Towards Agile Composition of Service Oriented Product Lines: A Mashup-based Approach

Ikram Dehmouch, Bouchra El Asri, Zineb Mcharfi
IMS Team, SIME Laboratory
ENSIAS, Mohammed V Rabat University
Rabat, Morocco
{ikram.dehmouch@gmail.com, elasri_b@yahoo.fr, zineb.mcharfi@gmail.com}

***Abstract*—Large scale product lines cover multiple domains with different concepts and concerns. Thus, involving domain users in the development life cycle is a key factor for the success of the composition process combining the different subdomains of the intended resulting large scale system. In fact, domain users master the domain concepts, the scope of each subdomain and the interactions between the different subdomains to be composed. This makes them key actors in the composition process. Adopting agile principles is then required to offer intuitive and simple composition techniques for end-users. One of these emergent techniques is Mashup, which is mainly concerned with web service composition in an ad hoc way. This paper proposes using a Mashup component as an underlying composition technique for large scale service oriented product lines in order to bring agility to the process of composing the different subdomains services. The proposed Mashup component in allows incremental composition to achieve agility and to address scalability issue as well.**

Keywords—Product Line Engineering; Feature Model; Agile Software Development; Service Oriented Computing; Mashup

I. Introduction

The Product line approach has been already successfully applied to various industrial domains, such as avionics [1] and automotive systems [2], etc. With regard to software engineering, Software Product Line Engineering (SPLE) constitutes a major advance, as it allows building software from a set of previously developed and tested parts, based on the domain knowledge. This generates considerable benefits in terms of time, quality and resources [3].

However, traditional SPLE is no more enough to face modern applications, which tend to be cross-industry and to cover multiple domains simultaneously. This has led to the advent of the large scale product lines concept combining various subdomains with heterogeneous crosscutting concerns such as health, telecommunication, transport, etc. Thus, composing these subdomains to generate the intended large scale system is becoming a crucial concern [4]. The composition process in such systems becomes more problematic, since it should scale to big development projects sizes with hundreds of users/developers from several fields.

On the other hand, domain users are an important ingredient of this composition process success, since they master the scope and the different concepts within the subdomains to compose. Thus, one possible way to address the scalability issue is to involve end-users in the composition process.

Consequently, opting for Agile Software Development (ASD) in combination with SPLE is the key to make the composition process simple and intuitive for end-users, solve possible conflicts that may occur in such heterogeneous crosscutting environments and allows incremental development, which meets the scalability issue.

In fact, ASD put end-users at the heart of the software development process, since it is based on constant interaction with customers [5].

In this context, we propose in this article an agile composition approach for large scale product lines based on a consumer-centric technique called “Mashup”. In fact, Mashup is an extremely consumer-centric and lightweight service composition technology [6], which can be exploited to address scalability issue throughout bringing agility to the proposed product line composition approach.

To explore how Mashup facilitates this service composition, this paper is organized as follows: In Section 2, we present some basic concepts related to two main paradigms involved in our work, which are SPLE and ASD. Section 3 draws up the motivations of our work. Section 4 discusses about the related work. Section 5 presents an overview of our approach. A motivating scenario showing the interest of our approach is described in Section 6. Finally, Section 7 concludes the paper.

II. Basic Concepts

- *SPLE*

SPLE is an emergent paradigm which is the result of bringing the reuse-based product line concept, adopted mainly in industry, to software engineering development process [7]. It consists of constructing software products from reusable core assets. This results in lower costs, shorter time-to-market, and higher quality, since a family of products is generated instead of developing them one by one from scratch [8].

According to Kang et al. [9], the product line is defined as a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission, but that still show distinct and different characteristics.

The SPLE process is usually divided in two main complementary phases: *Domain engineering* and *Application engineering* [3]. While the first one deals with the development and maintenance of reusable core or domain assets, the second one is about using those assets in order to build individual software products. The first step in domain engineering is *business scoping* which is performed using specific models called Feature Models (FM). The notion of FM was proposed by Kang et al. [9] to represent commonalities and variabilities among the products within the same domain. In fact, FM is a tree structure representation of features—*“a prominent or distinctive user-visible aspect, quality or characteristic of a software system or systems*—[9] and the relationships between them. A feature can be either mandatory, optional, alternative (Xor group) or part of an Or group. Thus, multiple products can be built form a set of reusable assets depending on which alternative was selected during product configuration. This leads us to a core principle in SPLE, which is Variability [3].

