

Modeling Context and Dynamic Adaptations with Feature Models

Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, Jean-Paul Rigault

▶ **To cite this version:**

Mathieu Acher, Philippe Collet, Franck Fleurey, Philippe Lahire, Sabine Moisan, et al.. Modeling Context and Dynamic Adaptations with Feature Models. 4th International Workshop Models@run.time at Models 2009 (MRT'09), Oct 2009, United States. pp.10. hal-00419990

HAL Id: hal-00419990
https://hal.science/hal-00419990

Submitted on 18 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Modeling Context and Dynamic Adaptations with Feature Models

Mathieu Acher[1], Philippe Collet[1], Franck Fleurey[2], Philippe Lahire[1], Sabine Moisan[3], and Jean-Paul Rigault[3]

[1] University of Nice Sophia Antipolis, I3S Laboratory (CNRS UMR 6070), France
{acher,collet,lahire}@i3s.unice.fr
[2] SINTEF, Oslo, Norway
franck.fleurey@sintef.no
[3] INRIA Sophia Antipolis Mediterranée, France
{moisan,jpr}@sophia.inria.fr

Abstract. Self-adaptive and dynamic systems adapt their behavior according to the context of execution. The contextual information exhibits multiple variability factors which induce many possible configurations of the software system at runtime. The challenge is to specify the adaptation rules that can link the dynamic variability of the context with the possible variants of the system. Our work investigates the systematic use of feature models for modeling the context and the software variants, together with their inter relations, as a way to configure the adaptive system with respect to a particular context. A case study in the domain of video surveillance systems is used to illustrate the approach.

1 Introduction

Dynamic Adaptive Systems (DAS) are software systems which have to dynamically adapt their behavior in order to cope with a changing environment. A DAS needs to be able to sense its environment, to autonomously select an appropriate configuration and to efficiently migrate to this configuration. Handling these issues at the programing level proves to be challenging due to both the large number of contexts and the large number of software configurations which have to be considered. The use of modeling and the exploitation of models at runtime provide solutions to cope with the complexity and the dynamic nature of DAS [1].

DAS have similarities with Software Product Lines (SPLs). The basic idea of SPL engineering is to design and implement a products family from which individual products can be systematically derived [2]. An SPL is typically specified as a set of variation points together with their alternatives. The products are derived by selecting different sets of alternatives associated to the variation points. SPLs are usually modeled using feature models which provide an intuitive way for stakeholders to express the variation points, alternatives and constraints between these alternatives. Depending on how the feature models are linked to the SPL artefacts (e.g. models), the product derivation can be more or less automated.

Just like an SPL, an important characteristic of a DAS is its variability. SPL techniques can be applied to model the variability in a DAS but do not provide

any solution to model *when* the system and *how* the appropriate configuration should be chosen according to the environment. To tackle this issue, an emerging approach is to use *Dynamic* SPLs [3] which try to achieve the self-modification of a system by dynamically (re)binding variation points at runtime. Modeling a DAS not only requires the modeling of variability in the system. It also needs models of its environment and some adaptation rules that specify which configuration of the DAS should run in each specific context.

The contribution of this paper is to propose an approach based on SPL techniques not only for specifying the variability in the DAS but also the variability in its environment. The idea is to model the DAS and its environment as two different SPLs and then to link them in order to capture adaptation. In practice, the DAS and its environment are modeled using two independent feature models. They are then connected by dependency constraints that specify how the DAS should adapt to changes in its environment. The use of feature models allows one to present homogeneously the system, its environment and the associated constraints. The feasibility of the approach has been evaluated through a case study of a digital video processing application. To ensure its usability, the proposed approach has been built on top of the Domain Specific Modeling Language (DSML) for adaptive systems proposed in [4].

The paper is structured as follows. Section 2 first introduces the DSML-based approach on top of which the contribution is built, then recalls the basics of feature models. It also introduces our case study and finally details how the proposed approach leverages feature models to revisit the DSML-based approach. Section 3 details how feature models are used to model the context, the adaptive system and the adaptation rules. Section 4 compares the approach to related work. Section 5 concludes and addresses future work.

2 From DSML to Feature Models

2.1 A DSML for Modeling Adaptation

In [5], the authors propose to combine the use of models at runtime and aspect-oriented modeling techniques to implement DAS. The proposed approach is based on an adaptation model connected to a set of alternatives implemented as aspects. At runtime, a reasoning engine processes the adaptation model and selects the functionality which matches the context. The adaptation of the system is implemented by weaving the corresponding aspects in a model kept causally connected with the running system. The adaptation model is made of four main elements:

Variants They make references to the available variability for the application. Depending on the complexity of the system, it can be a simple list of variants or a data structure like a hierarchy.

