Multistage Model Transformations in Software Product Lines

Sofia Azevedo, Ricardo J. Machado
Departamento de Sistemas de Informação
Universidade do Minho
Guimarães, Portugal
sofia.azevedo@dsi.uminho.pt, rmac@dsi.uminho.pt

Dirk Muthig
Software Development Division
Fraunhofer – IESE
Kaiserslautern, Germany
dirk.muthig@iese.fraunhofer.de

Abstract—**Raising the level of abstraction for software engineers to write applications is still an undergoing issue. So, models will most likely become the dominant artifact in the development of software. However, models are nothing without the framing of a methodology, like the software factories methodology, which includes the software product lines approach. In the context of software product lines, model-driven development imposes the structuring of the software development process around models adequate to each one of the moments within the software supply chain. The different moments are the different stages that comprise different development teams, as well as the target user of the different software family members. This multistage process is a powerful vision of software development in general when compared with the current software development processes' state-of-the-art and this is the vision that feeds the Ph.D. work presented in this paper. This work is concerned with the transformations that models must suffer in the particular context of software product lines development, inside each stage and in between stages.**

Keywords: software engineering process, software requirements, software design.

I. INTRODUCTION

Software factories [1] is a model-driven development approach from Microsoft, which integrates domain-specific modeling languages, a kind of domain-specific languages [2], with software product lines [1, 3].

When a family of software products shares the same features, it can be called a software product line, although the product line development process must comprise not only the concern with the commonality among products but with the variation between them as well.

Software product lines are mainly concerned with domain, which contextualizes the use of domain-specific modeling languages by the software factories approach [4, 5] and justifies the domain specificity of these languages within the context of software factories. In the context of product lines, domain-specific modeling languages are an instrument to let users determine the values of variabilities between family members.

Software product lines are going to be mentioned throughout this paper despite domain-specific languages are not. This is because product lines are the approach where domain-specific languages make the most sense. Instead of referring to software factories, we prefer to be more specific and refer to software product lines.

The software factories methodology is much about the distinction between the commonalities and the variabilities of software product lines. In fact, as the relationship between product lines and domain-specific modeling languages is a premise of software factories, product lines are a process that may make use of domain-specific modeling languages to determine the variabilities of a product family in order to realize it in individual products. Model transformations allow producing those individual products out of models.

In the context of software product lines, as well as in the context of other approaches, transformations are useful when transforming models between different levels of abstraction, but they are also useful when transforming models at the same level of abstraction [6]. A model transformation is a process of mapping a model (or more than one model) into another model (or more than one model), in which a mapping function is involved to give birth to either other models that still need to be transformed or different levels of generated code [6]. Mapping functions represent repeated design decisions that conduct to the reuse of those functions in models of similar design.

Model transformations are contextualized within stages. Stages have been defined in [7] as the stages of software product lines development. Stage configuration was first mentioned also in [7]. This kind of configuration is the process of specifying a member of a product line, which is performed in stages of configuration where features are selected. Software factories are developed with stage configuration. Configuration stages may be the different phases of the product lifecycle (design, testing, deployment, maintenance and others) or the different parties/roles in the configuration process (facing this process as the process of eliminating variabilities). In multistage contexts, model transformations occur inside and between stages.

At the present time, it is essential to consider software development methodologies based on models. The reason for this is that models *per se* are useless; they must be contextualized within a methodology. If organizations describe their computer-based systems using UML (Unified Modeling Language), then, they will be aware that UML is only about notation and that methodologies to develop models are one step ahead. It is also essential to consider the guidance of methodologies for modeling, since modeling without the guidance of a methodology can be ineffective and is, in general, costly. At last, if a software product line is fabricated with a model-driven methodology behind and also with the roles of each actor within that methodology thoroughly defined, the way to a well established process begins to be delineated and the industry can begin to benefit from the advantages of model-driven software development and the product lines approach.

The remainder of this paper is structured as follows. Section II is devoted to the state-of-the-art in what software factories, from a procedural point of view, are concerned. Section III presents the research goals, deliverables and research approach. Section IV expresses the first results obtained with the work of this Ph.D. Section V is about the work plan's concerns and activities. Finally, Section VI provides for some concluding remarks.