Domain engineering includes also the definition of a *reference architecture*, and the development of *Software reusable components*. At application engineering level, the FMs, reference architecture and software reusable components are then used as a basis for deriving a specific domain application to meet needs of a specific end-user.

Thus, profitable SPLE requires an extensive up-front investment to develop reusable domain assets for later efficient use in products. This carries the risk of not getting a viable return on investment if the pre-developed assets are not sufficiently reused [10]; hence the need to resort to ASD.

- *ASD*

Most of today's systems are evolving towards community-driven development approaches where the end-users are involved in the whole development life cycle. This kind of software approach is known as ASD.

According to Larman [11], ASD is based on short iterations. Each one is a self-contained, mini-project with activities that span requirements analysis, design, implementation, and test. This allows taking into account feedback from users in iteration N so that needed refinements and adaptations are made in iteration N+1. Hence, ASD give mush importance to people and put them at the heart of the development process.

Besides, research has shown that shorter iterations have lower complexity and risk as they are concerned with small fragments of the system, and allow then better feedback, higher productivity and higher success rates [9]. One other major characteristic of ASD is that it is basically built on response to change rather than change prevention [5], which fits the changing nature of software development in which requirements, technology and development team are in constant change.

ASD is based on four fundamental values and twelve principles as presented in The Manifesto for Agile Software Development written by a group of software consultants in 2001 [5]. Most of agile methods such as Extreme Programming (XP) and Scrum [12] share these principles, which are basically about frequent communication, frequent deliveries of working software increments, short iterations and active customer engagement throughout the whole development life-cycle.

III. MOTIVATIONS

The wide scope covered by large scale Software Product Lines (SPL) makes their management a very complex and tedious task. One efficient way of making this task easier and better mastered is the decomposition of these large scale systems in smaller subdomains, each one covering a specific field and involving only business users concerned with this field. To get a final product variant of the large scale product line, the corresponding feature models of the different subdomains should be composed.

This composition process has several limitations when it comes to cross-domain large scale systems. One major limitation is that it would not be obvious to adopt a traditional SPLE approach based on up-front development to assure valid compositions. Thus, scalable product line composition represents a central motivation for our work.

We believe that adopting ASD in this composition process is the key to address the scalability issue. Three arguments motivate our choice:

- ASD allows incremental composition: this meets the modularity logic consisting of decomposing the large scale system into many subdomains and then recomposing them in an incremental way to get a specific product variant.

- ASD is consumer-centric: this allows better management of the different stakeholders regardless of their heterogeneity. Besides, users are involved in the whole development life cycle which reduces the extensive up-front investment.

- ASD and SPLE combination generates many benefits: On one hand, some of the central agile practices may increase flexibility and customer collaboration. On the other hand, the concepts of SPLE are needed in order to manage the diversity of products, the large customer base, and the long-term perspective, which are the characteristics of managing and developing a product line over time.

IV. RELATED WORK

At first sight, ASD and SPLE seem to be contradictory approaches, since SPLE is a proactive approach which requires planning the development of assets in advance for later reuse, in contrast to ASD, which is a reactive approach that avoids up-front planning and development throughout perpetual interactions with end-users.

Though, several experimentations showed that there is a great interest in combining SPLE and ASD approaches [13]. Two case studies driven by Ghanam and Maurer [14][15] show that, besides being practically feasible, the combination of some XP practices and SPLE reduces rework and the cost of producing customized solutions, since it enables customers involvement.

Composite Feature Models (CFM) is another concept combining SPLE and ASD. According to Urliand et al. [16], CFM are an extension to classic Feature models, since these latter are not powerful enough to handle agility challenges. *Separation of concerns* is one of the main pillar on which CFM are built. It offers end-users simple views on the system, since they focus only on their domain concepts without being overwhelmed by the other domain concepts. CFM concept is also based on *bottom-up modeling*. In fact, users have the possibility to change their requirements at any point of the development life cycle. This modification is then introduced in the corresponding partial feature model and an automated algorithm is used to merge the modified partial FM into the CFM [17]. Finally, *automated refactoring* allows CFM handling vocabulary mismatch due to the heterogeneity in face to face conversations with different groups of users, which is an important agility issue.