Constraints They specify constraints on variants to be used over a configuration. For example, the use of a particular functionality (variant model) might require or exclude others. These constraints reduce the total number of configurations by rejecting invalid configurations.

Context The context model is a minimal representation of the environment of the adaptive application to support the definition of adaptation rules. It only considers elements of the environment relevant for expressing adaptation rules. These elements are updated by sensors deployed on the running system.

Rules These rules specify how the system should adapt to its environment. In practice, these rules are relations between the values provided by the sensors and the variants that should be used.

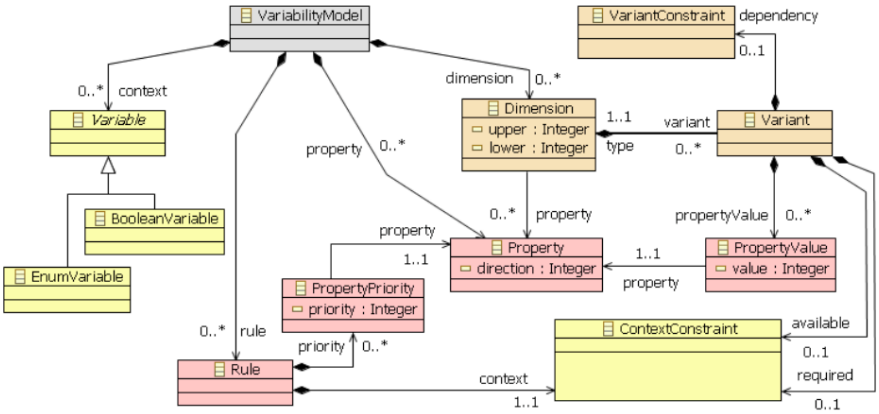


Fig. 1. Excerpt of the adaptation DSML

Until then, two different formalisms have been experimented for capturing the adaptation rules. In [5], the adaptation model is based on event-condition-action (ECA) rules. In [4], the adaptation model combines a set of hard-constraints and optimization rules. Both of these approaches rely on a model capturing the variability in the system and the variability in the context. Fig. 1 presents an excerpt of the adaptation DSML used in the last approach [4]. The root class for the DSML is *VariabilityModel*. On the left, it contains a set of variables which model the context of the DAS. On the right, it contains a set of *Dimensions* and *Variants* which models the variability in the system. The classes *VariantConstraint* and *ContextConstraint* are used to express hard-constraints between the system and its context. Finally, the classes *Rule* and *Property* are used to choose the best solution among the acceptable configurations in a particular context.

2.2 Feature Models

Feature models (FMs) are perhaps the most common formalism used to model SPL commonalities and variabilities [6,7,8]. A FM can capture different kinds of variability, ranging from high-level requirement variability to software variability. SPL variants are configured by selecting a set of features that satisfy FM constraints. Every *member* of an SPL is represented by a unique combination

of features and a FM compactly defines all features in an SPL and their valid combinations. It is basically an AND-OR graph with constraints which organizes hierarchically a set of features while making explicit the variability.

A *configuration* of a FM is a set of concrete features. An SPL is the set of all configurations that are valid for the FM which represents the SPL. A configuration is valid if the selection of all features contained in the configuration and the deselection of all other concrete features is allowed by FM. The semantics of FM is defined as follow: *i)* if a feature is selected, so should be its parent; *ii)* if a parent is selected, all the mandatory child features of its And group, exactly one of its Xor group(s), and at least one of its Or group(s) must be selected; *iii)* cross-tree constraints relating features (e.g. feature dependencies) must hold. For brevity's sake, we do not exemplify here FMs; the reader will find examples in Section 3.

2.3 Video Surveillance Case Study

Throughout the paper we propose to compare the systematic use of FMs with the DSML-based approach for modeling the adaptive system with a case study of Video Surveillance (VS) systems.

The purpose of VS is to analyze image sequences to detect interesting situations or events. The corresponding results may be stored for future processing or may raise alerts to human observers (detecting intrusion, counting objects or events, tracking people or vehicles, recognizing specific scenarios, etc.). At the implementation level, a typical VS processing chain starts with image acquisition, then segmentation of the acquired images, clustering to group image regions into blobs, classification of possible objects, and tracking these objects from one frame to the other. The final steps depend on the precise task (see Fig. 2).

