative and coordinated product derivation process as the user has indicated the decision set's precedence list as follows: {DS-A, [DS-C], [DS-D], DS-B}. As BPMN

[3] The user interface for conflict resolution is still under development. Currently, precedence is defined by selecting a valid precedence list.

models can be mapped to BPEL executable models [17] we expect produced process models to be fully executable by BPEL engines.

4 Research to Date

We started our research studying the use of process languages in the context of object-oriented application frameworks. In particular, we ran and reported a case study on the use of RDL [10] to describe the instantiation steps of the REMF framework [11]. We then proposed extensions to the RDL process language to support aspect-oriented frameworks [12]. However, motivated by the applicability of our background in a more advantageous context, i.e., software configuration, and encouraged by preliminary successful results on staged configuration [2], we decided to concentrate our efforts on enabling collaborative product configuration scenarios. More specifically, we developed an approach to enable collaborative and coordinated product configuration as shown in this paper. Currently, a preliminary version of the algorithm to derive process models from annotated feature models have been developed in Java along with data structures to represent feature models, configuration models, decision sets, decision roles, and process models. The immediate goal was to produce a simple tool to assess our approach through case studies. The rules for defining decision sets are the same as those presented in this paper though they may be subject of change to reflect future work. We are now developing a new version of our tool to provide a more elaborated user interface and to allow process models to be exported to different formals, e.g. BPEL [15] models.

5 Conclusion and Future Work

In this paper we presented our research on product configuration. The research proposed an approach to foster a collaborative and coordinated product configuration process. In the approach, feature models were decorated with decision sets and decision roles and then transformed into process models that may be executed by process engines. Important coordination issues were discussed including feature interaction, decision conflicts and decision propagation.

Future works include the i) specification of a metal-model for validating annotated feature models; ii) support for optimistic conflict resolution strategies; iii) support for complex feature dependencies (e.g. A requires X or Y xor Z); iv) embedment of the approach into existing product line methods as those mentioned in [21]; v) enhancements to the support tool including its conversion to an Eclipse [19] plug-in, the specification of a user interface for decision conflict resolution, and the development of a visual editor for drawing and annotating feature models (possibly by extending existing tools [3]); vi) run various case studies to assess our approach, in special its scalability.

6 References

1. Czarnecki, K., Eisenecker, U.: *Generative Programming: Methods, Tools, and Applications*, Addison-Wesley, 2000, ISBN 0-201-30977-7.
2. Czarnecki, K., Helsen, S., Eisenecker, U.: *Staged Configuration through Specialization and Multi-Level Configuration of Feature Models*, Software Process Improvement and Practice, 10(2), 2005.
3. Antkiewicz, M., Czarnecki, K.: *FeaturePlugIn: Feature Modeling Plug-In for Eclipse*, OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, 2004.
4. Sochos, P., Riebisch, M., Philippow, I.: *The Feature-Architecture Mapping (FArM) Method for Feature-Oriented Development of Software Product Lines*, ECBS, pp. 308-318, 13th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems (ECBS'06), 2006.
5. Kang, K., Cohen, S., Hess, J., Novak, W., Peterson, A.: *Feature-oriented domain analysis (FODA) feasibility study*, SEI, CMU, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.
6. Kang, K.C., Kim, S., Lee, J., Kim, K., Shin, E., Huh, M.: *FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures*. Annals of Software Engineering, 5 (1998) 143-168
7. Deelstra, S., Sinnema, M., Bosch, J., *A Product Derivation Framework for Software Product Families*, Lecture Notes in Computer Science, Vol. 3014, 2004, p. 473 – 484
8. Griss, M. L: *Implementing Product line Features with Component Reuse*, in Proceedings of 6th International Conference on Software Reuse, Vienna, Austria, June 2000.
9. Gurp, J. V., Bosch, J., Svahnberg, M.: *On the Notion of Variability in Software Product Lines*, wicsa, p. 45, Working IEEE/IFIP Conference on Software Architecture (WISCA'01), 2001.
10. Oliveira, T. C., Alencar, P. S., Filho, I. M., de Lucena, C. J., and Cowan, D. D. 2004. *Software Process Representation and Analysis for Framework Instantiation*. IEEE Trans. Softw. Eng. 30, 3 (Mar. 2004), 145-159.
11. Mendonca, M., Alencar, P. S., Oliveira, T. C., and Cowan, D. D.: *Assisting Framework Instantiation: Enhancements to Process-Language-based Approaches*, Technical Report CS-2005-025, School of Computer Science, University of Waterloo, Sept 2005.
12. Mendonca, M., Alencar, P. S., Oliveira T. C., and Cowan, D. D.: *Assisting Aspect-Oriented Framework Instantiation: Towards Modeling, Transformation and Tool Support*, OOPSLA Companion, 2005, San Diego, US.
13. Henderson, P.: *Software Processes are Business Processes too*, Third International Conference on the Software Process, IEEE Comp. Soc. Press, Reston, USA, 1994.
14. Osterweil, L.: *Software Processes are Software too*. In Proceedings of the Ninth International Conference on Software Engineering. IEEE Computer Society, Washington, DC, 1987, pp. 2-13.
15. BPEL: *Business Process Execution Language for Web Services*
    Internet site: http://www-128.ibm.com/developerworks/library/specification/ws-bpel/
16. BPMN: *Business Process Modeling Notation*
    Internet site: http://www.bpmn.org/index.htm
17. White, S. A.: *Mapping BPMN to BPEL Example*, IBM Corporation
    http://www.bpmn.org/Documents/Mapping%20BPMN%20to%20BPEL%20Example.pdf
18. Calder, M., Kolberg, M., Magill, M.H.; Rei-Marganiec, S.: *Feature Interaction A Critical Review and Considered Forecast*. Elsevier: Computer Networks, Vol. 41/1 (2003)
19. Eclipse Platform: http://www.eclipse.org/
20. Krueger, C.: *Software Product Lines web site* (www.softwareproductlines.com)
21. Matinlassi, M.: *Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA*, Software Engineering, 2004. ICSE 2004. pp. 127- 136.

5 Karen Cortes Verdin, Cuauhtemoc Lemus Olalde
Aspect Oriented Product Line Architecture (AOPLA)

Aspect-Oriented Product Line Architecture (AOPLA)

Karen Cortes Verdin[1], Cuauhtemoc Lemus Olalde

Computer Science Department
Center for Research in Mathematics (CIMAT, A.C.)
Calle Jalisco S/N
Col. Mineral de Valenciana Guanajuato, Gto. 36240, Mexico
karen@cimat.mx, clemola@cimat.mx

Abstract. Among a Software Product Line's (SPL) core assets, the architecture is the most important. The Product Line Architecture (PLA) encompasses the Product Line (PL) attributes, and the capability of deriving the PL's specific products. Therefore, the definition of the PLA is a key activity for organizations following a SPL approach. In addition to this, the PLA allows the satisfaction of the market's present and future needs. Aspect-Oriented Software Development (AOSD) strives for a proper separation of concerns in software development, leading to software easy to maintain, comprehend, customize, reuse, and evolve. If an Aspect-Oriented approach is followed in the definition of a PLA, a proper identification, separation, and modeling of PLA's concerns will be obtained. This paper presents the proposal for the development of an Aspect-Oriented approach for the definition of a Product Line Architecture.