II. State-of-the-Art

Software factories aim at reaching a wider range of customer needs with a wider software scope. A product family realizes an economy of scope, which means that multiple related designs are produced, and is targeted at custom markets where each product is unique [1]. Regarding the multiple related designs produced for each product family, models play a significant role as they can be reused across a number of applications in the product family. This is a strategy of long-term investment based on improving the quality of software design [5].

Software product lines are software systems that share common features considered as necessary for a market segment [8]. The development of family members is done using patterns, models, tools and frameworks (a framework can be seen as all the code that implements common domain aspects and extension points to customize applications built from that framework). Product lines are all about the distinction between commonalities and variabilities. Within the software development process of product lines two processes are related to each other [8]: the one that designs an architecture for the product line framework, which is the core asset development, and the other that uses the framework to produce individual products, which is the product development.

Domain-specific languages, which are described at the metamodeling level, are used to describe the concepts a software product line framework presents. Variability points in a domain-specific framework may be filled by using a domain-specific model. This is called framework completion [1]. The product line scope can also be defined through a domain-specific language [8]. Variabilities were defined by Mernik*, et al.* [2] as the information required to instantiate a system out of a broader one. Variabilities may be defined with a domain-specific language [2, 8]. Despite this, a generic architecture detaining the commonalities shared by the product line members must also be defined.

Software maintenance is a very important activity when working with models to generate software solutions over time. Some Software Engineering solutions that cover software evolution are [9]:

- *Object-oriented design patterns*; Design patterns are an approach for redesigning and generalizing design traces of object-oriented software. Software evolution, in this case, occurs when a design pattern is applied to an already existing software solution's design;
- *Software product line architectures*; Product line architectures represent reusable designs of product line members. Software evolution occurs when components with new features implemented are added to the overall architecture or others are removed from that same architecture;
- *Domain-specific languages*; Finally, when software needs to evolve, domain-specific languages must meet the required changes in software caused by that evolution.

The procedural dimension is imperative in the modeling of software product lines. The definition of such a process is needed in order to determine the role of models, patterns, domain-specific languages, tools and frameworks in the development of product lines. The process shall also be an agreement on how to handle commonalities, variabilities and architectures for the development of product families. Two examples of processes that consider models as the key artifacts to be manipulated in the development of software are the VA (Virtual Automation) methodology [10], which is about the roles of engineering professionals within the computer-based systems development, and the Bragança and Machado's multi-staged model-driven software development approach expressed in a series of model-driven transformation patterns [11]. Both the VA methodology and the multi-staged model-driven software development approach have well delimitated stages, actors who play well delimitated roles within one or more of those stages and activities which can be performed within one or more stages. Only this way companies can take advantages of applying a model-driven software development approach to their product lines' conception. Only by using a well structured process, companies can avoid the problems that emerge from ad-hoc modeling principles.

A multistage software development process can be defined as a software development process composed of some stages organized in a consecutive temporal order. Each stage is separated from the contiguous ones by well defined borders. Moreover, each particular stage is composed of a flow of well defined activities. Each stage's activities are conducted by specific professionals, using specific technologies (frameworks, languages, tools), under the directives of specific methodologies (processes, notations and methods) to achieve specific goals. A software product line can be considered as a software development process concerned with the features of a software product family according to three axis or dimensions: commonality, variability and detail. The modeling of a product line takes place at different stages, as Bragança and Machado stated earlier in [11]. Even though Bragança and Machado's multi-staged model-driven software development approach considered multistage software product lines, with actors and activities matching different stages, the process of developing product lines in a multistage manner was left unveiled. In particular, the stage transitions have not been defined, as well as the stages' technologies, methodologies and goals. In general, the implications of developing product lines obeying to a multistage process have not been addressed yet.

III. Research Objectives and Approach

Our work is targeted at accomplishing the following goals:

1. *Formalize a multistage software development process*; This goal is concerned with defining a software development process within the framing of the model-driven software development approach. The process' definition demands for the definition of the stages where the domain objects, which need to be identified as well, shall be specialized at. The refinement of models at different stages of a model-driven software development process is a current theme of the software product lines area;
2. *Characterize the model transformations required to occur in the context of the multistage process*; Model transformations will need to be performed inside each stage of the software development process, as well as between different stages. This second goal is about determining these model transformations and describing them. Process-oriented organizations benefit from the advantages of having a well defined process for developing their product families, which includes well defined model transformations, if the model-driven software development approach is considered. The software product lines' scientific community explores these topics nowadays.
3. *Automate the transformations to be executed throughout the software factory's development process*; This last goal has to do with exemplifying the whole set of model transformations that need to be performed between and inside the stages of the software development process. This goal must consider that automated models creation and maintenance is a current concern within the software product lines' community and, so, the model transformations will likely consider the automation requirement throughout the process. We shall also state that not only model-to-model transformations but also model-to-code transformations will have to be executed.

