

SPLEMMMA: A Generic Framework for Controlled-Evolution of Software Product Lines

Daniel Romero*, Simon Urli†, Clément Quinton*,
Mireille Blay-Fornarino†, Philippe Collet†, Laurence Duchien*, Sébastien Mosser†
*ADAM Project-team, Université Lille 1, LIFL (UMR CNRS 8022), INRIA Lille-Nord Europe, France
{first.last}@inria.fr
†MODALIS team, Université Nice Sophia Antipolis, I3S (UMR CNRS 7271), France
{urli,blay,collet,mosser}@i3s.unice.fr

ABSTRACT

Managing in a generic way the evolution process of feature-oriented *Software Product Lines* (SPLs) is complex due to the number of elements that are impacted and the heterogeneity of the SPLs regarding artifacts used to define them. Existing work presents specific approaches to manage the evolution of SPLs in terms of such artifacts, *i.e.*, assets, feature models and relation definitions. Moreover stakeholders do not necessarily master all the knowledge of the SPL making its evolution difficult and error-prone without a proper tool support. In order to deal with these issues, we introduce SPLEMMMA, a generic framework that follows a *Model Driven Engineering* approach to capture the evolution of a SPL independently of the kind of assets, technologies or feature models used for the product derivation. Authorized changes are described by the SPL *maintainer* and captured in a model used to generate tools that guide the evolution process and preserve the consistency of the whole SPL. We report on the application of our approach on two SPLs: YOURCAST for digital signage systems, and SALOON, which enables generation of configurations for cloud providers.

1. INTRODUCTION

In recent years, the evolution process of feature-oriented *Software Product Lines* (SPL)¹ has gained the attention of several research works due to the need of improving the derived software to meet the changing business requirements, to use new or evolving technologies or to integrate new functionalities. This evolution process, as the evolution of software systems, requires the consideration of several issues. The verification of *Feature Models* (FM)², assets and mappings (*i.e.*, links between features and assets) consistency and their co-evolution, as well as the control of evolution

impact on the set of products, and the heterogeneity management of SPL solutions are some of these issues [3, 6]. These issues make evolution complex and difficult to carry out in a controlled and automated way.

In the literature, several works analyse and tackle some of these issues. For example, Borba *et al.* [5] and Thüm *et al.* [19] consider the impact on the variability of the SPL by reasoning on feature models. Other approaches study the co-evolution of different elements of the SPL such as feature models, assets and mappings [11, 17] and even the derived products themselves [8]. Some work also focuses on versioning the assets, feature models and architecture of the SPL in order to deal with maintenance of derived products [12, 18]. Nevertheless, all these approaches are not generic enough to be easily used in the evolution of any SPL.

As a matter of fact, feature-oriented SPLs are heterogeneous regarding their assets, mappings and FMs. Even if all SPLs are based on common principles and building blocks, how each SPL is concretely made depends on the requirements on derived products. Moreover, evolution operations depend on the SPL domains and the kind of stakeholders. The evolution can be made by hand when few actors hold all the domain knowledge of a small SPL. However, this activity is much more difficult when the scale of the SPL increases, involving many different contributors as well as a large amount of operations with different granularities and impact depending on their knowledge level. Even if some tools [17, 19] propose a partial help with some aspects of evolution, the lack of genericity and support for evolution operations increases the burden of evolving feature-oriented SPL.

In this paper we introduce SPLEMMMA³, a framework for dealing with the evolution in feature-oriented SPLs. SPLEMMMA enables the validation of controlled SPL evolution by following a *Model Driven Engineering* (MDE) approach. By controlled, we mean that a SPL *maintainer* defines the authorized operations of evolution. Such definition is made through an evolution model enabling the generation of evolution tools. These tools automate the evolution process, *i.e.* the usage of generic services for validation, as well as the orchestration of the SPL specific services to execute the actual evolution operations. SPL *contributors*, who do not master the whole SPL, will use such tools to improve the SPL. Furthermore, SPLEMMMA is generic and domain-independent,

¹A feature-oriented SPL is composed by a set of feature modules (or code units implementing requirements) along with a feature model [1].

²Our work focuses on variability descriptions based on feature models [10].

³SPLE for *Software Product Line Evolution* and MMA for Emma Darwin, Charles Darwin's wife.

i.e., the framework is independent of the SPL implementation and therefore of *Feature Metamodels* (FMM), assets and mappings but still can be specialized regarding the requirements of the SPL in terms of evolution.

The rest of this paper is structured as follows. Section 2 provides background concepts and motivates our work through two case studies. In Section 3 we introduce the requirements related to SPL evolution. We continue in Section 4 with the presentation of SPLEMMMA, our approach for evolution. Section 5 describes our experience of applying SPLEMMMA in the case studies while Section 6 provides an overview of existing approaches dealing with SPL evolution. In Section 7 we discuss the advantages of SPLEMMMA before concluding and presenting the perspectives of our work in Section 8.

2. BACKGROUND AND CASE STUDIES

In this section we introduce our context of work and present two case studies to motivate our needs in SPL evolution.

2.1 On SPL Evolution

In order to define our context of work, we identify *who* participates in SPL evolution and we adapt to SPLs the taxonomy defined by Buckley *et al.* [6] related to the *where*, *what* and *how* of software evolution.