1 Introduction

A Software Product Line (SPL) is a "set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way" [1]. Software Product Line Engineering attempts to capitalize reuse by setting a framework for planning, development and management of core assets. Among such core assets, the Product Line Architecture (PLA) is perhaps the most valuable since it enables cost-effective development of PL products, models commonality, and supports SPL evolution.

Aspect-Oriented approaches seek to apply the separation of concerns principle in order to obtain software which is simpler to evolve, maintain, comprehend, customize, and reuse. Such separation of concerns can be applied from the early phases of software development. Concerns in these phases crosscut requirements' and/or architecture design's artifacts. These concerns are known as Early Aspects.

A PLA encompasses interesting challenges in addition to those of a software architecture. When dealing with a PLA, in addition to the quality attributes of the

[1] First year Ph. D. student.

products, the PLA has to exhibit its own quality attributes, The PL-specific quality attributes. Furthermore, the PLA should support commonality and variability, and it also has to be generic enough to support the realization of the planned products within the PL. This paper presents the doctoral research proposal for the development of a methodology for obtaining an Aspect-Oriented Product Line Architecture (AOPLA). This work proposes that following an Early Aspects approach for the PL architecting process will allow to obtain a generic, evolvable, and aspect-oriented PLA which properly supports commonality and variability. It is also proposed, that such approach should consider the PL-specific quality attributes as well. This proposal was developed during the first year of doctoral studies. This paper is organized as follows: the second section gives an introduction to AOSD. The third section describes the PL Quality Model, which will be the base quality model for the proposed approach. Next, the problem statement and importance of the research proposal are described. The fifth section describes the future work to be performed.

2 Aspect-Oriented Software Development

Separation of concerns is a key guiding principle in software development. It provides for software simpler evolution, maintainability, comprehensibility, customizability and reuse [2]. Aspect-oriented approaches strive for such separation of concerns, being programming approaches (Aspect-Oriented Programming or AOP) the first ones to emerge (vgr. AspectJ and Hyper/J).

Aspect-Oriented Software Development seeks to bring Aspect-Oriented (AO) principles to the software development process. AOSD provides the means for the identification, modularization, representation and composition of crosscutting concerns. According to Bergmans and Aksit crosscutting concerns or aspects are "those concerns that cannot be expressed as separate modules. . . resulting in implementations scattered over multiple operations" [3].

Separation of concerns can be applied from the very beginning of the software development process: in requirements engineering and architecture design phases. A concern in these early phases can crosscut a requirements' or architecture design's artifact (such as a requirements specification document,. a use case model, a sequence diagram, a module view, etc.). This kind of concerns are known as Early Aspects (EA) [5] [6].

In the same way as in AOP, EA must be properly managed in order to "establish critical trade-offs early on in the software life cycle" [6]. EA can span development activities, including implementation. Benefits from managing EA are:

- "increase consistency of requirements and architecture design with each other and with implementation;
- provide rationale and traceability for aspects across life-cycle activities; and

- help ensure that crosscutting concerns evident in a system's problem domain or solution space are captured as aspects in the implementation." [4]

AOSD is rather new, and approaches of AO analysis and design are also new. According to Chitchyan et al [4], any AO analysis and design approach should support the following criteria:

- Evolvability. The ease of changing the artifacts for an existing requirement, architecture, or design, or removal of addition of artifacts.
- Composability. The ability to compose artifacts and, consequently, to view and understand the complete set of artifacts and their interrelationships, as well as perceive them as a whole from the integrated artifacts.
- Traceability. This criterion records and communicates where the requirements come from and if they can be found in the architecture and design artifacts.
- Scalability. For an AO approach to be scalable it means that "it should be equally well suited for small and large projects "[4].

According to Chitchyan et al [4] these criteria can be "further refined for requirements, architecture, and design approaches as well as augmented with additional criteria". The above criteria will serve during the research work as a guide for the design of the proposed approach and also as part of the evaluation framework for the results to be obtained.

Aspect-Oriented Requirements Engineering attempts to provide "a systematic means for the identification, modularization, representation and composition of crosscutting properties, both functional and non-functional ones, during requirements engineering" [7]. An aspect at the requirements level is "a broadly scoped property, represented by a single requirement or a coherent set of requirements, that affects multiple other requirements in the system so that:

- It may constrain the specified behavior of the affected requirements.
- It may influence the affected requirements in order to alter their specified behavior" [7].

In architecture design, aspects can crosscut several architectural elements. Current approaches for architecture design do not differentiate conventional architectural concerns from architectural concerns crosscutting multiple architectural elements or components (conventional architectural concerns are those that can be localized using architectural abstractions). It is important to note that the crosscutting property of architectural aspects is inherent to architecture design. If such aspects are overlooked during architecture design, they may lead to tangled code, and therefore compromise the system's specified quality factors. Therefore, approaches for AOSA should identify architectural aspects as well as their related tangled components. The goal of this research is to provide an approach to solve such problem.

3 Quality Model from CAFÉ project

This section presents the PL Quality Model from CAFÉ project [8]. This model is taken as basis for the quality model in the proposed AOPLA approach herein. The CAFÉ project introduces a product family approach in an organization. The approach taken by CAFÉ project for PLA was to define a reference quality model for the PLA, from which product-specific architectures could be derived. The goal was to "provide quality-centric approaches to the analysis and the designs of product family architectures" [8]. One of the main outcomes of this project was the PL Quality Model. The model was defined as a means of specifying the quality requirements for the PL and, therefore, as a basis for evaluating the PLA.

The quality model is a free hierarchical decomposition of quality attributes, and there are no different names for different decomposition levels. A quality attribute is associated with a number of metrics that can be used to measure the degree to which a PLA satisfies a quality requirement. The model is expressed as a quality metamodel related to a pattern metamodel. The relationship between both metamodels is achieved by means of abstract scenarios. Furthermore, the quality metamodel includes a decision model that captures the variation points associated with the various elements within the model. The PL-specific quality attributes identified in the PL Quality Model are: "

1. Variability. The capability of an asset to contain common and varying parts thereby covering aspects of different product line members.
2. Derivability. Capability to derive a concrete, product-specific asset from a generic core asset.
3. Reusability. Capability to reuse and existing asset for different product line members.
4. Rateability. Capability to estimate the worth of a core asset.
5. Integrability. Capability to integrate a system-specific asset into the PL infrastructure.
6. Correctness. Extent to which an asset satisfies its specification and fulfils the product line's mission objectives.
7. Evolvability. Capability of an asset to evolve over time thereby dealing with growing complexity and demand as well as continuous change.
8. Manageability. Capability to manage core assets and asset configurations.
9. Maintainability. Capability of an asset to be modified" [9].

