A Bottom-Up Approach for Reconstructing Software Architecture Product Lines

Mohamed Lamine Kerdoudi, Tewfik Ziadi, Chouki Tibermacine, Salah Sadou

A Bottom-Up Approach for Reconstructing Software Architecture Product Lines

Mohamed Lamine Kerdoudi
L.Kerdoudi@univ-biskra.dz
Computer Science Department, University of Biskra
Algeria

Tewfik Ziadi
Tewfik.Ziadi@lip6.fr
LIP6, Sorbonne Universités, UPMC University Paris 6
France

Chouki Tibermacine
Chouki.Tibermacine@lirmm.fr
LIRMM, CNRS and Montpellier University
France

Salah Sadou
Salah.Sadou@irisa.fr
IRISA- University of South Brittany
France

ABSTRACT

A large component and service-based software system exists in different forms, as different variants targeting different business needs and users. This kind of systems is provided as a set of “independent" products and not as a “single whole". The presence of a single model describing the architecture of the whole system may be of great interest for developers of future variants. Indeed, this enables them to see the invariant part of the whole, on top of which new functionality can be built, in addition to the different options they can use. We investigate in this work the use of software product line reverse engineering approaches, and in particular the framework named BUT4Reuse, for reconstructing an architecture model of a Software Architecture Product Line (SAPL), from a set of variants. We propose a generic process for reconstructing an architecture model of such a product line. We have instantiated this process for the OSGi Java framework and experimented it for building the architecture model of Eclipse IDE SPL.

ACM Reference format:

1 INTRODUCTION

Software Product Line (SPL) Engineering (SPLE) considers the existence of a single architecture describing all the variants that implement each software product. The particularity of this “single” architecture is that it includes what is refereed as *a variability model*, in which *variability* and *commonality* are explicitly specified using high level characteristics of the so-called *features* [2]. These are then mapped to components, which are organized according to the identified features. Product variants can be *derived* (generated) by choosing the desired features, then SPL tools choose and assemble the appropriate components mapped to the selected features [2]. During recent years, multiple approaches have been proposed addressing SPL implementation, or product derivation [2, 22]. However, there are many systems that exist as several “independent" variants and not as a “single whole". Indeed, large component and service-based software systems exist in different forms, as different variants targeting different business needs and users. These systems often use ad hoc mechanisms to manage variability and they do not take complete benefits from the SPLE framework.

This paper considers the challenge of analysing these systems to reverse-engineer a software architecture following the SPLE framework that is common to all the existing variants. We call this constructed architecture *Software Architecture Product Line* (SAPL) which is different from the Software Product Line Architecture (SPLA)[16]. We defend a new vision by considering SAPL as a reference architecture starting from which the architecture of each product variant can be derived. Indeed, each derived software variant can have its own life that is regulated by evolution needs whose origin often depends on the context which is specific to each product. But, SPLA raises problems during the maintenance stage of a product on two points: i) referring to a generic architecture to understand a given product is a very difficult task. Knowing that understanding is the most costly activity during maintenance, this will generate considerable additional costs. ii) Modifying a generic architecture, to take into account the modifications made on one of its products, is a task that is not only difficult and error prone, but also with unforeseeable consequences on the other products. We propose in this paper a process (see Section 2 and Section 3) for SAPL-reverse-engineering. This process extends the BUT4Reuse framework, which is one of the most effective methods for SPL-reverse-engineering [13, 15]. This framework was proposed as a generic and extensible framework for SPL-RE. We extend BUT4Reuse to SAPL reverse-engineer large component and service-based software systems starting from a collection of their existing variants. We show the results of our experiments in Section 4.

2 A GENERIC PROCESS FOR SAPL-REVERSE ENGINEERING

Before presenting our SAPL-RE process, we first describe the meta-model for software srchitectures that are supported by our approach. Figure 1 depicts the defined SAPL meta-model. We have

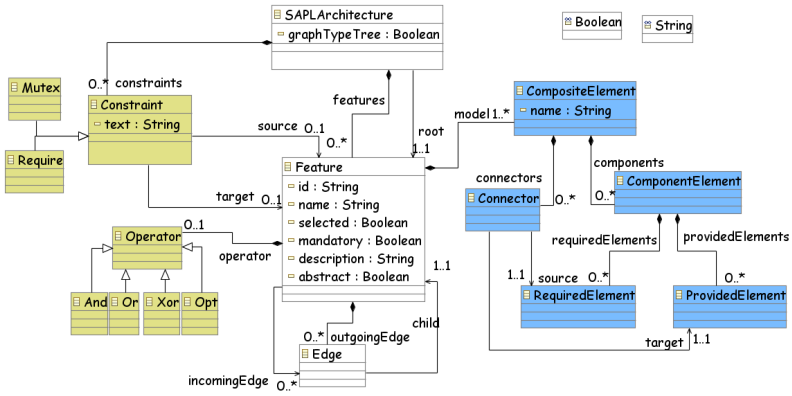

Figure 1: SAPL Metamodel for Component-Based Software Variants

been inspired in its definition by the feature meta-model in [18]. We enriched it by adding component-based architecture elements. An instance of this meta-model serves as a feature model that represents the variability in a family of software product variants and a comprehensive architecture (modules / components) that helps the developer to understand the structure of the SPL features. As our meta-model is generic and used for representing component-based systems, it has been defined based on an abstract syntax of a software component model. The overall process of our approach is illustrated in Figure 2. This process is defined in three main activities:

2.1 Reverse-Engineering of SA Variants

First, we use reverse-engineering techniques to extract a software architecture variant from the source code of each software variant.

