Dynamic Software Product Lines for Service-Based Systems

Paul Istoan, Gregory Nain, Gilles Perrouin, Jean-Marc Jézéquel
INRIA, Centre Rennes - Bretagne Atlantique,
Campus de Beaulieu, Bat 12F
35042 Rennes, France
Email: {paul.istoan,gregory.nain,gilles.perrouin,jezequel}@irisa.fr

***Abstract*—Ageing populations and the necessity to reduce environmental impact raise new challenges on our living buildings. Convergence of home control systems (air conditioning, light management) and computer science, or *house automation*, allows to enhance comfort, security and health of inhabitants, and reduce energy consumption. Each of these abilities can be perceived as a service provided by the house automation system. Starting from this point, we developed ENTIMID , a middleware able to make systems from different brands cooperate in a single service-based platform. Yet the proliferation and variability of such services, and needs to tailor each system to a particular building, make the design of these systems complex. In this prospective paper, we explain how the notion of dynamic software product line facilitates such designs by providing sophisticated techniques for managing variability across services from design time to runtime and allowing their automatic composition.**

I. INTRODUCTION

Evolution of populations as well as living habits (remote working, in-home caring for aged and/or disabled persons) are deeply modifying the relationships and needs that we have with respect to our buildings. In order to address such issues, the field of home automation, which used to be a purely electro-mechanical discipline, is introducing more and more software in its systems in order to increase the flexibility and adaptability of home automation solutions. In this context, we developed ENTIMID , a service-based middleware designed to solve house automation issues such as devices interoperability, linkage facilities or scenarios descriptions. The framework developed for ENTIMID [1] eases the communication of devices coming from different manufacturers and their publication over different device-control protocols such as UPnP for local use, or DPWS for a WebService access.

A typical scenario for using ENTIMID is the deployment of a system for managing a building. It will use information coming from sensors and several other services to act on the building, in order to increase the standard of living and comfort of the inhabitants. But houses and buildings are equipped with different products from different manufacturers that communicate using different protocols. As a consequence, the deployment of ENTIMID is unique and specific for each building or house. Moreover, the needs of each system user are different, and may require specific data retrieved from WebServices, which leads to an infinity of deployment configurations. Further more, such configurations are required to be adaptive. If a service stops functioning or if one that offers a better quality service is available, that service needs to be replaced.

Software, technologies and protocols constantly evolve and versions change with, sometimes, some compatibility problems. That is to say that during its life, an ENTIMID based system will have to implement new protocols, or different versions of a given protocol. Moreover, protocols can be used in different versions, at a given time, in different places of the city. Once again, the different personalities make it possible for ENTIMID to gain multiple version compliance, for different protocols.

This inherent variability engenders a great complexity for the design of such systems. Variability management is an issue that has been well studied in the context of software product lines in which assets are reused amongst products with respect to their commonalities and variabilities. This high-level of reuse allows significant economies of scope and unprecedented productivity gains. In addition runtime variability management is addressed by the emerging dynamic software product line approach. If we combine these techniques with a service oriented infrastructure such as ENTIMID , we would be able to offer developers the flexibility needed to address specificities in the configuration of a particular house/building while offering them the possibility to reuse as much as possible existing services as well as previous building configurations.

In this paper, we investigate the feasibility of the synergy between service oriented architectures and dynamic software product lines in the context of home automation systems. Section II introduces ENTIMID and its development context to highlight the deployment specification difficulties. We then introduce dynamic software product lines techniques in Section III. In Section IV we discuss of the particular application of DPSL techniques on ENTIMID and discuss in Section V of the challenges emerging from this synergy in our context. Section VI wraps up with conclusions and future work.

II. ENTIMID

Modern enterprises need to respond effectively and quickly to opportunities in todays constantly changing and increasingly competitive global markets. A current approach for addressing these critical issues is embodied by services that can be easily assembled to form a collection of independent and loosely coupled business processes [2], creating a service oriented

architecture (SOA). The goal of SOA, as mentioned in [3] is to eliminate barriers like application integration, transaction management and security policies so that applications run smoothly. SOA is a means of designing software systems to provide services to either end user applications or other services through published and discoverable interfaces. In this way, a SOA can deliver the flexibility and agility that business users require, and satisfy the ongoing and changing needs of businesses.

ENTIMID is a middleware implementation developed in a house or building automation context. The aim of this middleware, is to offer a level-sufficient abstraction of inhouse devices, making it possible for high level services to interact with physical devices (such as lamps, heater or temperature sensors) and ease their management.