This model has been selected as the quality model for the proposed AOPLA approach because it provides a base for the consideration of quality attributes in the development of a PLA. The benefits of using this model are that it:

1. Defines the quality attributes that a PL must fulfill, and that, in consequence, a PLA must fulfill too.

2. Associates a set of metrics to the quality attributes.
3. Provides relationships between quality attributes and architectural patterns.
4. Can be used as basis for developing customized architectural views.
5. Provides process patterns that give proven solutions to recurring problems. Process patterns describe the solution in terms of a sequence of activities to be performed.
6. Provides for PLA evaluation, since the model includes the definition of scenarios associated to quality attributes.
7. Includes a decision model that captures the variation points associated with the various elements within the model.

As opposed to other quality models (McCall, ISO/IEC 9126, and IEEE Std. 1061), the PL Quality Model encompasses the required knowledge and guidelines for considering quality attributes for the development of the PLA.

4 Problem statement and importance of the research.

This section describes the problem that the research proposal tries to solve, as well as its importance. Software Product Line Engineering provides a new approach to alleviate current software development problems. Among a PL's core assets, the PLA is the most important. The PLA must:

1. fulfill the specified products' quality attributes,
2. be generic enough to generate the PL products,
3. support commonality and variability, and
4. fulfill the PL-specific quality attributes.

Current approaches for PLA do not show evidence of having all of the above characteristics [10] [11] [12] [13] [14] [15]. They neither make an explicit distinction between architectural concerns that can be localized using architectural abstractions, and architectural concerns that crosscut architectural components. This can lead to tangled code during implementation, with the obvious consequences of loss of maintainability, integrability, manageability, and evolvability for the PL. This research proposes the development of an Early Aspects approach for obtaining an evolvable, and generic PLA model. The PLA model obtained must comply with the PLA's desired characteristics (listed in section 3). This problem can be divided in the following subproblems:

1. The design of a concern development process that includes early identification of architectural crosscutting concerns.
2. Representation of early aspects in such a way that they can be easily incorporated into the PL architecting approach.

3. The proper modularization of aspects during the PL architecting process.
4. The definition of mechanism for concern composition for the derivation of a product-specific architecture.
5. Representation and description of an Aspect-Oriented Product Line Architecture (AOPLA) in compliance with current PLA description approaches: use of UML, view-based, and IEEE Std. 1471 compliant.
6. Early in the process, the proper consideration of PL-specific quality attributes.

Based on PLA approaches previously investigated (cited in previous paragraphs), no current PLA approach clearly and explicitly follows an AO strategy. In consequence, it can be concluded that above issues have not been addressed. These issues are sought to be solved with the development of the proposed AOPL model and architecting approach. The following are the research questions that have been stated for the research work. These research questions will serve, as well, as validation framework of the results:

1. Does an Aspect-Oriented PLA properly support commonality and variability?
2. Do Aspect-Oriented principles provide the necessary mechanisms for the proper consideration of the PL-specific quality attributes during the PL architecting process so that they are represented in the resulting PLA?
3. By following an Aspect-Oriented architecting strategy and identifying evolvability as one of the crosscutting concerns for the PLA, will this quality attribute be modeled and represented in a PLA so that it can be derived to a product-specific architecture?
4. Is an Aspect-Oriented PLA generic enough to allow for the derivation of product-specific architectures?

The benefits to be obtained with the current research will be:

1. The proper separation of concerns since early in the development process.
2. The adequate consideration of concerns during the architecting process.
3. The definition of a evolvable PLA model, encompassing:
    i. PL-specific quality attributes,
    ii. commonality and variability support,
    iii. generality, and
    iv. aspect modularization
4. The definition of a mechanism for obtaining a Aspect-Oriented PLA (AOPLA) with the corresponding models and mechanisms for:
    i. composing concerns,

ii. deriving a product-specific architecture,

iii. with support for PL evolution, and

iv. evaluation of the PLA

5. The definition of a framework for describing an AOPLA.

This research will be useful for all organizations seeking to adopt a PL revolutionary approach and wishing their investment to be adaptable to market changes.

5 Future work

This section describes the work to be done. It must be noted that the points in this section pertain to the thesis research work that has not initiated yet. Therefore, they require further investigation.

The objective of the research is to design an AOPL architecting approach to obtain an evolvable, generic PLA. This PLA must fulfill the PL-specific quality attributes (listed in section 3) and also provide support for commonality and variability. To attain this objective, the following goals (and tasks) are specified: To

1. Define a generic PL evolvability quality model. The PL Quality Model will serve as basis for the definition of the evolvability model. The definition of this model will encompass the definition of evolvability metrics, the decision model, and abstract and concrete scenarios. The concrete scenarios will allow the identification of architectural patterns and tactics. The definition of models for other PL-specific quality attributes has been considered as part of the research work, although which attributes are to be selected has still to be decided.

2. Design a concern development process. It is important to identify all kinds of concerns from the beginning of the development process. Some of these concerns will be crosscutting and will therefore be modeled as aspects. Those concerns not modeled as aspects could be considered for future versions of the application. It has been thought that this concern development process should begin from the scope definition phase and continue through the requirements engineering phase up to the architecture definition phase. This concern development process goes hand-in-hand with the strategy for aspect modularization described below.

3. Develop a strategy for aspect modularization. Following the early identifcation of crosscutting concerns, this strategy will take such concerns and model them as aspects. The identification should begin with the domain models developed during the scope definition phase. There is already ongoing work on Aspect-Oriented Domain Analysis at the University of Twente [16] which still has to be investigated in order to determine its contribution to the proposed approach herein. In addition, a few approaches for Aspect-Oriented Software Architecture design already make extensions to domain analysis

models for aspects identification and modeling. AOGA [17] is one of such methods.

4. Design a mechanism for concern composition specification. Once aspects have been properly modeled and specified, they must be integrated or composed into the architecture. Therefore a means of specifying composition is required. This mechanism goes hand-in-hand with the strategy for aspect modularization previously explained.

5. Develop a strategy for commonality and variability analysis. Variability is a key property of domain artifacts, in this case of the architecture. So far, no specific approach has been devised to the definition of this strategy. However, research will be performed on current approaches, depending on available information.

6. Define the feature, architecture decision and join point models. Feature models are the most common techniques used in defining the PL scope and, they are also employed as a means of identifying and modeling aspects. Because of this feature models will form part of the proposed approach. These models will begin to be built during scope definition and further detailed during the requirements engineering phase. Up to this point it has not been determined which approach to follow for the PL requirements engineering phase but it is clear that the approach will have to consider feature models. The architecture decision model, in turn, will be built during the definition of the evolvability quality model. A joint point model, in turn, is a model that specifies the points in the architecture with which the aspects coordinate. Aspect modularization implies aspect composition and this composition is based on the join point model.

7. Develop a framework for describing the AOPLA. In [18] Gomaa has already extended UML to address software product lines modeling. These extensions include commonality and variability modeling as well as feature modeling. Besides, Jacobson and Ng in [19] have already approached aspect modeling with UML. Jacobson and Ng employ use cases to model and separate crosscutting concerns (a use case is a crosscutting concern). Their work makes use of extensions to use cases and extension points that are propagated from use cases to all the components that realize the use case. An extension point corresponds to a join point and the set of extension points belong to a join point model. The intention is to integrate both approaches for the description of a AOPL architecture. In addition, since use cases comprise scenarios there is a possible link with the scenarios of the PL Quality Model. This requires further investigation.