Regarding actors (*who*), we identify three of them: (i) SPL *maintainer* that manages SPLs, which includes the control of the evolution process to avoid inconsistency; (ii) SPL *contributor* performing the actual evolution operations; (iii) SPL *user* that configures products by using the tool support of the SPL.

Concerning the system properties (*what*), we assume that the SPL tools will be unavailable during an evolution. This means that products will not be derived when executing the changes. Moreover, all evolutions will be driven externally, responding to a user request, and the safety of the evolution must be ensured statically to avoid inconsistency. However the openness (*i.e.*, the planned capacity to evolve) of a SPL is not defined in a generic way. SPL *maintainers* should anticipate possible evolutions. In the same way, SPL *maintainers* also define the objects of change (*where*):

- *Artifacts* are evolvable elements. In feature-oriented SPLs such elements include FMs, assets and mappings.
- *Granularity* refers to the scale of evolution elements to be changed. The granularity can be coarse, medium or fine. In feature models, a coarse granularity refers to a general modification of the whole FM. A medium granularity is expressed in terms of change on specific features and a fine granularity enables modifications on attributes of features.
- *Impact* is related to the range of elements, which will be impacted by an evolution. An impact can be local (*e.g.*, one asset of the SPL is replaced) or global (*e.g.*, the structure of a feature model is modified affecting mappings and assets).

Finally, change support (*how*) refers to different mechanisms making the execution of the evolution easier. The framework defined in Section 4 gives such support.

2.2 YourCast

YOURCAST [22] is a system designed to support the definition of customized digital signage system [7]. To do that, the SPL captures the variability of such systems, and provides a user-friendly way of configuration. The project is funded as an industrial transfer and involves 2 SMEs interested in the automatic generation of digital signage systems applied to large associations meetings (*e.g.*, *Choralies*, an event involving 4,000 choir singers during 11 days), conferences, or sport events (*e.g.*, *Tour de France*).

There is a lot of variability to be managed to create a coherent digital signage system adapted to users' needs. YOURCAST targets non-specialist users (*e.g.*, association member) that have to select information sources implemented through common Internet services (*e.g.*, RSS feeds, Twitter), screens look and feel, and information rendering mechanisms.

Besides, YOURCAST aims to create an open community of users and developers. Currently, new Internet services are constantly created due to the expansion of Web 2.0. People want to use these services in digital signage systems through adapted rendering mechanisms they define. Then, we need to give the opportunity for everyone to contribute to the SPL by adding new assets, updating the feature model and changing mappings between assets and configurations. That is why *contributors* and *maintainers* are distinct in YOURCAST. *Contributors* are allowed to update and add products, enriching the family of products. *Maintainers* can delete some products to keep a consistent line regarding the available sources. Thus, a framework to support evolution in the YOURCAST SPL, exposing specific operations, is really a need since the community will drive the changes without having all the knowledge of the SPL.

2.3 SALOON

SALOON [14] is a SPL to manage and derive cloud *Platform-as-a-Service* (PaaS) configurations. The automated deployment of applications on PaaS has become very trendy. This deployment discharges application owners from dealing with virtual machine configuration, by providing application support and hiding the cloud infrastructure. However, the selection of a suitable PaaS to host an application remains complicated. Indeed, the wide range of available PaaS providers combined with the lack of visibility among them make their selection a real challenge.

In order to address this challenge, SALOON relies on ontology and feature modeling to handle cloud platforms heterogeneity and variability. The ontology captures the domain knowledge, in this case the PaaS provider offers, while the FMs are used to reason about configurations in the framework. Each FM represents a cloud PaaS provider and the set of configurations related to this PaaS. A semantic mapping is defined between ontology concepts and FM features. This mapping allows users to choose their application technical requirements only once in the configuration process, thus avoiding a tedious and error-prone selection of such requirements for each PaaS provider.

Because of constant changes of technologies and to stay alive in the market, offers from PaaS providers evolve and new ones appear. *Contributors* of SALOON, *i.e.*, cloud experts,

need to update information and include new offers in order to enable SALOON's *users* to benefit of them. However, the ontology should not be modified because of restrictions on the SPL definition. Thus, a framework that enables a controlled SPL evolution is required.

3. REQUIREMENTS

Evolution in SPLs has been a challenge for many years [13]. Though several works exist on feature models evolution [3, 9], we identify three challenges to provide a generic evolution framework able to guarantee some properties: 1) *how to ensure SPL consistency during evolution*, 2) *how to control the impact on the family of products* and 3) *how to deal with SPL heterogeneity*.

3.1 Ensuring SPL consistency

In order to ensure the consistency of a SPL, the correctness and co-evolution of impacted elements have to be checked:

1. **Assets' consistency.** When assets are added or updated, they should behave properly. In general, the definition of a correct behaviour depends on each SPL. Therefore, according to its domain, a SPL will define suitable tests to verify the behavior of its assets.

As YOURCAST evolution is driven by a community, each new asset has to be automatically tested before a complete integration in the SPL. Currently, as soon as we let anyone contributing, we have to ensure the correctness of the code through continuous integration services. In SALOON, tests on assets are lighter since contributors are the cloud experts and each PaaS defines its own set of fixed instructions (which are the assets in SALOON) for its configuration.