On the other hand, some other researches show a growing interest in how Service-Oriented Computing (SOC) [24] can be adopted as a mean for enhancing agility and flexibility in SPLE. Kotonya et al. [18] propose a consumer-centered approach combining SPLE and SOC through the two following main steps which are *Feature analysis* for representing different services involved into a family of Business Processes, and *Service analysis* in which dynamic services are selected depending on whether the corresponding features are selected or not in the FM configuration relevant to a specific Business Process (BP). Cubo et al. [19] have also developed DAMASCo framework (model-based service-oriented architecture approach that makes the design, development and deployment of processes more agile) in combination with feature models to safely handle the variability in the service composition at runtime. Thus, if the client request changes, a new valid configuration of the product family containing the required features is automatically created.

Dynamic adaptation is another advantage of combining SPLE and SOC. In fact, Alférez et al. [20] propose a framework that uses variability models to support the dynamic adaptation of service composition. These variability models describe the dynamic configurations of the service composition in terms of activation or deactivation of features. The information captured in these models is combined with context model, which collects context knowledge, and composition model describing the service composition. This combination is performed through the weaving model, which connects the variability model to the composition model based on the context model.

Some other works deal more with the scalability issue in SPLE regardless of the agility aspect such as Dhungana et al.'s work [21], which is about the System of Systems (SoS) paradigm [4] (i.e., systems designed and constructed by combining several heterogeneous subsystems that are themselves composed of many components, data structures and service, etc.) The composition process proposed in this approach is performed through two injection mechanisms *push* and *pull* that allow generating in a flexible way a conjoint model representing a common model of the selected components in the SoS, which can be deployed in a target platform.

Table 1 shows a comparative assessment of works above based on our motivations:

TABLE I. COMPARATIVE ASSESSMENT

Related work	Handling scalability	Handling agility	Adaptability to Service Oriented product line	Dynamic adaptation
[14][15]	-	++	-	-
[16][17]	+	++	-	+
[18][19][20]	-	++	+++	+
[21][4]	+++	-	-	-

V. AN AGILE MASHUP-BASED COMPOSITION APPROACH FOR LARGE SCALE PRODUCT LINES

A. Approach overview

To address the scalability issue in the SPL composition process while taking into account the ASD principles, our approach uses a mix of both SPLE and SOC paradigms. On one hand, SPLE brings a valuable knowledge about variability within the large scale product line throughout the whole development lifecycle. Consequently, late variability is also handled allowing users to specify the services to compose even at runtime. On the other hand, SOC allows loose coupling among interacting services, which enables flexible and agile service composition. Besides, its dynamic nature can be exploited to guide dynamic adaptations at runtime in order to fulfill specific business objectives according to context information especially in terms of availability of dynamic services, i.e., services that can be invoked only at runtime (e.g., real-time information services about current weather).

As our approach is an SPL-based one, it should cover the two main phases of SPLE, which are domain engineering and application engineering.

From a domain engineering view, our approach proposes to use a specific FM notation to distinguish dynamic services from static ones in order to define the scope of dynamic adaptations, which can take place at runtime. To this, features corresponding to dynamic services are represented within a dotted box in the FM. One other major information that should be also captured at this level of SPLE is the standard business workflow of service orchestration. We propose to represent this information using a BPMN 2.0 model [22], as it is a widely used standard for BP definition, not only as a graphical representation, but also as an execution language. Besides, it is a user-friendly model, which consolidates best practices from different modeling techniques such as UML Activity Diagram, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, etc. As covering all possible cases in this generic workflow is a very tedious task, we propose to use FM in order to represent all the variation points of the BPMN 2.0 model using alternative features. Besides, domain engineering includes also the definition of a service oriented reference architecture and the development of reusable BPEL fragments corresponding to the reusable parts of BP.

From an application engineering view, as our contribution is a user-centered approach, it is the end-user who defines the desired product configuration based on his needs. Thus, the generic BPMN 2.0 model is refined according to this specific need, the variation points are resolved, and the corresponding BPEL code and the applicative architecture are generated. In the following section we give more details about how our approach brings agility to the composition process of service oriented product lines.