8. Define the criteria for the use of architectural patterns and tactics. The definition of the evolvability quality model will allow to define criteria for using architectural patterns and tactics for the achievement of this quality attribute.

9. Define the derivation of a product-specific architecture from the AOPLA. This will encompass the definition of the aspect weaving process. Weaving

is the process of composing aspects to produce the desired architecture. The weaving process is based on the join point model

6 Conclusions

An approach for the development of an Aspect-Oriented Product Line Architecture has been proposed. It is expected that the application of AO principles will permit to obtain a generic, evolvable PLA architecture. The proposed approach considers, as well, the fulfillment of PL Quality attributes. What is sought by incorporating an AO approach, specifically an EA approach, into the design of a PLA, is the proper identification of crosscutting concerns, and their inclusion into a PLA.

The PL Quality Model from CAFÉ project has been proposed as the quality model for the approach. This model allows for the definition of quality attributes and associated metrics, architectural views and process patterns. It also will provide a framework for evaluating the resulting PLA.

The main expected benefits of following an EA approach are:

1. Consistency between requirements, architecture, and implementation.
2. Traceability of aspects across the life-cycle.
3. Proper identification and treatment of non-conventional architectural aspects.
4. Capture of domain's crosscutting concerns as aspects.
5. Support for PL evolution.

The future work that remains pertains to the thesis research work, which has also been described.

Karen Cortes Verdin is a professor at the School of Statistics and Informatics at the Universidad Veracruzana.

References

[1]. P. Clements, and L. Northrop, *Software Product Lines: Practices and Patterns*, Addison-Wesley, USA, 2001.

[2]. T. Elrad, R. E. Filman, A. Bader, "Aspect-oriented programming", *Communications of the ACM*, vol. 44, no. 10, October 2001, pp. 29-32.

[3]. L. Bergmans, M. Aksit, "Composing Crosscutting concerns using Composition Filters", *Communications of the ACM*, vol. 44, no. 10, October 2001, pp. 51-57.

[4]. Chitchyan R., Rashid A., Sawuer P., Garcia A., Pinto Alarcón M., Bakker J., B. Tekinerdogan., Clarke S., and A. Jackson, *Survey of Aspect-Oriented Analysis and Design Approaches* (AOSD-Europe-ULANC-9). Editor(s): R. Chitchyan, A. Rashid. Lancaster, UK, University of Lancaster, 2005.

[5]. E. Baniassad, P. C. Clements, J. Araujo, A. Moreira, A. Rashid, and B. Tekinerdogan, "Discovering Early Aspects", IEEE Software, Vol. 23, no. 1, January/February 2006, pp. 61-70.

[6]. http://www.early-aspects.net/

[7]. J. Araujo, E. Baniassad, P. C. Clements, a. Moreira, A. Rashid, and B. Tekinerdogan, *Early Aspects: The Current Landscape* (COMP-001-2005), Lancaster, UK, University of Lancaster, 2005.

[8]. http://www.esi.es/en/Projects/Cafe/cafe.html

[9]. M. Anastasopoulos, J. Bayer , *Product Family Specific Quality Attributes* (IESE Report No. 042.02/E, Version 1.0). Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering, 2002.

[10]. http://www.sei.cmu.edu/productlines/add_method.html

[11]. M. Anastasopoulos, J. Bayer , O. Flege, C. Gacek, *A Process For Product Line Architecture and Evaluation. PuLSE-DSSA – Version 2.0* (IESE Report No. 038.00/E, Version 1.0). Kaiserslautern, Germany: Fraunhofer Institut Experimentelles Software Engineering, 2000.

[12]. D. M. Weiss, and C. T. Robert Lai, *Software Product-Line Engineering. A Family-Based Software Development Process*, Addison-Wesley, 1999.

[13]. http://www.sei.cmu.edu/domain-engineering/FODA.html

[14]. M. Matinlassi, E. Niemelä , L. Dobrica, *Quality-driven architecture design and quality analisis method. A revolutionary initiation approach to a product line architecture* (VTT Report No. P456). Oulu, Finland: Technical Resear Centre of Finland (VTT), 2002.

[15]. C. Atkinson, I, Bayer, C. Bunse, E. Kamsties, O. Laitenberger, R. Laqua, D. Muthig, B. Paech, J. Wüst, and J. Zettel, Component-based Software Product-Line Engineering with UML, Addison-Wesley, 2002.

[16]. http://trese.cs.utwente.nl/taosad/ArchitecturalAspects/aspectual_domain_analysis.htm

[17]. U. Kulesza, A. Garcia, C. Lucena, *"Generating Aspect-Oriented Agent Architectures"*, Proceedings of the 3rd Workshop on Early Aspects (AOSD'2004), Lancaster, UK 2004.

[18]. H. Gomaa, *Designing Software Product Lines with UML: From Use cases to Pattern-based Software Architectures,* Addison-Wesley,USA 2004.

[19]. I. Jacobson, P. Ng, *Aspect-Oriented Software Development with Use Cases,* Addison-Wesley, USA 2005.

6 Uirá Kulesza, Carlos José Pereira de Lucena
An Aspect-Oriented Approach to Framework Development

An Aspect-Oriented Approach to Framework Development

Uirá Kulesza, Carlos José Pereira de Lucena

Computer Science Department, Software Engineering Laboratory
Pontifical Catholic University of Rio de Janeiro (PUC-Rio), Brazil
{uira, lucena}@inf.puc-rio.br

Abstract. In this work, we propose an approach which aims to improve the extensibility of object-oriented frameworks using aspect-oriented programming. Our approach proposes the definition of extension join points in the framework code, which can be extended by means of a set of extension aspects. These aspects are responsible to implement optional, alternative and integration features in the framework. Additionally, we also propose a generative model which allows to instantiate automatically the variabilities of a framework and its respective extension aspects.

Classification: PhD 3rd Year.

1 Introduction

Object-oriented (OO) frameworks [8] represent nowadays a common and important technology to implement software families and product lines. They enable modular, large-scale reuse by encapsulating one or more recurring concerns of a given domain, and by offering different variability and configuration options to the target applications. In the framework based development, applications are implemented by reusing the architecture defined by the frameworks and by extending their respective variation points or hot-spots [8]. Hence, the adoption of the framework technology brings in general significant productivity and quality in the development of applications. Besides their advantages, some researchers [5, 6, 17, 18] have recently described the inadequacy of OO mechanisms to address the modularization and composition of many framework features, such as, optional [5], alternative and crosscutting composition features [17, 18]. The limited modularity provided by the OO mechanisms brings difficulties to configure many framework features for specific needs, thus impeding the framework adaptation and reuse [5, 6, 17, 18] in different scenarios.

Aspect-oriented software development (AOSD) [9, 14] has been proposed as a technology which aims to offer enhanced mechanisms to modularize crosscutting concerns. Crosscutting concerns are concerns that often crosscut several modules in a software system. AOSD has been proposed as a technique for improving the separation of concerns in the construction of OO software, supporting improved reusability and ease of evolution. Recent work [1, 15, 16, 18, 19, 20, 22, 29] has explored the use of aspect-oriented (AO) techniques to enable the implementation of

flexible and customizable software family architectures. In these research works, aspects are used to modularize crosscutting variable (optional or alternative) and integration features.