In the context of the company where the Ph.D. project is going to be developed, the following are going to be delivered:

- The software product line's formalization, corresponding to the product line's product portfolio;
- The multistage model-driven software product line's development process model, which will contemplate the refinement of models by means of automated model transformations, a set of modeling tools, or the same configurable modeling tool, as well as a set of professional roles, contextualized at different stages of the product line's development lifecycle;
- A process-integrated variability and commonality requirements traceability method with the support of modeling.

The research approach which is going to be used is the proof of concept, or concept implementation [12]. The proof of concept research approach is about demonstrating the feasibility of a solution to a problem. In this work, the question with feasibility is whether it is possible to sustain well defined processes for the development of software product lines with model-driven approaches using Microsoft tools in the context of a business software company's products, thus, to a practical problem, or not [13]. A series of mock-ups is going to be used as a means of validation of the software development process to be defined.

IV. Current Work and Preliminary Results

Commonality and variability are not the only concerns that must be kept in mind when modeling software product lines. Their scale is also a factor influencing the software development process based on modeling. The scale of product lines may justify the need for the refinement of product line architectures. Transforming models by adding details to those models is equivalent to refining them [6].

The first results of this work have to do with the refinement of software product line logical architectures. We have elaborated some refinement techniques to complement a product line modeling method. The refinement may be due to three reasons:

- The addition of detail to the software product line logical architecture;
- The transition from a phase of user requirements to a phase of system requirements in the development of the software product line, meaning that design decisions already present at the user requirements phase were used in the system requirements phase and are applicable to the product line logical architecture;
- Exceeding the maximum limit of use cases to which the modeling method we used, the 4SRS [14], can be applied.

The refinement of logical architectures is relevant to the software product lines development process as the logical architectures are one of the artifacts that shall be handled during the modeling-based development of product lines.

V. Work Plan and Implications

Our work will comprise the characterization of the model transformations required to occur in the context of the software product lines multistage modeling process to be defined and the automation of those transformations to be executed throughout the company's product line's development process. The process of modeling product lines in a multistage way is intimately related to the refinement of models at different stages of a model-driven software development process, including models of logical architectures for product lines, as well as other types of design models, like mechanistic ones. The derivation of mechanistic views of product lines from their logical architectures is an issue that requires exploration due to the value it adds to the process.

Despite the automated refinement of models during the software product lines multistage development process, the work plan embeds some other concerns. Among those are the correlation of processes for the purpose of elaborating an integrated multistage model-driven product lines development process traversal to all of the organization's teams, the variability (requirements) management throughout the product lines development lifecycle, the elaboration of systematic transformational patterns for the conversion between different modeling views of the product line and the elicitation of variability supporting use cases elaborated from user requirements.

From the requirements engineering perspective, the work plan is considering that software product line requirements shall be traceable at the M1 level of the four-layer modeling architecture for synchronization purposes, whenever one of the models is refined. This traceability strategy may encompass the construction of meta-metamodels at the M3 level of the four-layer modeling architecture, the MOF (MetaObject Facility). This work plan also takes in the study of the possibility of having the elicitation of user requirements, with the client, through tools delivered by modeling frameworks, which may force elicitators, by means of tool restrictions imposed by the modeling framework team's decisions, to use a set of concepts above the solution terrain (the solution terrain is the M1 level), a set of concepts above the concepts in the problem terrain (the problem terrain is the M2 level) or even both (to facilitate the creation of traceability links between metamodels at the M2 level, the problem domain level). This strategy may avoid the elicitation of solutions instead of the elicitation of problems or user needs, which would bias the goal of the whole requirements elicitation process with the client.

The work plan is also concerned with software product line customization and team interface. The product line modeling framework must support the concept of *role*. Consider the roles that people and organizations play within the context of a business. The domain-specific languages and the models must reflect the changes the product may suffer in the presence of different roles. The modeling framework tools shall also be aligned with the specializations of the development team (e.g. finances coding). During this task's execution, investigation is going to be performed to understand in which form business patterns can be provided to business analysts by the modeling framework tools, as well as to understand how the modeling framework tools can be provided to developers as resources they can use to detail the design of the product based on the analysis supplied by the Product Management.