B. Agility and Mashup

Agility is the central added value of our contribution. To fulfill this, we propose using Mashup as an underlying service composition technique, since it is a lightweight and quick way to integrate multiple sources of applications into a single one, supporting programming for end-consumers without complex environment. In fact, we can take advantage from the Mashup component proposed by Liu et al. [6]. It is composed of three main parts, which are: User Interface (UI) component, Service component and Action component. Adopting this Mashup component allows bringing more agility to our proposed approach through three main principles, which are: Separation of concerns, Dynamic adaptation and Incremental development. Hereafter, we develop each one of these principles:

- Separation of concerns

As the large scale product line is a cross domain system, we propose to decompose it into several subdomains. To emphasize the agility principle, our approach involves end-users from the earlier steps of the development lifecycle. As depicted in Figure 1, each subdomain is represented using a swimlane and it is managed by a group of domain users and experts, as they are the best placed for defining domain feature models configurations and BP instances, as explained in the previous section. It is the UI component of the Mahsup component who offers the end-users a user interface to perform all those definitions and to transmit this information about the services retained and the workflow orchestrating them to the Action component.

Besides being an agile principle, separation of concerns allows also better control of large scale systems, which meets our scalability issue.

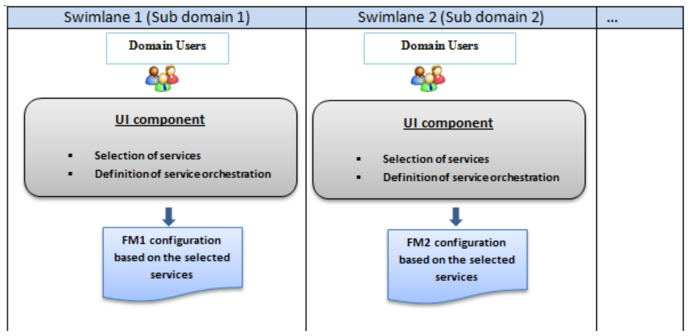
Swimlane 1 (Sub domain 1)
Swimlane 2 (Sub domain 2)
...
Domain Users
UI component
Selection of services
Definition of service orchestration
FM1 configuration based on the selected services
Domain Users
UI component
Selection of services
Definition of service orchestration
FM2 configuration based on the selected services

Figure 1. Separation of concerns.

- Dynamic adaptation

Distinguishing dynamic services from static ones in FM offers our approach the possibility of dynamic adaptation at runtime according to context changes. In fact, in contrast to static services, getting the accurate information about the availability of dynamic services providers can only be performed at runtime. Based on this information, the final variant of the sub-product line (subdomain) is generated.

But, there are several previous steps before achieving this generation:

a. Generation of an FXML file from the FM configuration: Based on the information provided by the UI component about the FM configuration (service selection), an XML file is generated corresponding to the current user's requirements called FXML. In fact, we propose that each feature in the FM configuration has its corresponding XML element. To distinguish dynamic services from static ones in the FXML, we propose to use two kinds of xml tags: <dynamic_service> for dotted boxes in FM configuration and <static_service> for solid ones.

b. Parsing FXML file: At this step, the Action component takes as input the generated FXML file and the BPEL 2.0 code corresponding to the generic BPMN 2.0 subdomain model (developed at implementation phase of domain engineering level). In fact, the Action component parses the FXML file based on a specific mapping relating FXML elements to <invoke> BPEL elements. If a service has its XML element retained in FXML file, then its corresponding invoke BPEL fragment is kept in the final BPEL file, else it is removed. Besides, certain fragments of the generic BPEL might be moved to respect the order required by the end-user. Thus, the Action component defines three actions: *add*, *remove* or *move*, which are used in order to invoke the right services in the right order according to the user's needs. The action component eliminates the variation from the final BPEL. In fact, each variation point is represented by a variable in the generic BPEL. Once the needed service is selected in the FM configuration, the variable is set to the selected value.

c. Checking service provider's availability: Before generating the final BPEL file, the Action component sends a request to the Service component in order to check the service provider availability at runtime. Thus, if the service provider is available then the corresponding BPEL fragment is kept in the final BPEL file else it is removed. Thanks to this, context changes are handled by our approach allowing dynamic adaptations at runtime.

d. Generation of the new variant of the sub-product line: Once the final BPEL file is constructed, it is executed by the service component in a specific execution engine and the result is sent to the Action component. This latter updates the user interface by returning the result to the UI component.