2.2 SAPL Construction

In this activity, the different SA variants are analyzed and compared to identify the common part and the different features. As illustrated in Figure 2, this activity extends the BUT4Reuse framework to support architectural artefacts. Indeed, BUT4Reuse [13, 15] was proposed as a generic and extensible framework to identify features from a set of similar artifacts. To support the different types of artifacts, and enabling extensibility, BUT4Reuse relies on *adapters* for the different artifact types. These adapters are implemented as the main components of the framework. An adapter is responsible for decomposing each artifact type into the constituting elements, and for defining how a set of elements should be constructed to create a reusable asset. In this paper, we extend BUT4Reuse by proposing a new adapter related to software architectures. In addition, to allow comparing software architectures, this new adapter is designed with a set of parameters to consider different architectural views (services, interfaces, packages, extensions, etc). Once the adapter is implemented, SAPL construction follows four sub-activities as illustrated in Figure 2.

Decomposition in Architectural Elements. The first step takes as input a collection of architecture variants that are obtained from the reverse-engineering activity. It decomposes each variant as a set of Architectural Elements (AEs). The computed AEs can be of different types depending on the considered view. This can include components, interface, services, extension, etc.

Block Identification and Feature Naming. This step reuses algorithms implemented in BUT4Reuse which automatically identify sets of AEs that correspond to the distinguishable features from the SA variants. These sets of AEs are named *Blocks*. In this paper, we reused especially the algorithm, called *Interdependent Elements* that formalize block identification using class equivalences. Once blocks are identified, the next step is a semi-automatic process where domain experts manually review the elements from the identified blocks to map them with the functionalities (i.e., features) of the system. BUT4Reuse integrates what is called *VariCloud* [14], an approach that analyzes the elements inside each block and extracts words that help domain experts to identify features. *VariCloud* uses information retrieval techniques, such as TF-IDF, to analyze the text describing elements inside blocks. The descriptions used by BUT4Reuse to build word clouds are thus provided by the specific adapter. As we will see in the next section, for our adapter, words correspond to the names of packages, interfaces and plugins.

Dependencies Identification. During this step, the approach identifies the dependencies between the different blocks. BUT4Reuse uses the dependencies defined within the adapter to identify dependencies between blocks.

Multi-View SAPL Constrcution. A software architecture of a large system is a complex entity; it cannot be presented in a single view. One of the most important concepts associated with software architectures are views. A view is the result of applying a viewpoint to a particular system of interest (for instance, service-, interface-, and extension-oriented views). In this step of our process, we enable the developer to construct a muti-view SAPL. These views can help and assist the developer to understand progressively the SPL.

2.3 Variants Derivation

In this step, the developer can select starting from the recovered SAPL a set of features that meet her/his requirements for deriving the architecture of the new variant. We provide a graphical tool to visualize the derived architecture. Once the developer analyzed and understood this architecture, she/he can derive the new product as a new variant.