The work plan is organized in some major activities: (1) practical research in order to analyze the company's software development activities by means of model-driven approaches, to analyze the company's products suite and, finally, to determine to which extent the suite is being handled as a software product line; (2) formalization of a multistage model-driven product lines development process; and (3) validation of that process in the real context of the company.

Major activity 1, which focuses on the company's product suite, is composed of the *software product line formalization* by defining the company's product portfolio, the product line's products, the product categories, the product requirements, the product line's functionalities, the product line's domain and subdomains and, at last, the responsibilities of the professionals who shall be involved in these definitions with particular roles, at different stages of the product line lifecycle. Naturally, the connections between the diverse definitions are going to be characterized with the aim of correlating them within the context of the process we are going to design. During major activity 1, each team working at the company is going to be analyzed in terms of accountabilities and established processes, since each one of them represents a different stage in the (internal) software development process, therefore, a different target user for the software product line's model-driven development framework.

Major activity 2 shall bring forth some *discussion sessions* within the company and particularly within each one of the software factory's teams, with the intention of arguing the proposed multistage model-driven software product line's development process and the effects of its activities projected on each of that teams' work.

Major activity 3 is going to yield some *experiencing sessions*, in which the process that is going to be proposed shall be experimented by its future key users (at least, these key users shall be the company's collaborators from the teams that are going to make use of the software product line modeling framework), according to some experimentation guidelines to be delineated. Such sessions shall also suit the purpose of iteratively improving the process with the participants' feedback, prompted by their trialing of the process and the implicated tools.

One of the most important outputs of this work is going to be the knowledge of the artifacts each team is going to manipulate, in which state they will enter the team's work, in which state they will leave the team's work and what transformations they will suffer in the context of each team's work in order to satisfy the needs of the team that is going to take those output artifacts as input for its work. Another important output of this work is going to be the knowledge of which artifacts in the metamodels and in the models shall be handled by the developers, only, and which shall be handled by the business analysts in the first place, for instance. These two outputs are going to be embedded in the process to be delineated during the execution of this work.

Figure 1 shows the evolution of the work plan over time. The plan is presented as a series of six exercises. The first is related to the preliminary results already mentioned in Section IV. It will address research objective 2 and has to precede experience 3 as it involves the transformation of models to increase the detail of the product line architecture. The second will propose a systematic approach for the use of patterns, in order to avoid subjective interpretations of patterns and to position the patterns along the stages of the multistage process that will be defined. It will address research objectives 1 and 2 as it will provide for procedural

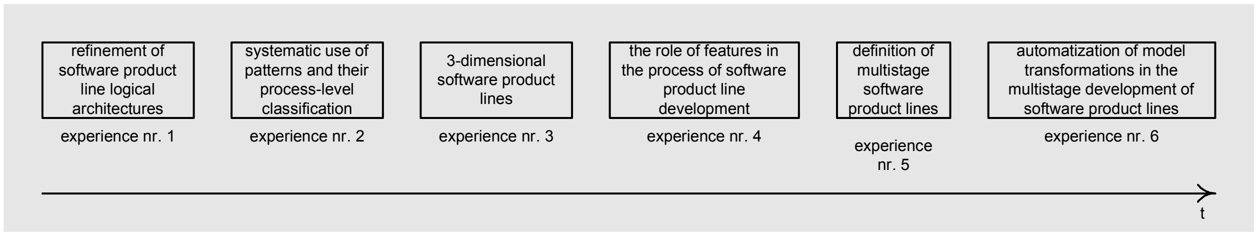
refinement of software product line logical architectures
experience nr. 1
systematic use of patterns and their process-level classification
experience nr. 2
3-dimensional software product lines
experience nr. 3
the role of features in the process of software product line development
experience nr. 4
definition of multistage software product lines
experience nr. 5
automatization of model transformations in the multistage development of software product lines
experience nr. 6
t

Figure 1. Work plan over time.

guidelines on the use of patterns, and a model-driven view of pattern transformations inside and between stages. It has to precede experiences 3 and 5 since it will deal with the detail dimension of product lines and plays a vital role in the multistage process because it attaches patterns to different stages. The third is targeted at exploring product lines from three dimensions: commonality, variability and detail. It will tackle research objective 1 since these dimensions are the basis of the whole product line procedural approach. Naturally, it has to precede experiences 4 and 5 as features have to be handled according to these dimensions and the whole multistage process must be founded on these same dimensions. The fourth is dedicated to features in the context of the whole product line development process and it must precede experience 5 as features are central in the whole process. For this reason, it will address research objective 1. The fifth is the process' definition and it will address research objective 1. At last, the sixth exercise is the automation of multistage model transformations and has to be performed after the multistage process is defined and all the transformations are characterized. It will address research objective 3.