2. **FM consistency.** In the literature, several operations are defined to anomaly detection such as void FM, dead features, false optional features, wrong cardinalities and redundancies [3]. Besides these anomalies, we can find problems related to the concrete domain of SPLs. For example, in YOURCAST, the addition of a new configuration can introduce features that are semantically equivalent. The addition of a new service for weather predictions by using a *forecast* feature instead of a *weather* feature will cause the creation of two different features with the same semantics. In SALOON, the inclusion of a new cloud provider can lead to repetition of an existing one if a different name or identifier is specified for the provider being added. Thus, verifications using existing research can be systematically used on each SPL but specialized verifications will be also required according to the domain.
3. **Mapping consistency.** Modifications on mappings also require checks. If SPLs define a metamodel for mappings, it is used to make structural verifications. In the contrary case, mappings have to include at least two elements of different type. Depending on the SPL, it is also required to verify that mappings do not exist.

In YOURCAST each asset is related to a given configuration of the FM. Mappings are defined through a mapping model. Because we have a lot of possible configurations, the mapping definition is an error-prone task without a proper tool. Each time that a mapping is added or updated, it is necessary to check if the configuration is not already related to another mapping.

The semantic mapping in SALOON requires the definition of relations between features and concepts in the ontology. Mappings are also based on a mapping metamodel, but their definition is difficult as the ontology has several concepts.

4. **Co-Evolution** Several studies [11, 19] confirm that changes on evolvable elements are not isolated. Then, changes on elements from the *Problem Space* (PS), containing feature models, can also need modifications on the *Configuration Knowledge* (CK) or mappings and the *Solution Space* (SS) that has the assets [10]. An evolution can then be *Intraspatial* (only elements of the same space are modified), *Interspatial First Degree* (the mapping and one of the spaces are modified) or *Interspatial Second Degree* (the mapping and both spaces are modified) [17]. Therefore, the co-evolution for different element of SPLs is required.

In YOURCAST and SALOON, co-evolution has to be guaranteed when a new product is added: a user has to provide both the new assets and the new configuration. Then, several validations have to be done (*e.g.*, checking assets consistency, checking updated feature model consistency, checking mappings) in a transactional way to keep the consistency of the SPL. If one of the validation fails, all the evolution should be cancelled and problems notified to contributors.

3.2 Controlling the impact on the SPL

In order to control the impact, we need first to identify it. According to the literature, an evolution can produce four kinds of impact on the family of products [19]: *generalization*, *specialization*, *refactoring* and *other*. In *generalizations*, we can still derive the set of original products but we add new ones. On the contrary, in *specializations* we can only derive a subset of the original products. *Refactorings* improve the cohesion between SPL elements without changing the product family. *Other* impacts should be avoided. and should be avoided. The control of impact refers to the automatic detection of evolution kind, preventing regressions during the evolution.

In YOURCAST, *contributors* are only authorized to make *generalizations*. This means that we only allow contributors to define new configurations opening the possibility of deriving new digital signage systems but we are still able to derive the original ones. In contrast, SPL *maintainers* are able to specialize and refactor the SPL family of products. *Specializations* are required when, for example, a source is no longer accessible. *Maintainers* also need to execute *refactorings* when the architecture of the SPL has to evolve or for simple maintenance reasons (*e.g.*, to add a missing feature in the feature model).

In a similar way, SALOON allows for *generalizations* and *specializations* of the SPL. *Generalizations* are needed when new PaaS are included in the SPL. *Specializations* are due to changes on cloud provider offers. Thus, SPL *maintainers* should control the impact on the variability.

3.3 Dealing with SPL heterogeneity

The previous issues illustrate the complexity of the evolution process in SPLs. An approach automating and making this

process easier is required. However SPLs are heterogeneous regarding the elements used to build them.

In a SPL, the nature of assets depends on its domain. YOURCAST has *Web application Archives* (WAR) files, javascript files and images while we find script fragments in SALOON.

In a similar way, each SPL can have its own mechanisms to express relationships between assets and decisions made on feature models. Indeed, our both case studies use different mapping metamodels.

Finally, there is no standard *Feature Metamodel* (FMM). SALOON employs a FMM with attributes and cardinalities while YOURCAST uses one without them. Thus, the SPLs are developed in different ways, which make the definition of an approach controlling the evolution process difficult.

Towards a generic approach. The analysis of the evolution in heterogeneous SPLs leads us to find commonalities to propose a generic approach. In particular, types of evolution operation in any SPL can be additions, removals or updates. Such operations produce a generalization, specialization or refactoring of the SPL (cf. Section 3.2) and consequently certain elements are expected to execute them. Furthermore, structural validations can be easily made by using FMMs and mapping metamodels. In a similar way, FM consistency can be ensured through existing algorithms (cf. Section 3). In the next section we use this commonality to introduce our approach to deal with *heterogeneity*, *impact* and *SPL consistency* related to the evolution process.

4. THE SPLEMMMA FRAMEWORK

In this section we introduce SPLEMMMA, a generic approach for the evolution of SPLs. The main objective of SPLEMMMA is the definition and execution of the evolution by keeping the consistency of the whole SPL. To do that, the approach supports basic validations on FMMs, mappings and assets as well as the evolution itself. The evolution process is automatized and controlled through a tool, generated from models specifying the authorized changes on the SPL.