The plethora of networked devices embedded in appliances, such as mobile phones, televisions, thermostats, and lamps, makes possible to automate and remotely control many basic household functions with a high degree of accuracy. Consequently, a new breed of technologies is needed to address the challenges of the development and deployment of building automation applications over an evolving, large-scale distributed computing infrastructure. The approach and the tools provided by ENTIMID are an example of such a technology. ENTIMID offers a first solution to manage the devices multiplicity and the evolution of their communication protocols through a layered multi-personalities middleware. This solution consists in offering a common abstraction of the home device topology and provides a generative approach for accessing devices through different personalities (protocols). To improve the flexibility of this middleware, the high level protocols are generated and loaded at runtime and so, enables a dynamic reconfiguration, of the application and the high-level protocol bundle, without any system restart. ENTIMID have been implemented to form a complete middleware for home automation.

Each bundle is designed to reach the highest level of independence, giving the software enough modularity to allow partial services updates, adds or removes. This programming style allows software-builders, to deploy the same pieces of software for all of their clients, either professionals or private individuals, and then simply adapt the services installed. Moreover, the services running on the system can be changed during execution.

The difficulties of deployment of this system can be expressed in two dimensions.
In space, the services used will be different from a building to another, and at a city scale, the possibilities of configurations are potentially infinite. At this point, we clearly need a mean to easily create or modify all these configurations.
In time, all the system configurations previously described will evolve during the system life. Protocols may change of version, and some buildings may adopt the latest version whereas some others may want to keep the version they have. Moreover, new products and/or protocols may have to be included in the system and so, considered and managed by the system configuration. This evolution in time adds to the already complex management of the system configurations.

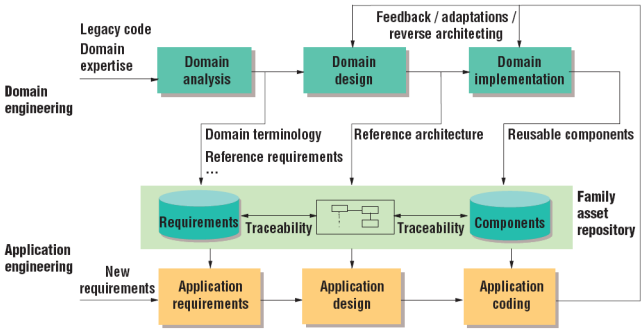

Fig. 1. The general process of product line engineering

III. DSPL

Software product lines, or software families, are rapidly emerging as a viable and important software development paradigm [4]. Companies, such as Hewlett-Packard, Nokia, or Motorola, are proving that using a product line approach for software development can generate important quantitative and qualitative improvements in terms of productivity, time to market and customer satisfaction. This practice can also efficiently satisfy the current need for mass customization of software [4]. Their growing success is due to their ability to offer companies ways to exploit their software products commonalities to achieve economies of production.

More precisely, a software product line is a set of software-intensive systems sharing a common, managed set of features. They satisfy the specific needs of a particular market segment or mission and are developed from a common set of core assets in a prescribed way [5]. The main idea behind Software Product Lines (SPLs) is to capture the essential concepts of commonality and variability. Rather than describing a single software system, the model of a software product line describes the set of products in the same domain. This can be accomplished by distinguishing between elements common to all the products of the line, and elements that may vary from one product to another. It also relies on the reusability of core assets, which form the basis of the product line, rather than working from scratch [6].

Adopting the SPL approach implies performing two main activities: domain engineering and application engineering. Figure 1 provides a graphical representation of the general process of Product Line Engineering, as found in the literature [7].

Domain engineering deals with the common set of core assets serving as a base to develop features. It starts with a domain analysis phase that identifies commonalities and variability among SPL members. It continues with the domain design, which consists of establishing the product line architecture. Finally, during the domain implementation, the architecture will be actually implemented [8]. There are two popular

techniques to define software product lines requirements: Feature Modeling and Use Cases. Variability amongst features is usually depicted using a feature diagram [9]. Use Cases [10], [11] are a well known notation to describe requirements of a software system. Several extensions were proposed to extend both textual and UML-based use case notations in order to improve variability support amongst software product lines.