In this work, we propose an approach which aims to improve the extensibility of object-oriented frameworks using aspect-oriented programming (AOP). Our approach proposes the definition of extension join points (EJPs) in the framework code, which can be extended by means of variability and integration aspects. These aspects are responsible to implement optional, alternative and integration features in the framework. Since the aspects can be automatically unplugged from the framework code, our approach makes easier to customize the framework to specific needs. Additionally, we also propose a generative model which allows to automatically instantiate specific customizations of an object-oriented framework with its respective extension aspects according to user needs.

The remainder of this paper is organized as follows. Section 2 presents background by detailing framework modularization problems addressed by our approach. Section 3 gives an overview of our approach for framework development with aspect-oriented programming based on the specification of EJPs. Section 4 describes the implementation of the JUnit framework using our framework development approach. Section 5 describes the generative model used to instantiate automatically the variabilities of a framework and its respective extension aspects. Section 6 discusses related work. Finally, Section 7 summarizes our contributions and provides directions for future work.

2 Modularization Problems in OO Frameworks

Despite the well-known benefits of OO frameworks in implementing program families, recent research has exposed the inadequacy of framework technology in modularizing features with particular properties, such as optional [5] and crosscutting composition [23, 24] features. In this section, we briefly revisit research work that describes the inadequacy of object-oriented mechanisms to modularize specific framework features. These issues hinder the framework instantiation process to meet specific user needs. As a result, framework reuse can become unmanageable or even impracticable. Next, we describe these two problems of framework feature modularization.

Modularizing Optional Framework Features. Batory et al [5] address the issues of the framework technique in modularizing optional features. An optional feature is a framework functionality that is not used in every framework instance. According to such research, developers typically deal with this problem either by implementing the optional feature in the code of concrete classes during the framework instantiation process, or by creating two different frameworks, one addressing the optional feature and the other one without it. As a result, many framework modules are replicated just for the sake of exposing optional features, thus leading to “overfeatured” frameworks [6], in which several instance-specific functionalities can be present.

By analyzing a number of available frameworks (such as JUnit and JHotDraw), we note that the most widespread practice in implementing framework optional features is the use of inheritance mechanisms to define additional behavior in the framework classes. In the JUnit framework, for example, inheritance relationships are used to define a specific kind of test case as well as additional and optional extensions to test cases and suites.

Crosscutting Feature Compositions in Frameworks Integration. Mattsson et al [23, 24] have analyzed the issues in integrating OO frameworks and proposed several OO solutions. Their research relates the composition of two frameworks to the composition of a new set of features (represented as a framework) in the structure of another framework. For example, suppose we need to extend the JUnit framework to send specific failures that occur to software developers. A specific test failure report could be send by e-mail to different software developers, every time a specific and critical failure happens. Imagine we have available an e-mail framework to support our implementation. The problem here is how we could implement this functionality in the JUnit framework. It involves the integration of the JUnit and the e-mail framework. This composition could be characterized as crosscutting since we are interested to send a failure report by e-mail during the execution of the tests.

Based on a case study [18] with feature compositions involving four OO frameworks of varying complexity and addressing concerns from distinct horizontal and vertical domains [7], we have concluded that the framework integration solutions presented by Mattson et al [23, 24] are invasive and bring several difficulties to the implementation, understanding, and maintenance of the framework composition code. Our analysis has shown that 6 out of 9 solutions described by those authors have poor modularity and a crosscutting nature, requiring invasive internal changes in the framework code.

3 An Approach to Extending OO Frameworks with Aspects

This section gives an overview of our framework development approach. Our approach deals with the framework modularization problems presented previously (Section 2) by using AOP and the notion of extension join points (EJPs). EJPs also support the disciplined specification of additional opportunities for framework extensions. Section 3.1 presents the proposed approach by describing the concept of EJPs. Section 3.2 describes different uses of aspects to improve framework extensibility. Section 3.3 presents the achieved benefits. Section 3.4 discourses about the EJP implementation in AspectJ language. Finally, Section 3.5 presents the application of the approach to the JUnit framework.

3.1 Extension Join Points

In our approach, an OO framework specifies and implements not only its common and variable behavior using OO classes, but it also exposes a set of extension join points (EJPs) which can be used to also extend its functionality. The idea of EJPs is

inspired by Sullivan et al's work [28, 11] on specification of crosscutting interfaces (XPIs). Similar to XPIs, EJPs establish a contract between the framework classes and a set of aspects extending the framework functionality. However, unlike XPIs, EJPs are adopted as a means to increase the framework variability and integrability. Thus, we propose to use the XPI concept in the context of framework development. EJPs can be used to three different purposes:

(i) to expose a set of framework events that can be used to notify or to facilitate a crosscutting integration with other software elements (such as, frameworks or components);

(ii) to offer predefined execution points spread and tangled in the framework into which the implementation of optional features can be included;

(iii) to expose a set of join points in the framework classes that can have different implementations of a crosscutting variable functionality.

In this context, EJPs document crosscutting extension points for software developers that are going to instantiate and evolve the framework. They can also be viewed as a set of constraints imposed on the whole space of available join points in the framework design, thereby promoting safe extension and reuse. A key characteristic of EJPs is that framework developers and users do not need to learn totally new abstractions to use them, as they can mostly be implemented using the mechanisms of AOP languages, such as AspectJ.

3.2 Framework Core and Extension Aspects

Our approach promotes framework development as a composition of a core structure and a set of extensions. A framework extension can be: (i) the implementation of optional or alternative framework features; or (ii) the integration with an additional component or framework. The composition between the framework core and the framework extensions is realized by different types of aspects. Each aspect defines a crosscutting composition with the framework by means of its exposed EJPs. Next, we describe the main elements of our approach:

(i) *framework core* – implements the mandatory functionality of a software family. Similar to a traditional OO framework, this core structure contains the frozen-spots that represent the common features of the software family and hot-spot classes that represent non-crosscutting variabilities from the domain addressed;

(ii) *aspects in the core* – implement and modularize existing crosscutting concerns or roles in the framework core. They represent the traditional use of AOP to simplify the understanding and evolution of the framework core;

(iii) *variability aspects* – implement optional or alternative features existing in the framework core. These elements extend the framework EJPs with any additional crosscutting behavior;

(iv) *integration aspects* – define crosscutting compositions between the framework core and other existing extensions, such as an API or an OO framework. These elements also rely on the EJPs specification to define their implementation.

Figure 1 shows the design of an OO framework with aspects following our approach. As we can see, both variability and integration aspects can only act in the EJPs provided by the framework and they must respect all the constraints defined by

them. This brings systematization to the framework extension and composition with other artifacts.