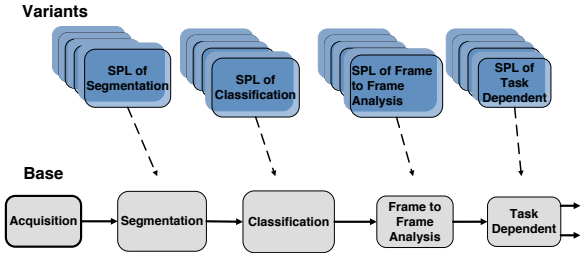


Fig. 2. A processing chain of the VS system and the SPLs

An important issue is that each kind of task has to be executed in a particular context. This context includes many different elements: information on the objects to recognize (size, color, texture, etc.), nature and position of the sensors (especially video cameras), etc. The number of different tasks, the complexity of contextual information, and the relationships among them induce many possible variants at the specification level, especially on the context side. The first activity of a VS application designer is to sort out these variants to precisely specify

the function to realize and its context. In our case study, the underlying software architecture is component-based. The processing chain consists of any number of components that transform data before passing it to other components. As a result, the designer has to map this specification to software components that implement the needed algorithms. The additional challenge is to manage the dynamic variability of the context to cope with possible runtime change of implementation triggered by context variations (e.g. lighting conditions, changes in the reference scene, etc.).

2.4 Revisiting the Approach with Feature Models

In comparison with the DSML approach previously described [4], we propose a more intuitive and more compact notation for the adaptation models. One of the major benefits of the DSML is to provide the ability to simulate and validate the adaptation model at design-time. Part of our objective is thus to keep a similar expressiveness so that existing simulation and validation techniques can be reused.

The key idea is to model the context and the software variants as two families (i.e. SPLs). The context model is represented as an SPL of context where each member of the SPL describes one valid state of the context. The software system is also an SPL and should adapt itself with respect to a contextual information. In SPL terminology, adaptation to context changes corresponds to product derivation or product configuration (i.e. the choice of a member of the SPL). In our case, we use a FM both for modeling the SPL of context and the set of possible variants. As previously described, the specification of constraints between features is possible. The constraints on variants can thus be directly expressed in the FM formalism. Moreover, the inter relations between context elements and software variants specify the adaptation rules. They are also represented with constraints between features of the context FM and features of the software variants FM. Using FMs, a configuration of the context corresponds to the actual contextual information where the software operates. At runtime, context changes mean that the context FM has a new configuration. A configuration of the software variants is an effective software system considering that all variants are then comprehensively integrated. The concept of configuration in FM formalism also helps to clarify the semantics relation between the two models: each configuration of the context should correspond to a configuration of the software system.

3 Modeling Context and Adaptation

3.1 Modeling Software Variants

All steps of the VS processing chain correspond to software components that the designer must correctly assemble to obtain a processing chain. A mandatory task is to acquire images. Then, for each step, many variants exist, along different dimensions (see Fig. 2). The first challenge is thus to model the software variants.

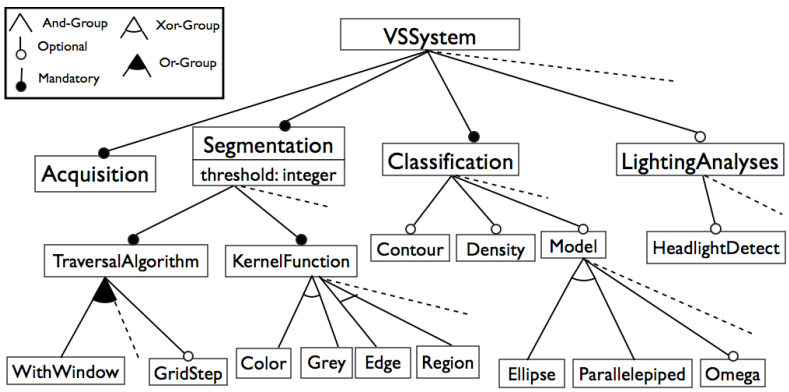


Fig. 3. SPL of the VS system

For instance, there are various Classification algorithms with different ranges of parameters, using different geometrical models of physical objects, with different strategies to identify relevant image blobs. Another example is depicted in Fig. 3 where the subtree of the FM whose root is Segmentation represents a family of segmentation algorithms. For each combination of subfeatures of Segmentation, we assume that there is a corresponding component. As an example, if TraversalAlgorithm, Grid Step, With Window, Kernel Function, Edge, Color features are selected, a fully parameterized component can be derived. Another component would be derived if the Region and Grey features are selected (instead of Edge and Color). Additional constraints are used to express dependencies between features:

$$\text{GridStep } \textbf{or } \text{WithWindow } \textbf{excludes } \text{Edge} \qquad (C_1)$$

$$\text{GridStep } \textbf{excludes } \text{Ellipse} \qquad (C_2)$$

$$\text{Edge } \textbf{excludes } \text{Density} \qquad (C_3)$$

The constraint C_1 means that if the features GridStep or WithWindow are selected, then it is not possible to select Edge. Note that the constraints do not necessary relate features of the same kind of algorithms (e.g. the constraint C_2 states that if the feature GridStep of a segmentation algorithm is selected, then the feature Ellipse of a classification algorithm is not selected). These constraints are expressed in propositional logics and correspond to the hard-constraints defined in the DSML (see Section 2.1). (A **excludes** B is a shortcut to express A **implies not** B).

Models at runtime may deal with values, which can be for example provided by sensors. To be able to handle them we need to use an extended formalism of basic FMs that propose adding extra-functional information to the feature using attributes [9]. A feature attribute has a a domain and possibly an assignment value when the feature is selected. For instance, *threshold* is an attribute of the feature Segmentation whose type is an integer in Fig. 3. Another example is the attribute of the feature TimeOfDay which can take the value *night* or *day* in Fig. 4. An attribute corresponds to the *Property* concept defined in the DSML.

3.2 Modeling Context

At present, there is a set of configurations available for each category of component in order to achieve each task of the processing chain. The contextual information is needed to reason at runtime on the effective choices of components. For instance, lighting changes can have an impact on the parametrization (i.e. configuration) of some components of the processing chain. Our approach is to represent the context model also as a FM. In Fig. 4, a contextual information that needs to be defined is the Objects of interest(s) to be detected, together with their properties.

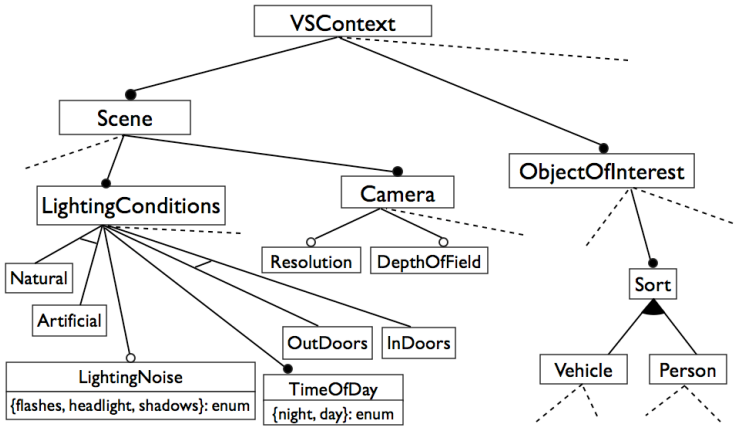
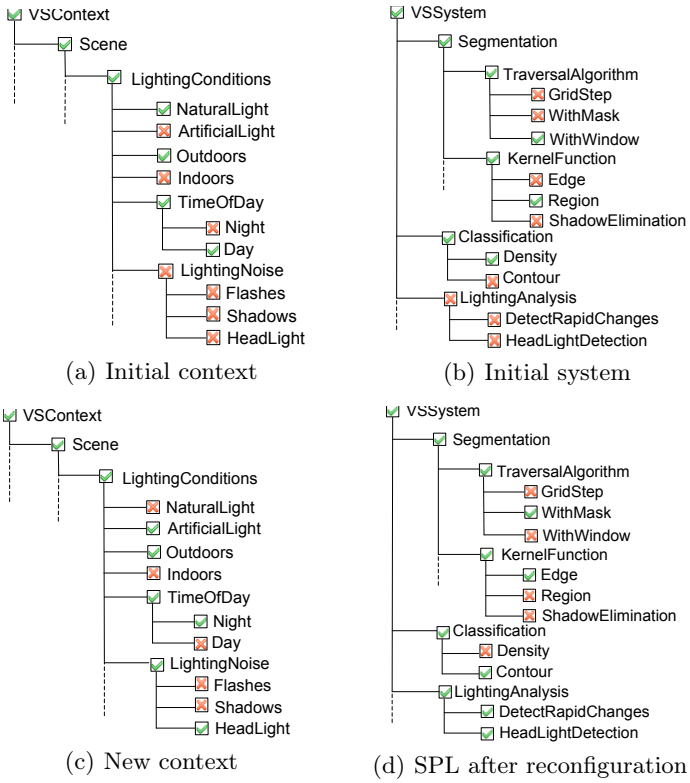