Figure 2 presents the details about the proposed Mashup component and the different steps covered before the generation of the sub-product line variant:

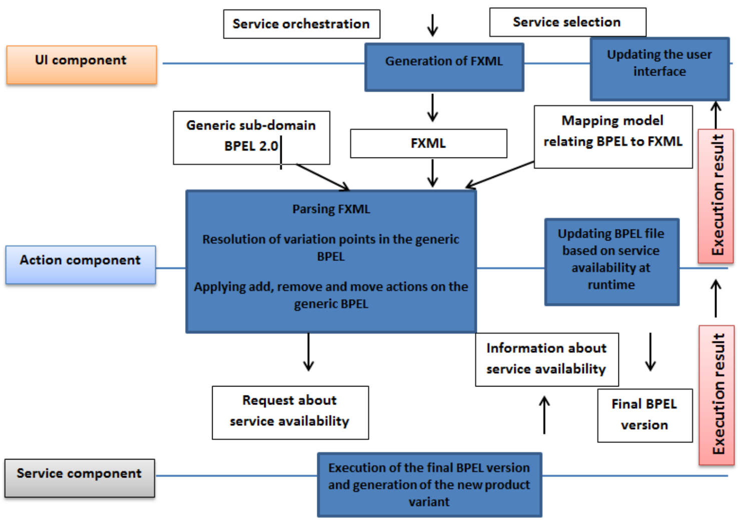
Service orchestration
Service selection
UI component
Generation of FXML
Updating the user interface
Generic sub-domain BPEL 2.0
FXML
Mapping model relating BPEL to FXML
Execution result
Action component
Parsing FXML
Resolution of variation points in the generic BPEL
Applying add, remove and move actions on the generic BPEL
Updating BPEL file based on service availability at runtime
Information about service availability
Request about service availability
Final BPEL version
Execution result
Service component
Execution of the final BPEL version and generation of the new product variant

Figure 2. Intra-domain Mashup component.

- Incremental development

At this stage, we have a product variant at the output of each swimlane. This output is validated by the domain users, since it has been produced according to their definition of BP. We propose that each one of these output is composed with the following one in an incremental composition process until covering all swimlanes, and thus, covering the large scale product line, as depicted in Figure 3.

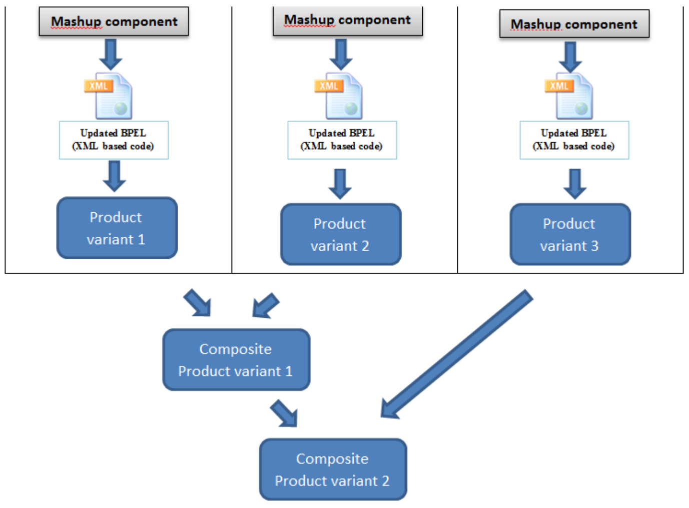
Mashup component
XML
Updated BPEL (XML based code)
Product variant 1
Mashup component
XML
Updated BPEL (XML based code)
Product variant 2
Mashup component
XML
Updated BPEL (XML based code)
Product variant 3
Composite Product variant 1
Composite Product variant 2

Figure 3. Inter-domain Mashup-based composition.

