Model Driven Development of Software Product Lines

Alexandre Bragança[1] and Ricardo J. Machado[2]
[1] *Dep. Eng. Informática, ISEP, IPP, Porto, Portugal,*
alex@dei.isep.ipp.pt
[2] *Dep. Sistemas de Informação, Universidade do Minho, Guimarães, Portugal,*
rmac@dsi.uminho.pt

Abstract

Software product lines and related approaches, like software factories, are starting to capture the attention of the industry practitioners. Nevertheless, their adoption outside the research community and big companies is still very restricted. We believe that model-driven approaches, like OMG's MDA, with proper tool support, can bring the advantages of product lines to a broader audience. In this thesis we propose an approach to achieve this goal in which modeling is inspired by UML and automation is based on metamodeling and transformation languages using publicly available tools.

1. Introduction

The development of software systems is still a very hard and difficult engineering process. Complexity in the software systems is increasing everyday. Almost every project has to deal with supporting some sort of internet technology, integrate with legacy software and using more than one programming language or software platform.

The development of software systems requires knowledge from two main sources. One is of technical (computer) nature, e.g., programming in a specific language, manipulating xml documents, or understanding a communication protocol. The other is usually of non-technical (computer) nature and relates to knowledge about the problem that the software system is supposed to attack. This latter knowledge is necessary to understand the problem domain, while the former is used to build a solution, i.e., it relates to the solution domain. Because usually abstractions from those domains are so far apart it is very difficult to make accurate previsions about software projects. As a result, software projects are hard to manage, their costs may largely surpass budgets and the solution may not correspond entirely to the requirements.

A software product line is an approach to software development based on intra-organizational reuse through the explicitly planned exploitation of similarities between related products. This implies one (or more) common domain(s) shared by the developed applications (products) of the organization. Because applications share domains, it becomes possible to reuse software artefacts between applications and reduce the conceptual gap between the problem domain and the solution domain.

In a software product line approach the domain knowledge grows as each new application of the domain is developed. It is commonly accepted in the field that the initial investment in a product line approach can have return by the third developed application. Some authors, particularly Krueger, go further and defend that the adoption of a software product line can be beneficial from the first application developed if the approach is introduced incrementally [1, 2]. Nonetheless, all well-known software product lines have been implemented in large organizations or have required significant consultant knowledge from software product line specialists. Examples of such organizations are: Nokia, Philips, Motorola, Hewlett-Packard and Cummins Engine. Examples of software product line expert support organizations are SEI and IESE. Even if there are a few documented examples of software product lines in small to medium enterprises [3], one has to agree that the effort required implementing the necessary processes and methods may be out of reach for the majority of SMEs.

Recently, the software engineering community has witnessed the appearance of several proposals, such as aspect-oriented programming, feature-oriented program, domain-specific languages and model-driven development. Although diverse in nature, they all share the pretension of complement or solve some limitations of the dominant object oriented paradigm. Some of these proposals, notably domain-specific modeling and model-driven development are starting

0-7695-2948-8/07 $25.00 © 2007 IEEE
DOI 10.1109/QUATIC.2007.25

IEEE computer society

to capture the attention of the industry and major software development environments, like Eclipse and Visual Studio .Net are starting to support them. Also, large industry consortiums, such as OMG, are supporting and promoting such proposals.
We particularly defend the model-driven approach because it can be used at several levels of abstraction and in different components of a method. Also, with adequate tool support, this approach may automate the more cumbersome and demanding tasks of software engineering methods, like the methods used in software product lines.

The primary objective of this thesis is to provide a method that effectively enables the widely adoption of software product line approaches by software engineering practitioners. We propose to accomplish this goal by adopting model-driven engineering techniques and metamodeling

The remainder of this paper is structured as follows. In Section 2, we briefly present the state of the art of the research field. In Section 3, we present our research objectives and approach. Section 4 is dedicated to present our current work and preliminary results. In Section 5, we present the work plan. Section 6 is dedicated to concluding remarks.