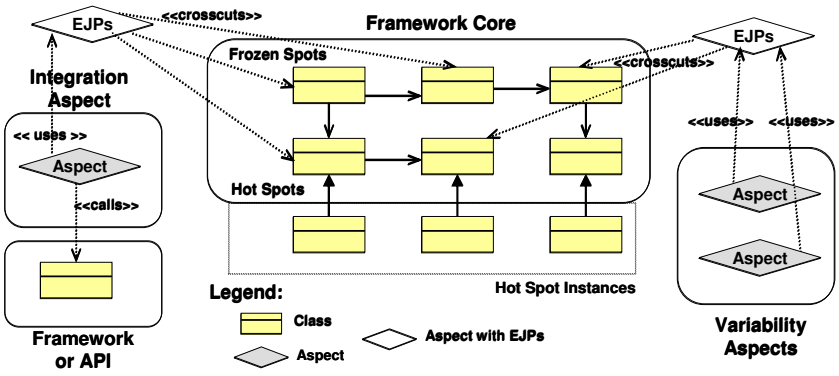

Figure 1. Elements of our Framework Development Approach

3.3 Benefits

Table 1 describes the benefits brought by each type of aspect in our framework development approach. It also indicates how the core, variability, and integration aspects address each of the modularization problems in framework development and evolution. As pointed out in the table, the use of internal framework aspects also has a positive impact on the framework variability and integrability. They facilitate the specification of EJPs because core aspects promote modularization of the internal class roles. Therefore, they offer additional join points to be exploited in extension scenarios.

3.4 EJPs Implementation

We have explored the use of AspectJ [3] language to specify the framework extension join points. The EJP codification in AspectJ language brings the following advantages to the framework extension process: (i) it enables the developer to expose a set of join points that are spread in the framework in a single aspect, that can be used to extend the framework functionality with integration and variability aspects; and (ii) it allows the representation of many constraints – that must be satisfied when extending those join points – in a way that they will not just be stated but they will be enforced during compilation and runtime.

Each EJP is represented by an aspect comprising a set of AspectJ pointcut descriptors that represents the set of extension join points of a framework. The EJP constraints which regulate the relationships between the framework, EJPs and extension aspects (mentioned in Sections 3.1 and 3.2) are represented, in our approach, by separate aspects. We are defining a methodology to specify different

kinds of contracts which define the constraints between the elements of our approach. We have classified these contracts in the following categories: (i) *framework internal contracts* - contracts between the framework and its EJPs whose purpose is to assure that framework refactorings and evolution do not affect the functionality of its extension aspects; and (ii) *framework extension contracts* - contracts between the EJPs and its extension aspects to assure that each extension aspect respects constraints and invariants of the framework.

We have used different mechanisms of AspectJ to implement the EJPs contracts. AspectJ offers both static and dynamic mechanisms that can be used to specify them. When choosing mechanisms for each contract type, we prefer static mechanisms (such as declare parents, declare error and declare warning constructions) to dynamic ones (such as advice execution). This allows to verify many of the contracts before the complete installation of the software. Due to space limitation, we do not present in detail in this paper our categorization of contracts as well as their implementation in AspectJ. For a complete description, please refer to [15].

Table 1. Framework Development Approach Elements

Approach Element	Benefits	Modularization Problem Addressed
Aspects in the Core	- Simplify the understanding and evolution of the framework core • Modularize existing crosscutting concerns or roles in the framework core. - Facilitate the design of EJPs	Crosscutting Roles and Concerns
Extension Join Points	- Systematize the framework extension and composition by promoting safe framework reuse • Enable the composition between framework core and extensions. • Encapsulate the framework and exposes only proper join points.	Tight Coupling between Core and Extensions
Variability Aspects	- Facilitate the framework reuse and extension. • Modularize optional and alternative framework features. • Make it possible to plug and unplug optional or alternative features.	Optional or Alternative Features
Integration Aspects	- Facilitate the framework reuse and composition. • Modularize the framework composition with other extensions. • Make it possible to plug and unplug crosscutting framework composition.	Crosscutting Framework Compositions

3.5 Case Studies

Our approach has emerged from our experience in different domains, through a process of continuous interaction and refinement between case studies and the approach itself. In this context, the approach was employed in the development of frameworks in the following domains: (i) JUnit testing framework [15, 16]; (ii) J2ME games [1, 15, 16]; (iii) multi-agent systems (MASs) [19]; and (iv) measurement support for product quality control [18]. Next section details the JUnit framework case study. For a more detailed description of the implementation of EJPs and framework extensions for our case studies, please refer to [15, 16].

4 JUnit Case Study

This section illustrates the use of the proposed approach in the context of the JUnit framework. Although JUnit presents a well-modularized architecture, we have found some modularization problems [21] hindering its future extension/evolution. In the context of other complex and large-scale frameworks, these problems can cause architecture erosion after a while. Due to space limitation, we have briefly mentioned JUnit problems in Section 2.

The main purpose of the JUnit framework is to allow the design, implementation and execution of the unit tests in Java applications. According to the JUnit framework, each unit test is responsible for exercising one class method in order to assure that it performs as expected. The JUnit main functionalities are: the definition of test cases or suites to be executed; the execution of a selected test case or suite; and the collection and presentation of the test results. However, different extensions (optional features) can be implemented to add new functionalities into the JUnit framework core. Some examples of simple extensions are the following: (i) enable JUnit to execute each test suite in a separate thread, and wait until all tests finish; (ii) enable JUnit to run each test repeatedly. In order to implement this extension we need to observe the event when each test method runs; and (iii) introduce some additional behavior before or after the test case or suite.

These extensions need to observe JUnit internal events, which are spread over JUnit classes. The modularization of these extensions using OO mechanisms is addressed by complex class hierarchies which bring difficulties to the understanding and evolution of the framework core and of each extension. In other words, such extensions are not well modularized in the OO design. In our approach, an EJP was used to expose such key events that are not adequately captured by the OO design and that are useful for crosscutting compositions scenarios. Figure 2 presents an EJP, called `TestExecutionEvents`, which exposes the following join points from the JUnit: (i) test case execution; (ii) test suite execution; and (iii) initialization of test runners. Some of these join points were discovered by checking them against these anticipated crosscutting extension scenarios. Based on this first set of discovered join points, we could foresee other relevant events that may be of interest when extending JUnit.

After codifying the `TestExecutionEvents` EJP, we can implement different variability aspects, as presented in Figure 2, to add the testing extensions into the

JUnit EJPs, such as: (i) to run test cases or test suites repeatedly (`RepeatAllTest` aspect); (ii) to execute them in separate threads (`ActiveTestSuite` aspect); and (iii) to introduce some additional behavior before or after the test case or suite (`TestCaseDecorator` and `TestSuiteDecorator` aspects). We reuse the join points exposed in the aspect `TestExecutionEvents` to implement each of them. It is also possible to codify aspects to affect just specific test cases or suites defined to test an application. Finally, the JUnit EJPs can also be used to compose it with other OO frameworks. Figure 2 shows, for example, the `MailNotification` integration aspect responsible for monitoring the test execution, building specific test reports and sending them by e-mail to specific developers. An email framework could be composed with the JUnit framework to provide that functionality by means of an integration aspect. The implementation of this functionality in the original OO design requires the introduction of invasive code in the `TestSuite` and `TestCase` classes in order to notify the classes responsible to implement and integrate it with the email framework.