4.1 SPLEMMMA Overview

SPLEMMMA is a framework that follows MDE principles for the generation of tools to guide a controlled evolution of SPLs. By controlled we mean that *SPL maintainers* decide about changes to be done by *SPL contributors*. Figure 1 depicts an overview of the process to generate evolution tools. The evolution of the SPL is defined by *SPL maintainers* through an *Evolution Model* (cf. Section 4.2). SPLEMMMA validates explicit and implicit constraints of such models (cf. step 1 in Figure 1) before continuing with the generation process (cf. Section 4.3). The *SPL maintainer* then selects the generic *Validation Services* (cf. Section 4.4) used by the *Evolution Manager*, which enable the verification of preconditions and postconditions on evolution. *Maintainers* also indicate specific services of the SPL to define specialized preconditions and postconditions. Generic validation services are held by the *SPLEMMMA Store*, which allows the extension of the framework through the addition of new services.

The *Model Checker* and *Evolution Manager* (cf. Section 4.4) are tools generated (step 3 and 4) and compiled (step 5) by

SPLEMMMA to support a controlled evolution. The former checks the validity of feature and mapping models by using the *FMM* and *MappingMM*. The latter enables *SPL contributors* to define a concrete evolution as well as its orchestration and execution through the use of *Maintenance Services* provided by the *SPL maintainer*.

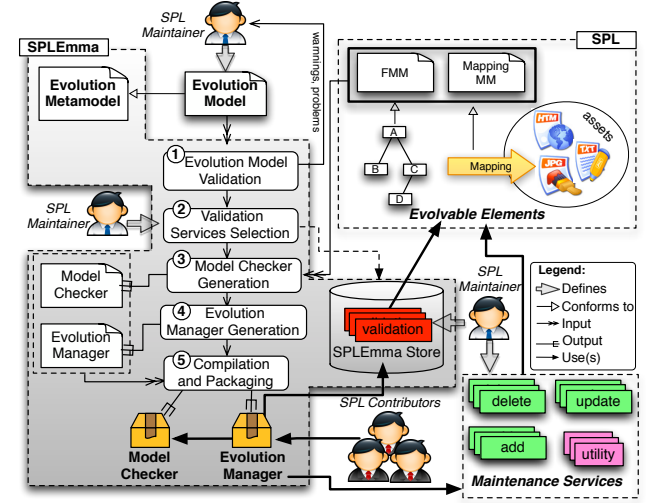


Figure 1: SPLEMMMA generation process overview

4.2 Evolution Metamodel

The keystone of our framework is the model reifying the evolution of the SPL. Figures 2, 3 and 4 depict the metamodel used to define such model. This metamodel includes the different elements required to characterize a feature-oriented SPL as well as its evolution. Such metamodel is independent of domains and technologies related to SPLs in order to deal with their heterogeneity (cf. section 3.3). In particular, the SPLEMMMA metamodel captures the *how* and *where* of SPL evolution (cf. Section 2.1). The *how* includes operations, which are atomic and independent from each other. We define the evolution in this way to simplify its execution and control. The *where* are concrete elements that represent evolvable elements. These elements have a type and can be modified through an evolution. Therefore, an evolution model is composed by a set of operations and different elements of the SPL reifying evolvable elements.

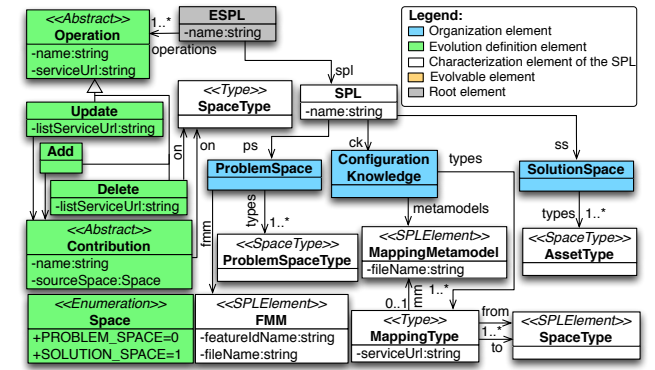


Figure 2: Evolution Metamodel in SPLEMMMA

Each operation is applied on a *SpaceType* such as *AssetType* or *ProblemSpaceType* (i.e., a *FeatureModelType*, *Fea-*

tureType or AttributeType in Figure 3). These types enable the expression of different granularity levels in evolution (cf. Section 2.1). The operations that can be defined are *delete*, *update* and *add*. The last two require a *Contribution* element, which defines the type of elements expected according to the evolution kind (cf. Section 3.1): *IntraSpatial*, *InterSpatialFD* (InterSpatial First Degree), and *InterSpatialSD* (InterSpatial Second Degree). The definition of *Contributions* in this way allows us to provide a basic validation of evolution by controlling its impact. Figure 4 depicts *Contributions* in SPLEmma.

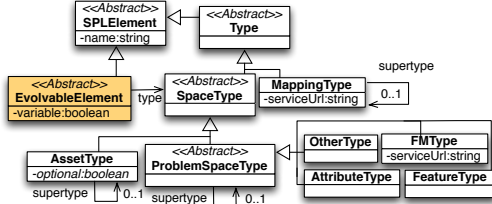


Figure 3: Types in SPLEmma Metamodel

In our metamodel, *EvolvableElements* are concrete elements of the SPL that can be impacted by *Update* and *Delete* operations. These elements have a *SpaceType* (cf. Figure 3) according to the category to which they belong: PS, CK and SS (cf. section 3.1). The PS has *FeatureModels*, their *Features* and *Attributes*. In the PS also is the *FeatureMeta-model* used by the SPL. The *Other* element is an evolvable element for the definition of other elements that are part of the PS such as the ontology in SALOON.