2. State of the Art

The state of the art of this PhD work relates to essentially two fields of knowledge: Domain Engineering and Model-Driven Development. The first year of the PhD was dedicated to produce a report that covers a significant part of the state of the art [4]. We have also published work that describes our experience in the development of a domain-specific platform in a Portuguese software-house [5].
Providing a domain engineering state of the art is an overwhelming task. We could start with the work of Parnas on program families [6] and with the work of Neighbors, to my knowledge, the first explicit domain engineering methodology [7]. If we want to go even further, we can say that domain engineering appears also in the work of Dijkstra on structured programming [8], where he already speaks of step-wise program composition and program families. The more recent works are, naturally, on more specific sub-topics of the domain engineering field of knowledge. A most referenced work in the product-line area is the work of Kang et al. with the introduction of feature diagrams [9]. Regarding methodologies, Gomaa discuss the adoption of UML 2.0 for software product-line development [10]. IESE has produced a lot of industrial experience reports on software product-lines [11]. An overview of the practical application of domain engineering in product lines can be found on [3] and on software factories on [12].

Model-driven development is commonly associated with the OMG interoperability initiative named Model-Driven Architecture (MDA) [13]. Jean Bezivin states, however, that Model-Driven Engineering (MDE) encompass a more broader vision than MDA [14]. In this vision, models become first class entities and any software artifact becomes a model or a model element. MDE has its sources on other approaches such as domain-specific languages (DSLs) [15], model-integrated computing (MIC) [16] and generative programming [17]. One could argue that model-driven development (or model-driven engineering) is by its nature a domain engineering sub-field. In fact, in many model-driven approaches, domain-specific modeling languages are developed through metamodeling. However, such approach is not mandatory. It is possible to use generic software modeling languages, such as UML, in a model-driven approach. Therefore, our option is to view model-driven development in a broader way. When a model-driven approach implies metamodeling then it usually also becomes a domain engineering approach.

3. Research Objectives and Approach

As we have seen in the previous sections, domain engineering and, particularly, software product lines, have been adopted and are in use from some time now. However, the methods used, the existing tools and the required knowledge are not at the reach of all organizations. Some efforts have been done in this area, notably the work at IESE with PuLSE [18] and Kobra [19]. In such approaches, the heavy methods and techniques required for domain engineering are adapted to be used in more general contexts, like the ones where organizations use object-oriented or component-oriented approaches. Even so, adoption is difficult, because not all models are supported by tools and the degree of automation is low. As a result, organizations are required to keep models synchronized manually and, eventually, they abandon these tasks. Since the software product line approach is based on models (e.g., generic architecture, variability representation and domain requirements), in this context, its successful adoption by small organizations is difficult.
The primary objective of this thesis is to provide a method that effectively enables the widely adoption of software product line approaches by software

engineering practitioners. We propose to accomplish this goal by adopting model-driven engineering techniques. Particularly, we aim at provide specific approaches to support the automation of some key components of software product line methods by adopting techniques from the model-driven engineering field.

This objective is not pursued as a theoretical study. Instead, we take a pragmatic viewpoint in which existing theories, methods, tools and techniques can be combined to support our goal.

We also take this approach because our research is in the field of software engineering where, in contrast to other fields of computer science, the human factor is of the most importance. In this context, it is not sufficient to have the best technical solution; the process and all involved resources/persons must be taken into account.

This thesis follows the classification for software engineering research methods proposed by Adrion [20]: Scientific, Engineering, Empirical, and Analytical. Of these four research methods, and according to the same author, the empirical method is the most appropriate for software engineering research. The empirical method is based on the application of the proposed model to case studies in order to measure and analyze the results and, eventually, repeat the process. In contrast, the analytical method does not force the use of case studies; the results can be derived. The scientific method is the *traditional* research method that is based on the observation of the real world, and as such, is more tailored to natural sciences. In the engineering method, existing solutions are observed and better solutions proposed and developed. The new solutions are measured, analyzed and evaluated and the process is repeated if needed.