Fig. 4. SPL of the context

Then, Scene is the feature with the largest sub-tree; it describes the scene itself (its topography, the nature and location of cameras) and many other environmental properties (only some of them are shown on the figure). The elements of the FM may be related together with intra constraints which can reduce the configuration space.

3.3 Modeling Adaptation

The purpose here is to define the adaptation logics that defines the behaviour of the software system considering the context. In our approach and terminology, it means that the SPL system should change its configuration. As the available context information is also a configuration that affects the SPL configuration, we propose to inter relate the two FMs with adaptation rules. Adaptation rules are defined with the constraint language of FMs (i.e. propositional logic-based language). Abstract syntax rules consist of a Left hand side (LHS) and Right hand side (RHS). Both of them address features possibly connected with “and”, “or”, “not”. With this mechanism, simple ECA rules can be expressed. The LHS represents the condition part and is an expression based on the context information. The action is a change in the configuration of variants (RHS of the rules).



(a) Initial context (b) Initial system

(c) New context (d) SPL after reconfiguration

Fig. 5. Configurations of the VS system with respect to context changes

As an example, the following adaptation rule:

$$\text{Night } \textbf{and } \text{HeadLight } \textbf{implies } \text{HeadLightDetection} \qquad (AR_0)$$

states that if the contextual information describes that the Night and HeadLight are active, then the feature HeadLightDetection which corresponds to a component of the platform is integrated in the software system.

Some other adaptation rules are defined as follow:

$$\textbf{not } \text{LightingNoise } \textbf{implies } \text{Region} \qquad (AR_1)$$

$$\text{LightingNoise } \textbf{implies } \text{Edge} \qquad (AR_2)$$

$$\text{ArtificialLight } \textbf{implies } \text{DetectRapidChanges} \qquad (AR_3)$$

$$\text{Flashes } \textbf{or } \text{HeadLight } \textbf{implies } \text{Contour} \qquad (AR_4)$$

The Fig. 5(a) depicts an excerpt of the initial context configuration where the VS system operates. (The green icon in the box states that a feature is selected while the red cross means that the feature is deselected). One can notice that the system is executed in an Outdoor environment during the Day. The corresponding configuration of the VS system is represented in Fig. 5(b). As the feature LightingNoise is not selected in the context, the feature Region is selected in the VS system applying the rule AR_1.

An adaptation of the VS system is required with respect to context changes depicted in Fig. 5(c). Here, the system should run during the Night and the

light is now Artificial. Additionally, the HeadLights (e.g. of the vehicles) should be taken into account. Applying the different rules (AR_0, AR_2, AR_3 and AR_4), the adaptive system is then configured (see Fig. 5(d)).

4 Related Work

Feature Modeling. As in our running example, a few other approaches use multiple FMs during the SPL development. Kang et al. define four layers, each containing a number of FMs [6]. The paper discusses these layers and their FMs on a structural level and provides guidelines for building FMs. Some layers defined in their papers, like *operating environment* or *capability*, can be part of the contextual information. In [7], FMs are used to model decisions taken by different stakeholders at different stages of the software development. Hartmann et al. introduce context variability model (CVM) to represent context information of software products [10]. The combination of the CVM and another FM results in a so-called Multiple Product Line FM. All above-mentioned papers do not explicitly present their work as suited to self-adaptive or dynamic systems. The configuration of the FMs is rather static and does not evolve over time. In [11], Fernandes et al. propose to model context-aware systems. As in our approach, the authors use FMs to represent the context and the software system. A domain-specific language is also designed and allows the developer to specify context rules. Nevertheless, the formalism and notation used to represent FMs are not standard ; their work can be seen as an hybrid approach between the DSML-based and the FM-based approaches.