The CK contains the *MappingTypes* and *Mapping Metamodels* (if they exist) defined by the SPL. These types describe the types of elements involved in the mapping and the metamodel that they should respect, which are used in our framework to validate *InterSpatial Contributions*. Each *MappingType* has a *serviceUri* attribute that defines the relation towards an *utility* service (cf. Section 4.4), for checking the existence of a mapping when *Add* or *Update* operations are executed. Finally, in the SS we find assets and their types.

In an evolution model, all the SPL elements that are not evolvable elements (e.g., types or metamodels) can not be changed by an evolution. The SPL *maintainer* decides about their modifications when required, and changes the evolution model accordingly. Furthermore, an evolution model does not include all the elements belonging to the SPL. Only evolvable elements that can not be modified freely have to be specified.

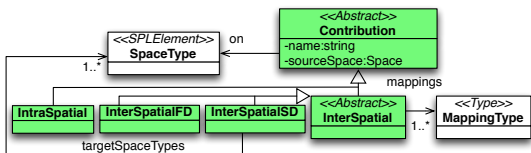


Figure 4: Contributions in SPLEmma Metamodel

4.3 Validations on Evolution Models

As already said, SPLEmma uses evolution models in order to provide tools for evolution automatization. However, in order to generate such tools, SPLEmma validates the explicit and implicit constraints on the model. Explicit constraints are defined by the relationships in the metamodel

and validated by verifying the “conforms to” relation. On the other hand, implicit constraints depend on *Contributions* and *Types* of evolvable elements in the model:

- *Contribution* checking: the *SpaceType* of the contribution (cf. *on* relationship between Contribution and SpaceType in Figure 4) belongs to the *sourceSpace*.
- *Interspatial* contribution checking: mappings in Interspatial contributions include the type defined by *on* as source or target types. In the same way, the *from* (resp. *to*) types in a mapping belong to the same space.
- *InterspatialSD* contribution checking: the *targetSpaceTypes* should belong to the opposite space of *sourceSpace*. Additionally, at least one mapping has to involve types from both spaces.

4.4 Generated Tools

The *Evolution Manager* tool (cf. Figure 1) enables SPL *contributors* to execute a controlled evolution. In particular, the Evolution Manager performs the following tasks: 1) contribution creation, 2) contribution validation, 3) validation of preconditions for the evolution, 4) operation execution, and 5) validation of postconditions for the operation.

The contributions are created by respecting the restrictions of the evolution model. Contribution validations supported by the *Evolution Manager* include the satisfaction of the “conforms to” relation by feature models and mapping models through the *Model Checker*, and the containment of the suitable mappings and elements by contributions. This last validation provides a basic control on the structural impact.

The validation of preconditions depends on the selected *Validation Services* and specific services of the SPL when the tools were generated (cf. Section 4.1). If at least one precondition is not respected, the operation is not executed. Once the *Evolution Manager* validates the preconditions, it proceeds to the operation execution through the *Maintenance Services*. Then, the postconditions are validated and results presented to the *contributor*. The non-selection of *Validation services* means that there are no preconditions and/or postconditions. This can happen because the *maintainer* considers that they are not required or because there are no compatible *Validation Services* with the SPL.

Maintenance Services. These services, developed by the SPL *maintainer*, define the semantics of operations (or evolutions) on the SPL. Each operation in the evolution model is related to an *evolution service* (i.e., a *Delete*, *Update* or *Add* service in Figure 1). *Maintenance Services* also include *utility* services, which provide information related to evolvable elements for the definition of update and delete operations as well as mapping and product existence verification.

SPLemma Store. The *Store* allows SPL *maintainers* to enrich SPLEmma through the incorporation of new *Validation Services* and it fosters the reuse of well-know verification mechanism in literature. Such verifications mainly focus on basic semantic checks on FMs including the presence of dead features, false optional features, wrong cardinalities and redundancies [3] but also the relationships between two FMs (i.e., generalization, specialization or refactoring) to identify the impact on SPL variability. These verifications deal with

the consistency of the SPL (cf. section 3). The usability of different *Validation Services* on evolution for a specific SPL is checked by SPLEMMMA when such services are chosen by the *maintainer*. Such checking process consists in determining if the FMM used by *Validation Services* are the same as the one used by the SPL. The *Validation Services* are used to define the preconditions and postconditions of evolution.

4.5 Evolution Process Summary

The automation of SPL evolution through SPLEMMMA is summarized as follows:

1. **Definition of the evolution model by SPL maintainer.** This model reifies the openness degree of the SPL and therefore the control *maintainers* keep on evolution operations.
2. **Generation of tools by SPLEMMMA.** The SPLEMMMA framework uses the Evolution Model to generate an *Evolution Manager* and *Model Checker* enabling the automatization of the evolution (cf. Figure 1).
3. **Definition of a concrete evolution by SPL contributors.** Through the *Evolution Manager*, contributors define an operation and its contribution according to changes on the business domain. A concrete evolution is an operation allowed by the evolution model.
4. **Validation of contribution and preconditions by Evolution Manager.** The contribution is analysed and checked by regarding the constraints from the Evolution Model. If there are preconditions, they are evaluated through the *Validation Services*.
5. **Execution of the evolution by Evolution Manager.** The Evolution Manager uses the *Maintenance Services* of the SPL to perform the concrete evolution.
6. **Validation of postconditions by Evolution Manager.** If there are postconditions, they are evaluated by calling the related *Validation Services*.