Although the empirical method is the most suited for software engineering research it is also a very demanding method since it requires the application of the proposed models to case studies in order to measure the results. In the context of the work of this thesis, it is not realistic to think it is possible to develop the sufficient case studies that the method requires and, consequently, we probably will not have the necessary quantitative evaluations of our work. As such, we decided to adopt qualitative measures of our proposals. These measures result essentially from our own experience in several software engineering projects in the form of discussions and observations with practitioners. These qualitative assessments of our proposals are usually presented in the form of demonstration cases that reflect real cases but are simplified in order to facilitate their description in research papers. We also use another form of evaluation of our work that consists in the analysis and comparison of other solutions to the same problems, which is a validation more common to the engineering research method.

Regarding validation approaches used by software engineering research works, Mary Shaw identifies the following types of validation [21]: Analysis, Evaluation, Experience, Example, and Persuasion. Shaw states that it is essential to select a form of validation that is appropriate for the type of research result and the method used to obtain the result. She also states that a simple example derived from a practical system may play a major role in validating a new type of development method. This is more in phase with the validation that, realistically, we hope to provide. Another indirect form of validation of our work is the feedback that scientific experts of the field provide at top conferences and workshops. In the last phases of the PhD, we also plan on support our approaches with concrete experimental implementations. We hope this also serves as validation, at least for validating the technical possibility of the implementation of the method proposals.

4. Current Work and Preliminary Results

During the PhD, we have adopted the 4SRS (*Four Step Rule Set*) model-driven method [22, 23], developed at Minho University, as a framework where we could integrate the different pieces of our work.

Figure 1 presents a very high-level and filtered overview of the activities involved in the 4SRS adapted for product line development. This adaptation was based on key characteristics of domain engineering and software product lines methods, such as the ones presented in Section 2. Our approach is also inspired by UML [24]. We have made this option because the original 4SRS is also based on UML. We also hope our findings will be more interesting to the community if based on a standard modeling language like UML.

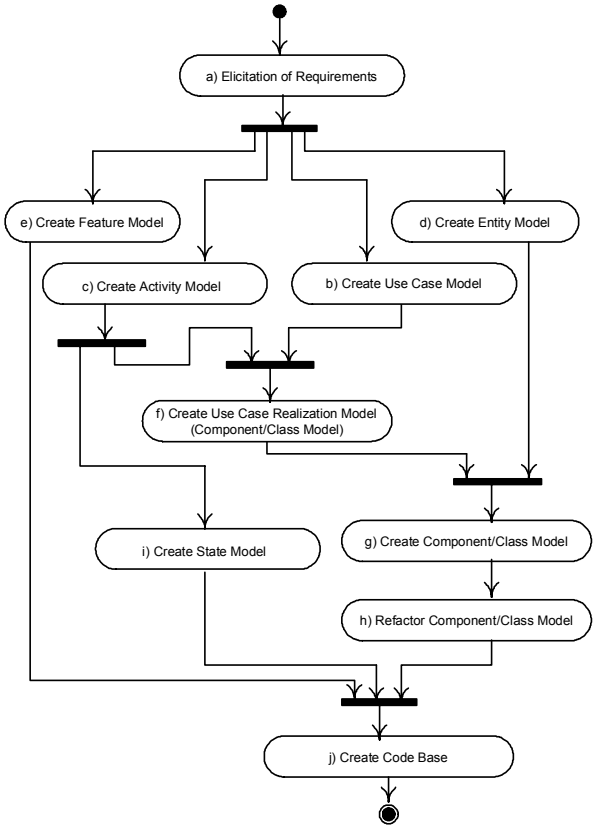

Figure 1. Proposed Process for Model-Driven Development of Software Product Lines.

The goal of this thesis is divided in three sub goals. Next, we detail the sub goals.

Sub goal 1: Provide a methodological approach to explicitly support variability in model-driven software development processes.