Application Engineering, also known as product derivation, is the specific activity of developing the set of software intensive systems from the common set of core assets. The PL derivation consists in generating from the PL model the UML class diagram of each product. The system is built based on the results of Domain Engineering. Requirements are selected from the existing domain model, which matches the customers needs. The applications are assembled from the existing reusable components. According to the derivation technique used, currently available approaches to support product derivation can be organized in two main categories: configuration and transformation. Product Configuration [12] originates from the idea that product derivation activities should be based on the parameterization of SPL core assets. Product Derivation by Transformation [6] uses Model Driven Engineering techniques in order to support product derivation. It provides models as useful abstractions to understand assets and transformations able to use them as first-class artifacts for product generation.

Emerging domains such as ubiquitous computing, service robotics, house automation or ambient intelligence are proving that applications are increasing in complexity, and will demand a higher degree of adaptability from their software systems [13]. Resource constraints and user requirements can fluctuate and change dynamically during runtime. New SPLs will have to face this challenge. There is therefore a need for dynamic SPLs which produce software capable of adapting to such fluctuations. Another reason for the introduction of dynamic SPLs is the impossibility to foresee all the functionality or variability an SPL requires. In contrast with traditional SPLs, dynamic SPLs bind variation points at runtime, when software is launched to adapt to the current environment, as well as during operation to adapt to changes in the environment. Building a product line that dynamically adapts itself to changing requirements implies a deployment of the product configuration at runtime [14]. It also means that the system requires monitoring capabilities for detecting changes in the environment. As a response to these changes, the system adapts by triggering a change in its configuration, providing context-relevant services or meeting quality requirements. Dynamic software reconfiguration is concerned with changing the application configuration at runtime after it has been deployed.

In terms of features, dynamism means that both configuration of features and their quality constraints vary at runtime [15]. Reconfigurations are also needed to implement feature availability and qualities. Dynamic addition, deletion or modification of product features are some examples of dynamic reconfiguration for SPLs. Other issues concerning DSPLs concern the commonality and variability analysis from the quality point of view. Identifying them at the software quality level is much more difficult than in the traditional case.

Generally speaking, a DSPL may have many of the following properties [13]:

- dynamic variability: configuration and binding at runtime,
- binding changes several times during its lifetime,
- variation points change during runtime: variation point addition (by extending one variation point),
- deals with unexpected changes(in some limited way),
- deals with changes by users, such as functional or quality requirements,
- context awareness and situation awareness,
- autonomic or self-adaptive properties

Interest in DSPLs is growing as more developers apply the SPL approach to dynamic systems [13]. They represent an efficient way for modelling adaptive capabilities in software product lines. Our brief overview of SPLs pointed out some of the advantages such an approach brings. It seems therefore natural to take advantage of its benefits for developping a service oriented application. The next section tries to present this approach into more detail.

IV. Application for Service-Based Systems design

From what we have seen so far, a convergence of SOA and SPL is highly possible. They both promise to help develop flexible, cost-effective software systems and to support a high level of reuse. They both encourage an organization to reuse existing assets and capabilities rather than redeveloping them again and again for new systems. Organizations can gain enormously on reuse and achieve many desired benefits such as productivity gains, decreased development costs and improved time to market, higher reliability and competitive advantage [16].

Research concerning the possible connections is still in its early days. With this in mind, we try to apply product line practices to our service based platform, EnTiMid . We'll explore the existing connections between SOA and SPL and try to harness the resulting benefits. For this purpose, we introduce a simple application example to illustrate our approach and ideas. In our scenario we make use of black-box services, implemented and provided by different vendors and plugged together in an ad-hoc manner.

We intend to develop a large scale house automation system, for managing several buildings, at a city scale. Several issues arise for designing such an application. First of all, each particular building and actually apartments from the same building, may be equipped differently. So there is a high probability that our system will differ from one building to another or even between apartments. It would be highly recommended not to develop all versions of our system from scratch. An efficient way to manage all these variations in system configuration and also to easily configure the deployment of necessary services is required. Developing all possible configuration variants would prove to be infeasible and also inefficient. Moreover, a certain degree of flexibility is required from the system. It should be able to adapt to changes in user requirements, availability

of resources or quality of services. Consider, for example, a situation in which the default communication service fails. It should be replaced by a functional one. In this way, the system can continue to perform in the desired manner.

Adopting an SPL approach enhanced with some dynamic properties offers solutions for many of the above mentioned issues. A more thorough look at the problem specifications reveals several possible connections between a product line approach and our service oriented platform.