5. IMPLEMENTATION AND APPLICATION TO CASE STUDIES

In this section, we present the details of the tool support for our approach as well as its use in both case studies.

Tool support. SPLEMMMA provides a standalone tool based on Java and Eclipse EMF⁴. The tool, called SPLEMMMA *Generator*, allows SPL *maintainers* to generate evolution tools for their SPLs. Figure 5 depicts the architecture of the SPLEMMMA *Generator*. The architecture modularizes the different tasks in SPLEMMMA, *i.e.*, checking, generation, compilation and service management (cf. Figure 1). Such modularization fosters reuse and makes modification on different tasks easier.

The **Evolution Orchestrator** guides the generation process depicted by Figure 1. The **Evolution Checker** verifies the implicit constraints of the evolution model as well as the usability of *Validation Services* by regarding the metamodels of the SPL. The **Manager Generator** and **Model Checker Generator** enable the *Evolution Manager* and *Model Checker* generation, respectively. The **Manager Compiler** and **Model Checker Compiler** support the compilation and packaging of generated tools. The SPLEMMMA *Store* provides functionality to manage *Validation Services*.

⁴Eclipse EMF: <http://www.eclipse.org/modeling/emf/>

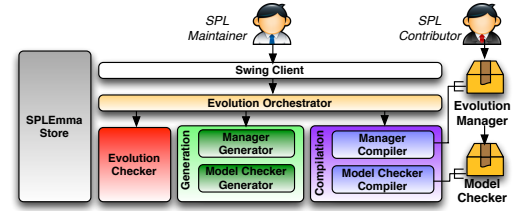


Figure 5: SPLEMMMA implementation overview

Application to YourCast. In YOURCAST two different evolution models are needed: a first to express the authorized evolution for *contributors* and a second one for SPL *maintainers*. This stresses the fact that evolving the SPL is an error-prone and tedious activity, even for *maintainers*.

In the YOURCAST community, recurrent needs are the use of new information sources and information rendering mechanisms (called renderers). An information source is related to a WAR file and an image. On the other hand, a renderer is a javascript function for a kind of information.

Based on this information, the YOURCAST evolution model for *contributors* contains add operations for sources and renderers. The *add source* operation requires an *InterSpatialSD Contribution*. For each source addition expected elements are the FM representing the configuration, a war file, an image and a mapping between the configuration and assets. The *add renderer* operation needs an *InterSpatialSD Contribution*. Expected elements are the FM representing the configuration, a javascript file containing the renderer function and a mapping between the configuration and asset.

The evolution model for SPL *maintainer* provides the previous operations and also operations to update the variability description of sources and delete sources. The former requires an *InterSpatialFD Contribution* and is used, for example, to rename a feature. In this case, the asset remains the same but the feature model and all mappings involved (possibly a large amount) need to be updated. The latter operation can be used if an external source has been shut down. The corresponding assets are removed as well as the related mappings and features (in the FM).

Validations provided by SPLEMMMA include structural verifications on FMs and mappings (YOURCAST defines a metamodel for the definition of relationships) as well as correct construction of *Contributions*. As the previous evolutions involve FM changes, we use generic preconditions related to FM consistency (*e.g.*, detection of dead features and void FM). Moreover, we define specialized preconditions for the addition of sources and renderers. Such preconditions execute tests to check the reliability of the assets being added. For the add evolution, a postcondition is specified, which verifies that they always produce a generalization of the SPL.

Application to SALOON. In SALOON required evolutions are additions and updates of PaaS providers. Each provider is reified by a FM and a set of parametrisable instructions enabling its configuration. Mapping between ontology concepts and features is done through a dedicated mapping metamodel. Thus, the specified operations in SALOON evolution model for cloud experts or *contributors* are the addition and update of providers. For each addi-

tion expected elements are a FM, a set of instructions, and a mapping between features and ontology concepts. The mapping between instructions and FM is defined with an attribute in features. Each update includes a FM, and new instructions and/or ontology concept mappings for features being modified or added.

Structural validations and generic preconditions are the same used by the YOURCAST evolution. However, we also include an additional generic precondition related to FM with cardinalities such as the wrong cardinality detection. For both, add and update operations, we define a specific precondition to verify that mappings towards the ontology use existing concepts. In SALOON case, no postcondition is specified.

6. RELATED WORK

In this section we present existing works by grouping them according to three requirements introduced by section 3.

SPL consistency. In the literature, some works face SPL consistency in evolution by focusing on FMs consistency (cf. [2, 4, 15, 20]). Batory [2] proposes an approach to integrate FMs, grammars and propositional formulas in order to debug FMs through satisfiability solvers. Benavides *et al.* [4] analysis the usage of constraint programming to provide a base for an automated reasoning on extended FMs. Tools as FEATUREIDE [20] and VMWare [15] provide automated reasoning on FMs. All these works provide valuable mechanisms to detect the already mentioned FM anomalies. In SPLEMMMA, we benefit from them to provide basic validations on evolution through the definition of preconditions.