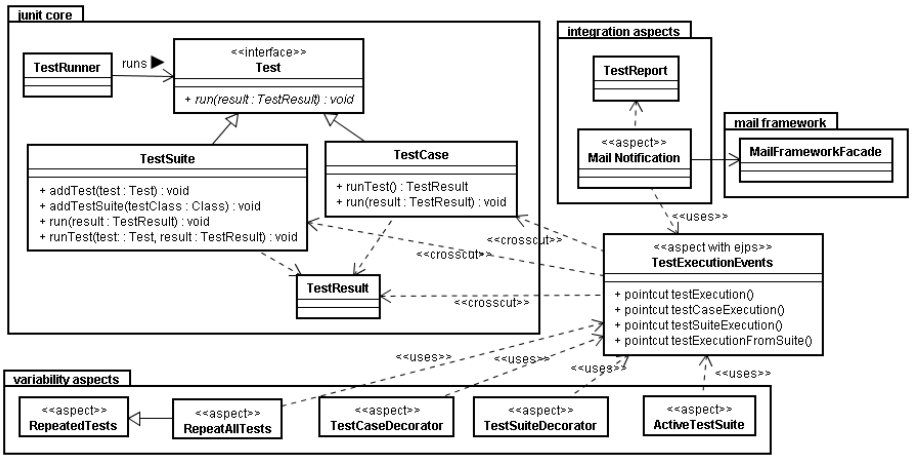

Figure 2. Elements of our Framework Development Approach

5 An AO Generative Model to Framework Instantiation

The result of the implementation of a software family or product line architecture using our framework development approach is a set of artifacts (classes, aspects, etc) which addresses the commonalities and variabilities for a specific domain. Many of the classes and aspects specified for the architecture will be instantiated only if they are necessary to implement an individual application or product. The manual selection and configuration of these elements can become a complex process which can make impracticable the use of the approach. To facilitate the instantiation of the framework and its aspect extensions, we proposed an aspect-oriented generative model [17, 20] which enables the automatic configuration of the framework based on requests from application developers.

Our AO generative model [17, 20] follows the general structure presented by Czarnecki and Eisenecker [7]. However, we propose the extension of that generative model to support the instantiation and customization of AO architectures. It allows configuring and generating specific crosscutting and non-crosscutting variabilities. Our generative model is composed by the following elements:

(I) *a feature model* – this model works as a configuration domain-specific language (DSL) responsible to specify and collect the features to be instantiated in the software family architecture. It is used to collect information to configure both the crosscutting and non-crosscutting variabilities. A set of crosscutting relationships between features is used to help the customization of aspects pointcuts.

(II) *an AO architecture* – it defines the main components of a software family architecture. This architecture defines a set of variabilities which need be customized to define a complete application. Crosscutting variabilities are implemented as aspects in this architecture. Each component of the architecture is specified as a set of classes, aspects and templates. The latter ones define elements that will be customized during the instantiation of the architecture. The implementation of the architecture is supported by our framework development approach based on EJPs (Section 3). Thus, our approach provide guidelines to implement these AO architectures by means of a base OO framework and a set of aspects which define optional and alternative crosscutting features existing in the OO framework;

(III) *a configuration model* – it specifies the mapping between the features existing in the crosscutting feature model and the components (or their respective sub-elements, such as, class, aspect or templates) of the AO architecture. The configuration model is used to support the decision of which components must be instantiated and what customizations must be realized in those components considering a specific application.

There are several activities involved in the process of development of the elements of our generative approach [17, 20]. These activities are organized under the perspectives of domain implementation[1] and application engineering. The domain implementation involves: (i) the implementation of the software family architecture following the guidelines of our framework extension approach; (ii) the representation of the architecture variabilities in a feature model; and (iii) the specification of a configuration model which defines the mapping between features and architecture elements (classes, aspects, etc). In application engineering, developers request an instance of the AO architecture by specifying all desired variabilities. This request is composed of two activities: (i) choice of variabilities in a feature model instance; and (ii) choice of valid crosscutting relationships between features. This latter step is used to enable the customization of aspect pointcuts. A tool uses the information collected by these steps and the configuration model to generate an instance of the AO architecture.

[1] This work only covers the domain implementation phase [7] from domain engineering. The domain analysis and design are out of scope of the thesis.

6 Related Work

Feature oriented approaches (FOAs) have been proposed [27] to deal with the encapsulation of program features that can be used to extend the functionality of existing base program. Batory et al [5] argue the advantages that feature-oriented approaches have over OO frameworks to design and implement product-lines. Mezini and Ostermann [25] have identified that FOAs are only capable of modularizing hierarchical features, providing no support for the specification of crosscutting features. These researchers propose CaesarJ [26], an AO language that combines ideas from both AspectJ and FOAs, to provide a better support to manage variability in product-lines. The work of those authors has a direct relation to our work, since we believe that the design of product-line architectures may benefit from the composition and extension of different frameworks using integration and variability aspects.

Zhang and Jacobsen [29] propose the Horizontal Decomposition method (HD), a set of principles guiding the definition of functionally coherent core architecture and customizations of it. The core is customized with aspects implementing orthogonal functionality. Unlike our approach, which uses EJPs to achieve bi-directional decoupling of the core from its extensions in the framework context, HD has a principle explicitly embracing obliviousness, whereby the core should be completely unaware on the aspects.

Our concept of EJPs is inspired by Sullivan et al's work [28] on specification of crosscutting interfaces (XPIs). XPIs abstract crosscutting behavior, isolating aspect design from base code design and vice-versa. Continuing this work, Griswold et al show how to represent XPIs as syntactic constructs [11]. EJPs play a similar role to XPIs, but specifically in the context of framework development, by exposing a set of framework events for notification and crosscutting composition, and by offering predefined execution points for the implementation of optional and alternative features. In the specification of the semantic part of EJPs, however, we have defined a different methodology to specify the constraints which regulate the relationships between the framework, EJPs and extension aspects [15].

Framed Aspects [22] is an approach that combines AOP and frame technology to facilitate the automatic customization of applications. Aspects are used to modularize crosscutting concerns and frames enables the parameterization and configuration of variabilities encountered in aspects. The role played by frame technology in the Framed Aspects is addressed in our approach by the use of aspect code templates. Our code templates can be customized by a code generator (Section 5) based on information collected by feature models. Framed Aspect approach does not propose explicitly guidelines to the modularization of framework features using AOP, as we do using EJPs and extension aspects.

7 Conclusions and Future Work

Object-oriented frameworks represent nowadays a relevant technology to implement software family architectures since they can specify some part or the entire SPL architecture by offering different variability and configuration options. However, several obstacles have been identified which bring difficulties to the framework reuse and composition when implementing software family architectures. In this work, we

proposed an approach for the design and implementation of OO frameworks with aspects which aims to improve their extensibility and integrability (Section 3). Our approach addresses the modular implementation of framework optional features and enables framework crosscutting composition with other OO extensions. The exposition of only specific framework extension join points (EJPs) brings systematization to the process of extension and composition of the framework. EJPs enable the framework systematic extension by means of extension aspects. We have already developed some case studies that demonstrate the benefits brought by the approach. Additionally, we have defined an aspect-oriented generative model which facilitates the instantiation and configuration of the framework and its respective extension aspects (Section 5).