The underlying composition approach in this inter-domain composition phase is the same Mashup component. However, there is a slight difference, which is the adoption of design by contracts [23] as a set of pre- and post-conditions annotating the BPELs 2.0, relevant to the sub-product lines to be composed, at the input of the inter-domain Mashup component. In fact, design by contracts allows defining the interconnection order rules between the different subdomains, as their respective domain users are not intended to know these cross domain rules. The Action component uses these latter to apply the action *move* in order to put the services in the right order in the output composite BPEL 2.0. In fact, if all pre-conditions of a BPEL 2.0 fragment relevant to the first subdomain are fulfilled, the appropriate BPEL 2.0 fragment, relevant to the second subdomain, is invoked and placed at a specific binding point in the resulting composite BPEL 2.0.

VI. MOTIVATING SCENARIO: DIABETES SELF-MANAGEMENT SYSTEM

We choose as a motivating scenario to demonstrate the results of our approach the Diabetes self-management system, i.e., a system allowing diabetes patients to do a regular monitoring of their health and of the different risk factors, which may influence their disease and imply complications.

In the following, we present the three main steps of applying our proposed approach to this example. These steps are:

Separation of concerns: the first step consists of decomposing the Diabetes self-management system in two main subdomains, which are telecommunication domain and health domain managed by patients and doctors respectively. At this level, the UI component offers doctors a specific user interface in order to define the objectives that should be fulfilled throughout the daily treatment and monitoring proposed to their patients based on their criticality degree. These objectives are represented as features in the health subdomain FM such as taking insulin injections or tablets, performing health records (e.g., blood pressure, blood glucose, etc.), walking during thirty minutes, note unusual symptoms, etc. The doctor’s feature selection is then transmitted to the Action component in order to determine which services to invoke and in which order. In fact, once the treatment objectives are selected, their corresponding order can be retrieved from the generic BPMN 2.0 model as it has already been defined by the doctors.

On the other hand, another user interface allows patients to choose the most suited way of interaction with their doctors, (e.g., Telephone, SMS, Interactive Voice Responder (IVR), etc.), as a means of telecommunication. Besides, service orchestration is also possible for patients throughout the definition of the interaction frequency with their doctors via the specified mean of telecommunication. For example, if a patient chooses IVR as mean of interaction, he should define a specific schedule of virtual home visits accomplished via IVR based on his availability.

Dynamic adaptation: at this level, FXML files corresponding to both health and telecommunication subdomains are generated based on the information provided by the UI component. The action component then parses the FXML files in order to update the generic BPELs files. For example: for a specific patient, the daily treatment consists of making a blood glucose record, sending the record result to the doctor in charge, receiving response and applying doctor’s recommendations (i.e., taking a tablet of a specific medicine, walking during twenty minutes, visiting the doctor, etc.), etc. The generic health BPEL is then updated according to this order.

In order to send the appropriate recommendations to the patient, doctors need the accurate values of health records. These values could be transferred instantly to the system via mobile recording devices such as Personal Digital Assistant (PDA). According to our approach, the next step consists of checking the availability of the service, which collects the health records information from the mobile device at runtime in order to generate the right health BPEL.

Incremental development: According to our approach, we have as a result two BPELs, each one corresponding to one subdomain. To generate the composite diabetes self-management variant, we use the pre- and post-conditions annotating the input BPEL fragment at the input of the intra-domain Mashup component. For example, the BPEL fragment corresponding to the IVR services cannot be invoked until the patient notices an unusual symptom; else the system simply sends SMS to remind the patient about medicines and regular health records.

VII. CONCLUSION AND FUTURE WORK

Combining ASD and SPLE has proven to be a worth exploring track as it generates many advantages in terms of reduced time to market and valuable return on investment.

In this paper, we proposed to take advantage from this combination in large scale product lines composition. The main finding was that bringing agility to the composition process throughout the Mashup component ensures the scalability of our composition approach. In fact, the iterative and incremental nature of ASD allows modularity and thus a better control of each sub-system of the large scale system. On the other hand, the user-centric nature of ASD involves only domain users concerned with the appropriate subsystem, which optimizes the time and cost of development.

As users are put at the heart of our agile approach, the main challenge of our future work is dealing with the perpetual changes reflecting the new user requirements. Our future work will then emphasize on the definition of a weaving model relating the FM to the BPMN 2.0 model, in our proposed Mashup component, in order to ensure the repercussion of the new user requirements on the resulting composite product variant.