Other approaches deal or analyze the co-evolution of evolvable elements (cf. [5, 17]). Borba *et al.* introduce templates to guide the co-evolution [5]. The approach presented by Seid *et al.* considers the co-evolution of PS and SS and define remapping operators [17]. With our approach we do not redefine or improve these approaches but rather we focus on ensuring the presence of required elements according to the kind of evolution (*i.e.*, intraspatial or interspatial). By doing this, SPLEMMMA provides a first validation mechanism for the consistency of the co-evolution.

Impact in the SPL family of products. Existing works propose algorithms to determine impact of the evolution on SPL product family [5, 19]. In particular, Borba *et al.* [5] propose a theory of SPL refinement independent of the language used to implement assets, where a refinement expresses a refactoring or generalization of the SPL. In order to define algorithms for determining the relationship (*i.e.*, generalization, specialization, refactoring or arbitrary) between two feature models, Thüm *et al.* model operations on FMs using propositional formula [19]. FEATUREIDE [20] provides support for determining such relations. SPLEMMMA uses such algorithms. Furthermore, SPLEMMMA allows *maintainers* to define operations or evolutions accepted by SPLs to provide some control on the impact of SPLs variability.

SPL heterogeneity. To the best of our knowledge, there are not works focusing on evolution of heterogeneous SPLs. However, we found some tool support for analysis of heterogeneous FMs such as FAMA [21] and BETTY [16]. FAMA is an extensible framework for an automated analysis of FMs.

The framework deals with SPL heterogeneity by supporting different variability models, reasoners and analysis questions. BETTY, which is built on the top of FAMA, is a framework for benchmarking and testing on the analysis of feature models. The framework can be easily extended to support different formats of variability models. Although FAMA and BETTY do not focus on the evolution and only care about FMs, in SPLEMMMA we exploit their idea of using automated analysis on FM based on existing research.

7. DISCUSSION

The use of SPLEMMMA on evolution of YOURCAST and SALOON allows us to confirm the following advantages:

- SPLEMMMA supports the validation of evolution by fostering reuse of well validated algorithms as made by the FAMA framework [21] and FEATUREIDE [20]. For example, in YOURCAST and SALOON we use the algorithm proposed by Thüm *et al.* [19] to determine the impact of evolution on the SPL.
- Evolution is less error-prone. By reifying evolution in a model, we force the definition of restrictions about elements that can be modified and how. This provides a degree of control on evolution to *maintainers*. Thus, in YOURCAST, the SPL *maintainer* decides that only sources and renderers can be added. In SALOON, the ontology remains untouched by evolution.
- Evolution tools generated by SPLEMMMA are easily modifiable. The incorporation of new operations or evolvable elements only requires the modification of the evolution model and the regeneration of tools.
- The framework is SPL independent. The successful tool generation and execution of the evolution in both cases show that we can deal with heterogeneous SPL. The framework provides a suitable abstract level to avoid conflicts with the kind of assets, mappings of feature models employed by SPL, enabling us to deal with SPL heterogeneity. Existing tools such as *FeatureEdit* [19] and *FeatureMapper* [17] assume the all SPL uses their formalism to express variability.
- The consistency of the whole SPL is considered. As *FeatureMapper* and the theory proposed by Borba *et al.* [5], SPLEMMMA deals with the co-evolution of mappings, FMs and assets according to the kind of evolution. In YOURCAST and SALOON, the definition of *Interspatial Contributions* makes tools generated by SPLEMMMA check the presence of the required elements as well as their correct structure and basic semantics according to preconditions and postconditions. Other approaches (cf. [19]) only focus on the impact of FMs.

The main drawback of the approach is related to the formalism to deal with models. In SPLEMMMA implementation, we assume that the feature-oriented SPL uses EMF for the model definition. This assumption impacts the genericity of the framework by limiting the kind of SPL that can be evolved through SPLEMMMA. However, the EMF dependency is at the implementation level and not conceptual. Therefore, SPLEMMMA tools can be improved to support different model formalisms as made by FAMA [21].

8. CONCLUSION

In this paper we introduced SPLEMMMA, a framework for the validation of feature-oriented SPL evolution. The validation

is reached through the generation of tools from evolution models describing the granularity level of changes as well as evolvable elements. In order to *keep consistency* of the SPLs, SPLEMMMA enables the definition of preconditions and post-conditions on the evolution by reusing existing algorithms for FMM analysis as well as basic validations regarding the kind of evolution. Each evolution is reified as one operation that defines the kind of elements to modify in order to *control the structural impact* on SPL. Furthermore, the framework is independent from business domains and technologies used for the development of SPLs, to *deal with SPL heterogeneity*. The framework is validated with two use cases from different domains. Experiences of the evolution with them confirm the soundness of the approach.

In terms of perspectives, we intend to consolidate the results obtained so far, following three directions. First, we will explore the introduction of the *when* dimension from software evolution by means of version management. This would make the approach more flexible by enabling SPL *maintainers* to retrieve previous versions of the SPL which can be useful when the evolution produces specializations of SPLs. Secondly, we plan to introduce an ontology for improving the definition of relationships between operations and validation services. This ontology will enable a partial automatization of service selection according to the evolution operations. Finally, we plan to extend the SPLEMMMA metamodel and tools to enable privilege management on the evolution. Until now, if different operations are required regarding the kind of role (*i.e.*, *maintainer* or *contributor*) different evolution models and the related tools need to be generated. The idea is to have an evolution tool by SPL with authentications that grant access only to operations according to user role.