1.1 Extend model-driven methods to explicitly support variability.

1.2 Propose a variability notation for analysis models of model-driven methods.

Sub goal 2: Provide model-driven support to the analysis phase of the development of variability intensive systems.

2.1 Propose an approach to support the inclusion of analysis models into model-driven methods such that their transformation into platform independent models can be automated.

2.2 Propose an approach to integrate variability specific models and general purpose models such that their mappings can be automated.

Sub goal 3: Provide model-driven support to the architectural design phase of variability intensive systems.

3.1 Propose an approach to support the transformation between computation independent models and logical architectural models (platform independent models).

3.2 Propose an approach to support multi-staged model-driven scenarios, which are common in variability intensive systems.

Each of these sub goals topics resulted in approaches that were already described in research articles. These articles were all peer-reviewed anonymously. They were also publicly presented and discussed by researchers specialized in the scientific field.

Next we present our contributions and relate them to the major activities of the method (see Figure 1):

1) Extend the 4SRS model-driven method to support variability [23] (sub-goal 1.1, phases *b, c, f, g, h*).

2) Extend the UML 2.0 metamodel to add support for variability in use case diagrams [25] (sub-goal 1.2, phases *b, c*).

3) Propose an approach to specify the behavior of use cases by using activity diagrams [25, 26] (sub-goals 1.2 and 2.1, phases *b, c*).

4) Propose an approach to realize use case behaviors by using component diagrams [26] (sub-goal 3.1, phases *b, c, f*).

5) Propose an approach to refactor component diagrams in order to achieve the logical architecture of a system (or product line) [26] (sub-goal 3.1, phases *g, h*).

6) Propose an extension to the UML-F profile to support UML analysis models [27] (sub-goal 1.1, phases *b, c, f, g*).

7) Propose a clarification of the relationships between features and use cases and the automation of transformations between use cases and features (based on EMF and QVT) [28] (sub-goal 2.2, phases *b, e, c*).

8) Propose an approach to create Feature instantiation models from feature configuration models [28] (sub-goal 2.2, phase *e*).

9) Propose a set o patterns for multi-staged model-driven scenarios (and their solutions) [29] (sub-goal 3.3, phase *d*).

5. Work Plan

In the remainder of our work we plan on continue the research in the method in order to integrate all its phases and workproducts. We also plan on integrate all

the technological proposals into a set of tools based on EMF [30] and QVT [31].

6. Conclusions

In this paper we have presented our PhD work. Our goal is to provide a method that effectively enables the widely adoption of software product line approaches by software engineering practitioners. We have presented our work so far. A significant part of the work is already done. In the near future, we plan on integrate all the specific proposals into a set of experimental tools that can be used in case studies realized in the context of software development enterprises.

7. References

[1]C. W. Krueger, "New Methods in Software Product Line Development," 10th International Software Product Line Conference SPLC 2006, Baltimore, Maryland, USA, 2006.

[2]A. Braganca and R. J. Machado, "A Methodological Approach to Domain Engineering for Software Variability Enhancement," OOPSLA'2004 Second Workshop on Method Engineering for Object-Oriented and Component-Based Development, Vancouver, Canada, 2004.

[3]P. Clements and L. Northrop, *Software Product Lines - Practices and Patterns*: Addison Wesley, 2002.

[4]A. Braganca, "Run-Time Variability in Domain Engineering for Post-Deployment of User-Centric Software Functional Completion," U. Minho, Guimarães, PhD Report 2003.

[5]A. Braganca and R. J. Machado, "Run-time Feature Realization based on Domain-Specific Platforms," ICSR8 (Poster Presentation), Madrid, 2004.

[6]D. Parnas, "On the Design and Development of Program Families," vol. 2, pp. 1-9, 1976.

[7]J. Neighbors, "Software Construction Using Components," in *Department Information and Computer Science*. Irvine: University of California, 1980.

[8]E. W. Dijkstra, "Notes on Structured Programming," Technological University of Eindhoven, Eindhoven, The Netherlands 1969.