REFERENCES

[1] F. Dordowsky, R. Bridges and H. Tschope, “Implementing a software product line for a complex avionics system”. In 15th International SoftwareProduct Line Conference (SPLC), IEEE, 2011, pp. 241-250.

[2] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, “A case study in applying a product line approach for car periphery supervision systems”, (No. 2001-01-0025), SAE Technical Paper, 2001.

[3] K. Pohl, G. Böckle, and F. Van Der Linden, “Software product line engineering”. Springer, 10, 2005, pp. 3-540.

[4] M. Jamshidi, “Systems of Systems Engineering: Principles and Applications”.Taylor & Francis, ISBN 9781420065893, 2010.

[5] M. Fowler and J. Highsmith,“The agile manifesto. Software Development”,9(8), 2001, pp. 28-35.

[6] X. Liu, Y. Hui, W. Sun, and H. Liang, “Towards Service Composition Based on Mashup,” 2007 IEEE Congr. Serv. (Services 2007), Jul. 2007, pp. 332–333.

[7] F. J. Linden, K. Schmid, and E. Rommes. "Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering". Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[8] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review", Information Systems, 35(6), 615-636, 2010.

[9] K. C Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study" (No. cmu/sei-90-tr-21). Carnegie-Mellon Univ Pittsburgh, Software Engineering Inst, 1990.

[10] I.S. Jacobs and C.P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G.T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271-350.

[11] C. Larman, "Agile and Iterative Development: A Manager's Guide". Boston: Addison Wesley, 2004.

[12] R. C. Martin, "Agile software development: principles, patterns, and practices". Prentice Hall PTR, 2003.

[13] G. K. Hanssen and T. E. Fægri, "Process fusion: An industrial case study on agile software product line engineering". Journal of Systems and Software, 81(6), 2008, pp. 843-854.

[14] Y. Ghanam and F. Maurer, "Extreme product line engineering: Managing variability and traceability via executable specifications" In Agile Conference, 2009. AGILE'09, August. 2009, pp. 41-48.

[15] Y. Ghanam and F. Maurer, "Extreme Product Line Engineering–Refactoring for Variability: A Test-Driven Approach". In Agile Processes in Software Engineering and Extreme Programming , Springer Berlin Heidelberg, 2010, pp. 43-57.

[16] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser, "Using composite feature models to support agile software product line evolution". InProceedings of the 6th International Workshop on Models and Evolution, ACM, October.2012, pp. 21-26.

[17] M. Acher, P. Collet, P. Lahire, and R. B. France, "Separation of concerns in feature modeling: support and applications". In Proceedings of the 11th annual international conference on Aspect-oriented Software Development, ACM, March. 2012, pp. 1-12.

[18] G. Kotonya, J. Lee, and D. Robinson, "A consumer-centred approach for service-oriented product line development," 2009 Jt. Work. IEEE/IFIP Conf. Softw. Archit. Eur. Conf. Softw. Archit., September. 2009, pp. 211–220.

[19] J. Cubo, N. Gamez, L. Fuentes, and E. Pimentel, "Composition and Self-Adaptation of Service-Based Systems with Feature Models", In Safe and Secure Software Reuse, Springer Berlin Heidelberg, 2013, pp. 326-342.

[20] G. H. Alférez, V. Pelechano, R. Mazo, C. Salinesi, and D. Diaz, "Dynamic adaptation of service compositions with variability models", J. Syst. Softw., vol. 91, May. 2014, pp. 24–47.

[21] D. Dhungana, A. Falkner, and Haselböck, "Generation of conjoint domain models for system-of-systems". In Proceedings of the 12th international conference on Generative programming: concepts & experiences, ACM, October. 2013, pp. 159-168.

[22] T. Allweyer, "BPMN 2.0: introduction to the standard for busines process modeling". BoD–Books on Demand, 2010.

[23] T. Thüm, I. Schaefer, M. Kuhlemann, S. Apel, and G. Saake, "Applying Design by Contract to Feature-Oriented Programming". In Proceedings of the International Conference on Fundamental Approaches to Software Engineering (FASE), Springer, 2012, pp 255–269.

[24] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles". Internet Computing, IEEE, 9(1), 2005, pp. 75-81.