We have noticed that a clear mapping between the two is possible:

- feature $\longmapsto$ atomic service
- product $\longmapsto$ configuration
- product line $\longmapsto$ entire system

We start from the level of basic features of the system, which will be implemented in terms of atomic services. They represent the core assets of the system and will form the basis of the product line. A particular composition of such atomic services is called a configuration. Such a configuration represents a possible product of our product line. Looking from a higher level, the entire system can be thought of as the product line itself.

A major advantage of this approach is that, rather than describing a single software system, our product line will describe the set of products in the same domain. In this way, we don't have to develop all the necessary configurations of our system from scratch. This can be accomplished by distinguishing between elements common to all the products of the line, and elements that may vary from one product to another, and reusing services rather than working from scratch.

Now a domain analysis phase helps capture the commonalities and variabilities amongst different configurations. The system is first modeled in terms of features. A feature is a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems" [17]. Feature models are a popular notation to describe requirements of a software system in such a way that the final user can understand and modify these descriptions. They are hence excellent candidates to perform the elicitation of SPL requirements.

In the case of our system, we distinguish between two types of features: composite and atomic. Atomic features, present at the leaf level of our feature diagram, are directly mapped and implemented by existing services. For implementing a particular atomic feature, we can choose between multiple existing services offering the same general functionality. Several atomic features are grouped together into a composite one. Such a composite feature may also include other composite features. Its role is to offer a new service to the user, not available before, whose functionality is derived from that of the atomic services it encompasses. Finally, a product of our SPL, called a configuration, contains one or more composite services. We insist on a particular restriction that we impose at the configuration level: a configuration will be ultimately decomposed, at the leaf level, into atomic services. This is essential, because we would like to build our application by composing several atomic services, deployed on already existing and functional services.

Variability amongst system features is usually depicted using a feature diagram. It is represented as a tree whose root denotes the complete system and which is progressively decomposed using mandatory, optional, or alternative features. In feature diagrams, mandatory features are those which are always included in every product. Feature that are not necessarily included in every product are qualified as variable. Variation points are features that have at least one direct variable sub-feature. Relations between nodes(features) are materialized by decomposition edges and textual constraints. The later are used to model restrictions imposed to certain configurations. They help express that presence of a certain feature in one product imposes the presence or exclusion of another one. Often, cardinality notations are also used to express variability in a feature diagram. A feature diagram of our system is presented in Figure 2.

The system provides basic security, communication, alarm, agenda and lighting facilities to the users, considered as mandatory features. The energy management functionality is an optional one. All the functionalities at this level are examples of composite features. They offer new facilities to the user, by composing different functionalities provided by atomic features. These atomic features are found at the leaf level. To offer the optional energy management facility for example, information from electricity, heat and water meter services is required. They are also considered as optional, being atomic features used to create an optional composite feature.

Taking a closer look at the application requirements reveals some interesting issues. The atomic feature agenda, is needed for both security and lighting composite features. We model this with the aid of the "require" textual constraint. The communication composite feature is particularly interesting. It can be implemented in one of three ways: GSM, RTC or Internet connection. For a proper functioning of the system, this feature has to be active at all time. For this purpose, the cardinality notation used expresses that 2 out of the 3 possible services that implement it must always be chosen. One is used to actually implement this feature while the other one is used as backup solution in case the first service fails or the quality of the service offered drops below a predefined limit. The security composite feature behaves in a similar way, requiring at least one of camera or open/closed detector to be functional at all time.

But probably the most interesting aspect can be found by examining the alarm feature. It works in the following manner: it takes information from a camera and/or a movement detector; it requires access to an agenda containing information about the people in the building; to send an alert with the picture of the person entering the building, an internet connection and an internet address book are needed. If the internet connection is down and communication is provided as RTC or GSM, a phonebook is needed to transmit the alert. We model this complex functioning and dependencies using textual "require"