[9]K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study Technical Report," Software Engineering Institute, Carnegie Mellon University CMU/SEI-90-TR-21, 1990.

[10]H. Gomaa, *Designing Software Product Lines with UML*: Addison Wesley, 2005.

[11]M. Anastasopoulos, J. Bayer, O. Flege, and C. Gacek, "A Process for Product Line Architecture Creation and Evaluation - PuLSE-DSSA - Version 2.0," IESE 038.00/E, 2000.

[12]J. Greenfield, K. Short, S. Cook, and S. Kent, *Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools*: Wiley, 2004.

[13]OMG, "Model-Driven Architecture Guide Version 1.0.1," OMG, 2003, Available at http://www.omg.org.

[14]J. Bezivin, "Model Driven Engineering: Principles, Scope, Deployment and Applicability," Generative and Transformational Technics in Software Engineering, Braga, Portugal, 2005.

[15]P. Hudak, "Modular Domain Specific Languages and Tools," Fifth International Conference on Software Reuse, Victoria, Canada, 1998.

[16]A. Ledeczi, M. Maroti, A. Bakay, G. Karsai, J. Garrett, C. Thomason, G. Nordstrom, J. Sprinkle, and P. Volgyesi, "The Generic Modeling Environment," WISP'2001, Budapest, Hungary, 2001.

[17]K. Czarnecki, "Generative Programming Principles and Techniques of Software Engineering Based on Automated Configuration and Fragment-Based Component Models," in *Department of Computer Science and Automation*: Technical University of Ilmenau, 1998.

[18]J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "PuLSE: A Methodology to Develop Software Product Lines," Symposium on Software Reusability '99 (SSR'99), Los Angeles, 1999.

[19]C. Atkinson, J. Bayer, and D. Muthig, "Component-Based Product Line Development: The KobrA Approach," First Software Product Line Conference, Denver, Colorado, 2000.

[20]W. R. Adrion, "Research methodology in software engineering," Dagstuhl Workshop on Future Directions in Software Engineering, 1993.

[21]M. Shaw, "Writing Good Software Engineering Research Papers," 25th International Conference on Software Engineering, 2003.

[22]R. J. Machado, J. M. Fernandes, P. Monteiro, and H. Rodrigues, "On the Transformation of UML Models for Service-Oriented Software," ECBS International Conference and Workshop on the Engineering of Computer Based Systems, Greenbelt, Maryland, 2005.

[23]A. Braganca and R. J. Machado, "Deriving Software Product Line's Architectural Requirements from Use Cases: an Experimental Approach," 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software, Rennes, France, 2005.

[24]OMG, "Unified Modeling Language Version 2.0: Superstructure (formal/05-07-04)," OMG, 2005, Available at http://www.omg.org.

[25]A. Braganca and R. J. Machado, "Extending UML 2.0 Metamodel for Complementary Usages of the «extend» Relationship within Use Case Variability Specification," SPLC 2006, Baltimore, Maryland, 2006.

[26]A. Braganca and R. J. Machado, "Adopting Computational Independent Models for Derivation of Architectural Requirements of Software Product Lines," Mompes 2007, Braga, Portugal, 2007.

[27]A. Braganca and R. J. Machado, "Tracing Variability from Requirements to Design: A Model-Driven Approach with UML-F," *unpublished*, 2006.

[28]A. Braganca and R. J. Machado, "Automating Mappings between Use Case Diagrams and Feature Models for Software Product Lines," SPLC 2007, Kyoto, Japan, 2007.

[29]A. Braganca and R. J. Machado, "Transformation Patterns for Multi-Staged Model-Driven Software Development," *unpublished*, 2007.

[30]Eclipse, "Eclipse Modeling Framework," Eclipse Foundation, 2006, available at http://www.eclipse.org/emf/.

[31]OMG, "MOF QVT Final Adopted Specification (ptc/05-11-01)," OMG, 2005, Available at http://www.omg.org.