Acknowledgments. This work is supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council, the FEDER through the *Contrat de Plan Etat Region Campus Intelligence Ambiante* (CPER CIA) 2007-2013 and the ANR YOURCAST project under contract ANR-2011-EMMA-013-01.

9. REFERENCES

- [1] S. Apel, C. Kästner, A. Größlinger, and C. Lengauer. Type safety for feature-oriented product lines. *Automated Software Engg.*, 17(3):251–300, Sept. 2010.
- [2] D. Batory. Feature models, grammars, and propositional formulas. In *Proceedings of the 9th international conference on Software Product Lines*, SPLC’05, pages 7–20, Berlin, Heidelberg, 2005. Springer-Verlag.
- [3] D. Benavides, S. Segura, and A. Ruiz-Cortés. Automated analysis of feature models 20 years later: A literature review. *Inf. Syst.*, 35(6):615–636, Sept. 2010.
- [4] D. Benavides, P. Trinidad, and A. Ruiz-Cortés. Automated reasoning on feature models. In *Proceedings of the 17th international conference on Advanced Information Systems Engineering*, CAiSE’05, pages 491–503, Berlin, Heidelberg, 2005. Springer-Verlag.
- [5] P. Borba, L. Teixeira, and R. Gheyi. A theory of software product line refinement. In *Proceedings of the 7th International colloquium conference on Theoretical aspects of computing*, ICTAC’10, pages 15–43, Berlin, Heidelberg, 2010. Springer-Verlag.
- [6] J. Buckley, T. Mens, M. Zenger, A. Rashid, and G. Kniesel. Towards a taxonomy of software change: Research articles. *J. Softw. Maint. Evol.*, 17(5):309–332, Sept. 2005.
- [7] F. Dupin and M. Adolph. Digital signage: the right information in all the right places. Technical report, International Telecommunication Union (ITU), Nov. 2011.
- [8] N. Gamez and L. Fuentes. Software product line evolution with cardinality-based feature models. In *Proceedings of the 12th international conference on Top productivity through software reuse*, ICSR’11, pages 102–118, Berlin, Heidelberg, 2011. Springer-Verlag.
- [9] J. Guo, Y. Wang, P. Trinidad, and D. Benavides. Consistency maintenance for evolving feature models. *Expert Systems with Applications*, 2011.
- [10] K. Kang, S. Cohen, J. Hess, W. Nowak, and S. Peterson. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. Software Engineering Institute, Carnegie Mellon University Pittsburgh, PA., 1990.
- [11] R. Lotufo, S. She, T. Berger, K. Czarnecki, and A. Wasowski. Evolution of the linux kernel variability model. In *Proceedings of the 14th international conference on Software product lines: going beyond*, SPLC’10, pages 136–150, Berlin, Heidelberg, 2010. Springer-Verlag.
- [12] R. Mitschke and M. Eichberg. Supporting the evolution of software product lines. In *Oldevik, J., Olsen, G.K., Neple, T., Paige, R. (eds.) ECMDA Traceability Workshop (ECMDA-TW) 2008 Proceedings*, pages 87–96, 2008.
- [13] K. Pohl, G. Böckle, and F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.
- [14] C. Quinton, N. Haderer, R. Rouvoy, and L. Duchien. Towards Multi-Cloud Configurations Using Feature Models and Ontologies. In *Proceedings of the 1st International Workshop on Multi-Cloud Applications and Federated Cloud*, Multi-Cloud’13, 2013. To appear.
- [15] C. Salinesi, C. Rolland, and R. Mazo. VMWare: Tool Support for Automatic Verification of Structural and Semantic Correctness in Product Line Models. In *International Workshop on Variability Modelling of Software-intensive Systems*, page 173, Sevilla, Espagne, Jan. 2009.
- [16] S. Segura, J. A. Galindo, D. Benavides, J. A. Parejo, and A. Ruiz-Cortés. Betty: benchmarking and testing on the automated analysis of feature models. In *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, VaMoS’12, pages 63–71, New York, NY, USA, 2012. ACM.
- [17] C. Seidl, F. Heidenreich, and U. Assmann. Co-evolution of models and feature mapping in software product lines. In *Proceedings of the 16th International Software Product Line Conference - Volume 1*, SPLC’12, pages 76–85, New York, NY, USA, 2012. ACM.
- [18] C. Thao. Managing evolution of software product line. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 1619–1621, Piscataway, NJ, USA, 2012. IEEE Press.
- [19] T. Thüm, D. Batory, and C. Kästner. Reasoning about edits to feature models. In *Proceedings of the 31st International Conference on Software Engineering*, ICSE ’09, pages 254–264, Washington, DC, USA, 2009. IEEE Computer Society.
- [20] T. Thüm, C. Kästner, F. Benduhn, J. Meinicke, G. Saake, and T. Leich. Featureide: An extensible framework for feature-oriented software development. *Science of Computer Programming*, 2012. to appear; accepted 7 Jun 2012.
- [21] P. Trinidad, D. Benavides, A. Ruiz-Cortés, S. Segura, and A. Jimenez. Fama framework. In *Proceedings of the 2008 12th International Software Product Line Conference*, SPLC’08, pages 359–, Washington, DC, USA, 2008. IEEE Computer Society.
- [22] S. Urli, M. Blay-Fornarino, P. Collet, and S. Mosser. Using Composite Feature Models to Support Agile Software Product Line Evolution. In *Models and Evolution 2012 (ME’12), workshop*. ACM DL, Sept. 2012.