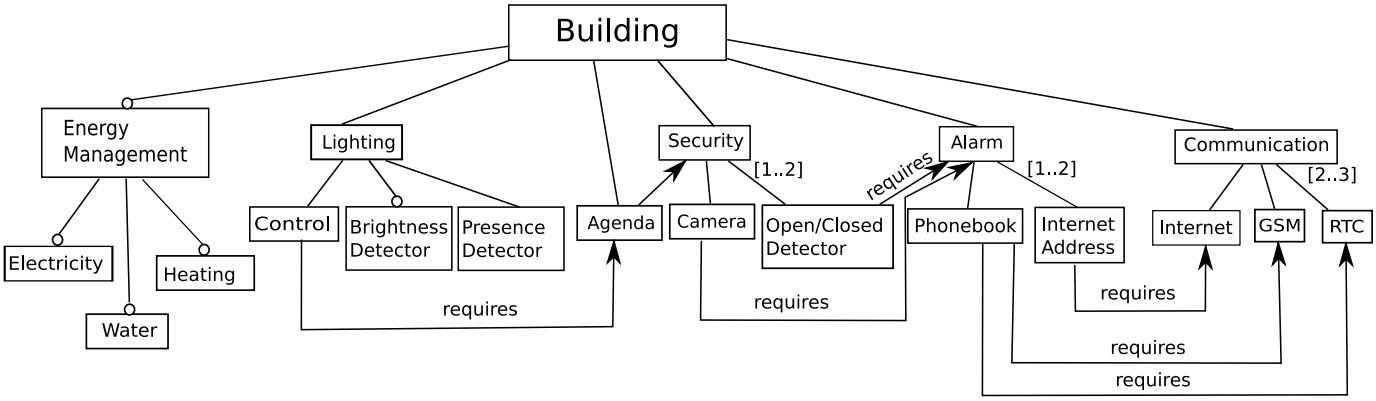

Fig. 2. System feature diagram

constraints. A valid internet connection requires the use of an internet address book. If communication is offered as GSM or RTC, a phonebook is required. Due to limitations in space, we have introduced two "require" constraints between the composite feature alarm and the atomic features camera and open/closed detector. Normally, textual constraints are applied only between atomic features. We notice an interdependency between the alarm feature and the communication one, modeled using textual constraints. The difficulty here is that these connections will change, because the system is adaptive.

V. Challenges

Establishing the connections between SOA and SPL is currently just in its incipient phases. Therefore, applying an SPL approach to a service based platform presents numerous challenges. We try to classify them in two categories, according to the phase in the SPL process they belong to: domain and application engineering.

Concerning domain engineering, the issues arise mostly due to dynamic aspect of the system, which needs to have self adaptive capabilities. In combination with monitoring, it initiates state-triggered reconfigurations to adapt to changes in the environment or in user needs. Most importantly, this has to be done at runtime. For example, when a service stops offering the needed functionality, it has to be replaced by a new one, allowing the system to keep performing normally. Replacement of a service by a new one may also be triggered by quality of service criteria. This change needs to happen without stopping or interrupting system functioning. Service replacement helps therefore enhance system availability and meet agreed QoS standards. This challenge in monitoring and adapting system at runtime joins the work done in the WP-JRA-1.2 of S-Cube[1]. To solve the problem of failing services, the use of a backup service was introduced. A list of such backup services could actually be established, based on different quality and availability factors. The EnTiMid platform offers possible methods and facilities for monitoring the state of the system, determining services that need to be replaced and performing the actual replacement under the appropriate conditions.

In our example application, an interdependency between the alarm feature and the communication one can be noticed. This was modelled using textual constraints. The difficulty here is that these connections will change, due to the adaptive nature of the system. Changes in the environment or user needs will cause the system to pass from one configuration to another. When this happens, the previously described connections will also change. Modeling correctly these dynamic types of connections between different groups of services, the inherent inter-dependencies and restrictions, the way they evolve as the system configuration changes, is a major challenge that needs to be overcome. At this phase, we also need to choose which service to use for implementing an atomic feature. Such a choice depends on service availability, quality of service requirements and user preferences.

The approach needs to continue naturally with the application engineering phase, also known as product derivation. It consists of building configurations, based on the results of the Domain Engineering phase. Based on user requirements, different configurations are assembled from the existing reusable components, the atomic services. Developing a flexible, model driven product derivation technique using Kermeta [18], that addresses the product line's customer specific and unanticipated requirements, is one of the essential steps that we consider in our future work.

VI. Conclusion

Domains like ambient intelligence or house automation have taken advantage of the recent introduction of software solutions in their areas. This has lead to an increase in software complexity, with extensive variations in both requirements and resource constraints. In addition, modern applications require a higher degree of adaptability from their software systems. Developers are pressured to deliver high-quality software with

[1]http://www.s-cube-network.eu/about-s-cube

increased functionality both meeting tight deadlines while minimizing costs. Moreover, environmental conditions, user requirements, and resource constraints can change dynamically during runtime.