This paper described a doctoral work in progress. Follow we present the next activities that are being developed as part of this work:

(i) we will continue the evaluation of the approach in the development and refactoring of object-oriented frameworks. In these new case studies, we plan to realize quantitative studies [10] to compare the approach against the use of OO techniques and other approaches [22, 25, 27, 29] with respect to traditional software metrics. These new case studies will allow to derive more conclusive results and data about the benefits from our approach. In these new case studies, we also intend to investigate how our approach can deal with feature interaction problems [28];

(ii) we also intend to derive a more systematic implementation method which offers more detailed steps and guidelines to the implementation and instantiation of extensible OO frameworks with aspects using our approach;

(iii) we are currently implementing a tool, as an Eclipse plug-in, which supports the generative model presented in the paper based on existing technologies [2, 3, 4].

(iv) finally, we plan to explore the extension of current domain analysis and design methods [7] to support the early modeling of extension join points and framework extension aspects. In particular, we are interested to investigate the synergy between the use case extensions points proposed by Jacobson [27] and our extension join points during the proactive development of product lines architectures.

Acknowledgements. This research is partially sponsored by FAPERJ (grant No. E-26/151.493/2005) and CNPq (grants No. 140252/03-7). The authors are also supported by the ESSMA Project under grant 552068/02-0.

References

1. V. Alves, P. Matos, L. Cole, P. Borba, G. Ramalho. “Extracting and Evolving Mobile Games Product Lines”. Proceedings of SPLC'05, LNCS 3714, pp. 70-81, September 2005.
2. M. Antkiewicz and K. Czarnecki. FeaturePlugin: Feature modeling plug-in for Eclipse, OOPSLA’04 Eclipse Technology eXchange (ETX) Workshop, 2004.
3. AspectJ Team. The AspectJ Programming Guide. http://eclipse.org/aspectj/.
4. F. Budinsky, et al. Eclipse Modeling Framework. Addison-Wesley, 2004.
5. D. Batory, R. Cardone, and Y. Smaragdakis, Object-Oriented Frameworks and Product-Lines. 1st Software Product-Line Conference (SPLC), pp. 227-248, Denver, August 1999.
6. W. Codenie, et al. “From Custom Applications to Domain-Specific Frameworks”, Communications of the ACM, 40(10), October1997.
7. K. Czarnecki, U. Eisenecker. Generative Programming: Methods, Tools, and Applications, Addison-Wesley,2000.

8. M. Fayad, D. Schmidt, R. Johnson. Building Application Frameworks: Object-Oriented Foundations of Framework Design. John Wiley & Sons, September 1999.
9. R. Filman, T. Elrad, S. Clarke, M. Aksit. Aspect-Oriented Software Development. Addison-Wesley, 2005.
10. A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. Staa. Modularizing Design Patterns with Aspects: A Quantitative Study. Proc. 4th Intl. Conference on Aspect-Oriented Software Development, Chicago USA, March 2005.
11. W. Griswold, et al, "Modular Software Design with Crosscutting Interfaces", IEEE Software, Special Issue on Aspect-Oriented Programming, January 2006.
12. M. Jackson, P. Zave, "Distributed Feature Composition: A Virtual Architecture For Telecommunications Services", IEEE Transactions on Software Engineering, October 1998.
13. I. Jacobson. "Use Cases and Aspects – Working Seamlessly Together". Journal of Object Technology, pp. 7-28, Vol. 2, Number 4, August 2003.
14. G. Kiczales, et al. Aspect-Oriented Programming. Proc. of ECOOP'97, Finland, 1997.
15. U. Kulesza, R. Coelho, A. Neto, V. Alves, A. Garcia, C. Lucena, A. von Staa, P. Borba. "Implementing Framework Crosscutting Extensions with EJPs and AspectJ", Proceedings of ACM SIGSoft XX Brazilian Symposium on Software Engineering (SBES´2006), Florianópolis, Brazil, October 2006.
16. U. Kulesza, V. Alves, A. Garcia, C. Lucena, P. Borba. Improving Extensibility of Object-Oriented Frameworks with Aspect-Oriented Programming, Proceedings of ICSR'2006, Springer Verlag, LNCS 4038, pp. 231-245, Torino, Italy, June 2006..
17. U. Kulesza, A. Garcia, F. Bleasby, C. Lucena. "Instantiating and Customizing Product Line Architectures using Aspects and Crosscutting Feature Models". Proceedings of the Workshop on Early Aspects, OOPSLA'2005, October 2005, San Diego.
18. U. Kulesza, A. Garcia, C. Lucena. "Composing Object-Oriented Frameworks with Aspect-Oriented Programming", Technical Report, PUC-Rio, Brazil, April 2006.
19. U. Kulesza, A. Garcia, C. Lucena, A. von Staa. "Integrating Generative and Aspect-Oriented Technologies", Proceedings of ACM SIGSoft Brazilian Symposium on Software Engineering (SBES'2004), pp. 130-146, Brasilia, Brazil, October 2006.
20. U. Kulesza, C. Lucena, P. Alencar, A. Garcia. "Customizing Aspect-Oriented Variabilites using Generative Techniques". Proceedings of International Conference on Software Engineering and Knowledge Engineering (SEKE'06), pp. 17-22, San Francisco, July 2006.
21. U. Kulesza, C. Sant´Anna, C. Lucena. "Refactoring the JUnit framework using aspect-oriented programming", OOPSLA´2005 Companion, Poster Session, pp. 136-137, San Diego, October 2005.
22. N. Loughran, A. Rashid. "Framed Aspects: Supporting Variability and Configurability for AOP". Proceedings of ICSR'2004, pp. 127-140, 2004.
23. M. Mattson, J. Bosch, M. Fayad. Framework Integration: Problems, Causes, Solutions. Communications of the ACM, 42(10):80–87, October 1999.
24. M. Mattsson, J. Bosch. Framework Composition: Problems, Causes, and Solutions. In [7], 1999, pp. 467-487.
25. M. Mezini, K. Ostermann: "Variability Management with Feature-Oriented Programming and Aspects". Proceedings of FSE'2004, pp.127-136, 2004.
26. M. Mezini, K. Ostermann. "Conquering Aspects with Caesar". Proc. of AOSD'2003, pp. 90-99, March 17-21, 2003, Boston, Massachusetts, USA.
27. Y. Smaragdakis, D. Batory. Mixin Layers: An Object-Oriented Implementation Technique for Refinements and Collaboration-Based Designs, ACM TOSEM, 11(2): 215-255 (2002).
28. K. Sullivan, et al. Information Hiding Interfaces for Aspect-Oriented Design, Proceedings of ESEC/FSE´2005, pp.166-175, Lisbon, Portugal, September 5-9, 2005.
29. C. Zhang, H. Jacobsen. "Resolving Feature Convolution in Middleware Systems". Proceedings of OOPSLA'2004, pp.188-205, Vancouver, BC, Canada, October 2004..

Document Information

Title:	Proceedings of the Software Product Lines Doctoral Symposium
Date:	August 14, 2006
Report:	IESE-Report No. 104.06/E
Status:	Final
Classification:	Public