Adaptation Modeling. Beyond the approach presented in [4] and discussed in the beginning of the paper, a number of techniques have been proposed in the literature to capture and express adaptation. These techniques can be categorized under two families. The first one is the most commonly used and is based on event-guard-action type of rules [5, 12]. In these approaches the context and the configurations are related by a set of rules, which express how the evolution of the context should affect the running configuration of the application. The second family of approaches relies on the optimization of some utility functions [13,3] associated to the system. In these approaches, changes in the environment trigger an optimization process that evaluates possible alternative configurations and adapt the system to maximize the utility of the running configurations. The FM notation proposed in this paper is very well suited to represent event-guard-action type of rules as these rules can be captured as constraints across the FMs. For optimization based-techniques, a possible solution would be the use of attributes in the FMs in order to represent the information needed by the optimization process. This will be investigated as future work.

5 Conclusion and Future Work

This paper addresses the reconfiguration of Dynamic Adaptive Systems and proposes to reason at the model level in order to compute the new configuration according to context changes. We have shown that the concepts in the DSML are expressible with the FM formalism. The concept of configuration is naturally present in FMs: the valid combination of variants or context elements is

defined by the semantics of the FMs. As a result, there is no need to define an ad-hoc semantics or constraint checking techniques. The context elements are no longer represented as Boolean variables and the user can structure hierarchically domain concepts. Besides, it is possible to express constraints between the elements of the context model, invariants and adaptation in the FM formalism. The uniform representation of the context model and the software system makes possible to express relations between the two models. The DSML-based approach and the FM-based approach can complement each other. On the one hand, the FM-based approach can take advantage of simulation and validation checking proposed in [5, 4]. On the other hand, the DSML-based approach can use the infrastructure, tools, techniques, etc. associated to FMs. As part of our future work, we intend *i)* to address the validation and simulation directly at the level of FMs and *ii)* to leverage the expressiveness of the FM-based approach (e.g. using attributes). We also plan on achieving a better separation of concerns thanks to a set of operators allowing to extract a subset of the context model; we believe that the context that may influence the runtime execution of the system is most of the time only a part of it. Finally, our long term goal is to connect state-of-the-art adaption engines to our models and provide an end-to-end software solution.

References

1. Morin, B., Fleurey, F., Bencomo, N., Jézéquel, J., Solberg, A., Dehlen, V., Blair, G.: An Aspect-Oriented and Model-Driven approach for managing dynamic variability. In: Model Driven Engineering Languages and Systems conference. (2008)
2. Pohl, K., Böckle, G., van der Linden, F.J.: Software Product Line Engineering: Foundations, Principles and Techniques. Springer-Verlag (2005)
3. Hallsteinsen, S., Stav, E., Solberg, A., Floch, J.: Using product line techniques to build adaptive systems. In: Software Product Line Conference. (2006)
4. Fleurey, F., Solberg, A.: A domain specific modeling language supporting specification, simulation and execution of dynamic adaptive systems. In: Model Driven Engineering Languages and Systems conference. (2009)
5. Fleurey, F., Delhen, V., Bencomo, N., Morin, B., Jézéquel, J.M.: Modeling and validating dynamic adaptation. In: Proceedings of the 3rd International Workshop on Models@Runtime, at MoDELS'08, Toulouse, France (oct 2008)
6. Kang, K., 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**(1) (1998) 143–168
7. Czarnecki, K., Helsen, S., Eisenecker, U.: Staged Configuration through Specialization and Multilevel Configuration of Feature Models. Software Process: Improvement and Practice **10**(2) (2005) 143–169
8. Batory, D.S.: Feature models, grammars, and propositional formulas. In Obbink, J.H., Pohl, K., eds.: SPLC. Volume 3714 of LNCS., Springer (2005) 7–20
9. Benavides, D., Ruiz-Cortés, A., Trinidad, P.: Automated reasoning on feature models. LNCS, CAiSE 2005 **3520** (2005) 491–503
10. Hartmann, H., Trew, T.: Using feature diagrams with context variability to model multiple product lines for software supply chains. In: SPLC '08, IEEE (2008) 12–21
11. Fernandes, P., Werner, C.M.L.: Ubifex: Modeling context-aware software product lines. In Thiel, S., Pohl, K., eds.: SPLC (2), Limerick, Ireland (2008) 3–8
12. Zhang, J., Cheng, B.H.C.: Specifying adaptation semantics. In: WADS '05: Proceedings of the workshop on Architecting dependable systems, ACM (2005) 1–7
13. Floch, J., Hallsteinsen, S., Stav, E., Eliassen, F., Lund, K., Gjorven, E.: Using architecture models for runtime adaptability. IEEE Softw. **23**(2) (2006) 62–70