We consider that software product lines and service oriented architectures have a lot of common aspects, and that there is a significant advantage in combining these two concepts. That is why we propose to apply dynamic product line practices to facilitate the design of service-based systems. We detailed our approach by means of an example application for managing buildings using ENTIMID . A connection between SPL and SOA concepts comes naturally, and we establish a clear mapping between them. Atomic services are used to represent basic system features. A composition of such services creates a configuration, which is a product of our product line. Then, during a domain analysis phase, we model the requirements of our application in terms of a feature diagram. We distinguish between two types of features: atomic and composite. The advantage of our approach is that the entire system will be ultimately built from atomic features, mapped directly onto a set of existing services. We have also identified several of the challenges implied by the development of such an application. They are mostly due to dynamic aspects of the system.

There is still room for improvement. In particular we will focus on developing a flexible, model driven, product derivation phase for completing the process. Based on the capabilities of the ENTIMID platform, we also intend to offer some solutions for implementing dynamic aspects into our system. Our approach will be validated on a ongoing project carried out by the Rennes city council exploring how home automation systems can improve in-home caring by implementing them in show houses.

ACKNOWLEDGMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 under grant agreement 215483 (S-Cube).

REFERENCES

[1] G. Nain, E. Daubert, O. Barais, and J.-M. Jézéquel, "Using mde to build a schizofrenic middleware for home/building automation," in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, S. B. . Heidelberg, Ed., vol. 5377/2008. Madrid, Spain: IRISA/INRIA/University of Rennes1, 2008, pp. 49–61. [Online]. Available: http://www.irisa.fr/triskell/publis/2008/Nain08a.pdf

[2] M. P. Papazoglou and W.-J. Heuvel, "Service oriented architectures: approaches, technologies and research issues," *The VLDB Journal*, vol. 16, no. 3, pp. 389–415, 2007.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, *Web Services: Concepts, Architecture and Applications*. Springer Verlag, 2004.

[4] L. M. Northrop, "Sei's software product line tenets," *IEEE Softw.*, vol. 19, no. 4, pp. 32–40, 2002.

[5] L. Northrop, "A Framework for Software Product Line Practice," in *Proceedings of the Workshop on Object-Oriented Technology*. Springer-Verlag London, UK, 1999, pp. 365–366.

[6] T. Ziadi and J.-M. Jézéquel, "Software product line engineering with the uml: Deriving products," in *Software Product Lines*, 2006, pp. 557–588.

[7] T. Ziadi, J.-M. Jézéquel, and F. Fondement, "Product line derivation with uml," in *Proceedings Software Variability Management Workshop, Univ. of Groningen Departement of Mathematics and Computing Science*, feb 2003.

[8] G. Perrouin, "Architecting software systems using model transformation and architectural frameworks," Ph.D. dissertation, University of Luxembourg (LASSY) / University of Namur (PReCISE), September 2007.

[9] P.-Y. Schobbens, P. Heymans, and J.-C. Trigaux, "Feature diagrams: A survey and a formal semantics," in *RE '06: Proceedings of the 14th IEEE International Requirements Engineering Conference*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 136–145.

[10] A. Cockburn, *Writing Effective Use Cases*. Addison-Wesley Professional, January 2000.

[11] A. Fantechi, S. Gnesi, I. John, G. Lami, and J. Dörr, "Elicitation of use cases for product lines." in *PFE*, ser. Lecture Notes in Computer Science, F. van der Linden, Ed., vol. 3014. Springer, 2003, pp. 152–167.

[12] C. W. Krueger, "New methods in software product line practice," *Commun. ACM*, vol. 49, no. 12, pp. 37–40, 2006.

[13] S. Hallsteinsen, M. Hinchey, S. Park, and K. Schmid, "Dynamic software product lines," *Computer*, vol. 41, no. 4, pp. 93–95, 2008.

[14] H. Gomaa and M. Hussein, "Dynamic software reconfiguration in software product families," in *PFE*, 2003, pp. 435–444.

[15] J. Andersson and J. Bosch, "Development and use of dynamic product-line architectures," *Software Engineering. IEE Proceedings* -, vol. 152, no. 1, pp. 15–28, Feb. 2005.

[16] R. Krut and S. Cohen, "Service-oriented architectures and software product lines - putting both together," in *SPLC*, 2008, p. 383.

[17] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, "Feature–oriented domain analysis (foda) feasibility study," Software Engineering Institute, Carnegie Mellon University, Tech. Rep. CMU/SEI-90-TR-21, November 1990.

[18] P.-A. Muller, F. Fleurey, and J.-M. Jézéquel, "Weaving executability into object-oriented meta-languages," in *Proc. of MODELS/UML'2005*, ser. LNCS. Jamaica: Springer, 2005.