VI. Conclusion

This work intends to accomplish the goals presented in Section III and it must be guided by the following principles: (1) model-driven software development and software product lines convey several advantages to organizations comparing with the well established code only approach; (2) organizations can profit from the implementation of processes specifically tailored to their approaches.

This project is going to be executed within the environment of an enterprise resource planning software house, therefore, within a real-world context. The demonstration case to be undertaken at the company is needed in order to validate the thesis to be proposed during the project's execution. The company's conscience on the need of a process to harmonize and optimize its undergoing and future activities on the field of model-driven development, reporting to the company's software product family, is preponderant to our work's grounding.

This thesis will provide for a contribute in the software engineering process area due to the considerations on the multistage development of product lines it will give, namely on the stage transitions, the stages' technologies, methodologies and goals.

References

[1] J. Greenfield and K. Short, "Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools," in 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA 2003). Anaheim, California, USA: ACM, 2003.

[2] M. Mernik, J. Heering, and A. M. Sloane, "When and How to Develop Domain-Specific Languages," ACM Computing Surveys, vol. 37, pp. 316-344, 2005.

[3] P. Knauber, D. Muthig, K. Schmid, and T. Widen, "Applying Product Line Concepts in Small and Medium-Sized Companies," Software, vol. 17, pp. 88-95, 2000.

[4] A. Demir, "Comparison of Model-Driven Architecture and Software Factories in the Context of Model-Driven Development," in 4th Workshop on Model-Based Development of Computer-Based Systems and 3rd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD/MOMPES'06). Potsdam, Germany: IEEE Computer Society, 2006, pp. 75-83.

[5] J. Bettin, "Model-Driven Software Development," in MDA Journal, 2004.

[6] A. W. Brown, J. Conallen, and D. Tropeano, "Introduction: Models, Modeling, and Model-Driven Architecture (MDA)," in Model-Driven Software Development, S. Beydeda, M. Book, and V. Gruhn, Eds. New York: Springer-Verlag, 2005, pp. 1-16.

[7] K. Czarnecki, S. Helsen, and U. Eisenecker, "Staged Configuration Through Specialization and Multi-Level Configuration of Feature Models," Software Process: Improvement and Practice, vol. 10, pp. 143-169, 2005.

[8] D. S. Frankel, "Business Process Platforms and Software Factories," in International Workshop on Software Factories. San Diego, California, USA: ACM, 2005.

[9] D. Batory, C. Johnson, B. Macdonald, and D. v. Heeder, "Achieving Extensibility Through Product-Lines and Domain-Specific Languages: A Case Study," ACM Transactions on Software Engineering and Methodology, vol. 11, pp. 191-214, 2002.

[10] R. J. Machado and J. M. Fernandes, "Heterogeneous Information Systems Integration: Organizations and Methodologies," in 4th International Conference on Product Focused Software Process Improvement (PROFES 2002). Rovaniemi, Finland: Springer-Verlag, 2002, pp. 629-643.

[11] A. Bragança and R. J. Machado, "Transformation Patterns for Multi-staged Model Driven Software Development," in 12th International Software Product Line Conference 2008 (SPLC 2008). Limerick, Ireland: IEEE Computer Society, 2008.

[12] I. Vessey, V. Ramesh, and R. L. Glass, "Research in Information Systems: An Empirical Study of Diversity in the Discipline and Its Journals," Journal of Management Information Systems, vol. 9, pp. 129-174, 2002.

[13] M. Shaw, "The Coming-of-Age of Software Architecture Research," in 23rd International Conference on Software Engineering (ICSE). Toronto, Ontario, Canada: IEEE Computer Society, 2001.

[14] A. Bragança and R. J. Machado, "Deriving Software Product Line's Architectural Requirements from Use Cases: An Experimental Approach," in 2nd International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MOMPES'05). Rennes, France: TUCS General Publications, 2005.