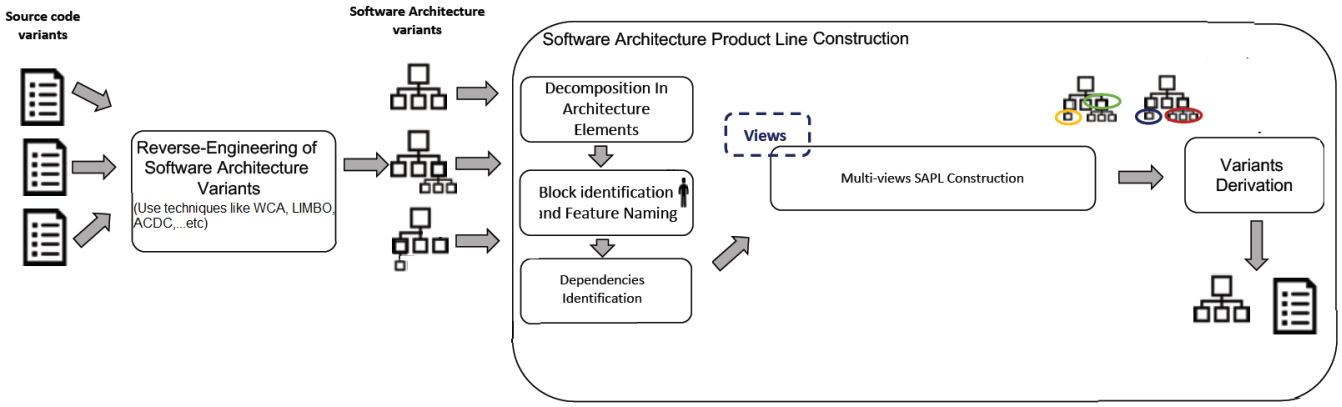

Figure 2: Proposed SAPL-Reverse Engineering Process

3 INSTANTIATION OF THE PROCESS FOR OSGI COMPONENT/SERVICE MODEL

We have instantiated our process for OSGi Java applications. The OSGi specification defines a component model and a framework for creating highly modular Java systems [17]. Eclipse-based applications run on top of Equinox which is the reference implementation of the OSGi specification. It is a collection of similar software products that share a set of software assets. It offers a set of "releases" where each one is a large-sized Java application composed of hundreds to thousands of components, registering and consuming hundreds of services. The default Eclipse releases are predefined for targeting specific developer needs. Currently, if a developer wants to create a customized release, she/he has to select one of the default releases (for instance, IDE for C/C++ Developers) and then manually install new plugins which meet her/his requirements. In this paper, we consider Eclipse releases as product variants and we aim to adopt the SAPL approach in order to be able to develop efficiently a personalized Eclipse variant. We have adapted the meta-model in Figure 1 for the OSGi component model. In this meta-model an OSGi component is represented by a `PluginElement` (specialization of `ComponentElement`). The required elements in the OSGi meta-model are: `Extension`, `ImportPackage`, `RequiredInterface`, and `ConsumedService`. The provided elements are `ExtenionPoint`, `ExportedPackage`, `ProvidedInterface`, and `RegistredService`.

In order to implement our adapter for Eclipse-SA variants, we have followed the generic activities which are defined in [15][1]. Besides, our OSGi meta-model allows to produce several SA views that represent different kinds of plug-in's capabilities and requirements. The supported architecture views are: *interface, service, package, and extension views*. Of course these views are not orthogonal, there are intersections between each other. But, nobody would be able to understand the whole system by analyzing all the views together. Thanks to this meta-model, developers can progressively understand the system by analyzing each architecture view separately.

At the end, in order to create a new Eclipse variant, we implemented a derivation mechanism inegrated with the `FeatureIDE` tool. The developer can configure manually the SAPL and select a set of desired features among the identified list. Before deriving the variant, we offer to the developer a way for mapping this configuration into an architecture model for this variant. This architecture model represents the selected features without variability information, which is useful as a product documentation. Finally, the new variant can be derived by collecting the extracted software assets which correspond to the selected features.

4 EVALUATION OF THE PROCESS

In order to evaluate our approach, we have conducted a set of experiments on 12 Eclipse releases [2]. First, we have measured the accuracy of SAPL recovery process. We used the *Architecture2Architecture* (`a2a`) [10] metric to measure the architectural change between the derived SA using our approach and the SA of the variant that is created manually (installing manually a new feature by clicking on Help->Install New Software...). Second, we have used a set of measures for comparing the size and complexity of the recovered SAPL views using our process.

We have used the `LoongFMR` implementation[3] of the `a2a` metric for comparing the two architectures. The obtained value is `a2a` = **87%**. This value can be considered as a good result that indicates that the two architectures are almost the same. In fact, this little difference between them is due to the fact that the manual installation allows to add plugins with old versions which already exist.

Besides, we compared the size of the recovered SAPL views with the whole SAPL (all views together). First, we have observed that the number of elements in each SAPL view is much less than the number of elements in the whole SAPL. This confirms our intuition that focusing on a single view allows to reduce the size and complexity of the SAPL. We have also observed that the number of elements in the extension view is less than the number of elements in the other views. This supports our idea that the developer needs to start with an architecture view that contains a few elements (only the plugins and their extensions). After that, (s)he can pass to another view with more information about other kinds of dependencies.

[1]For more details about our adapter see: https://pages.lip6.fr/tewfik.ziadi/ecsa19.html

[2]Downloaded from: https://bit.ly/2uylkT8

[3]Downloded from:https://github.com/csytang/LoongFMR

5 RELATED WORK

Wesley et al. [3] present a complete survey on the existing SPLE approaches. Several extensions of the framework BUT4Reuse have already been developed and published in [9, 12, 24]. Besides, software architecture recovery (SAR) is a challenging problem, and several works in the literature have already proposed contributions to solve it (e.g., works cited in [4, 10, 11]). Most of these approaches are proposed for a single software architecture recovery. In the last decade several works had proposed approaches that aim to recover component-/service-oriented architectures from existing systems. For example, the work in [19] is based on the definition of a correspondence model between code elements and architectural concepts. In [1] a component is considered as a group of classes collaborating to provide a system function. The authors in [7] recover BPMN models starting from service oriented systems that have been generated from web applications. Some works such as [6, 8] have been proposed to recover software architecture at run-time. In our approach, we assume that the SAs of the product variants can already exist and considered as inputs for our SAPL-RE process. Otherwise, we can use one of the existing approaches for recovering them.

Besides, few works were proposed in the literature that aim to recover SPLA. [21] presented a mapping study of the existing SPLA recovery approaches. Shatnawi et al. [20] have proposed a process for recovering software product line architectures of a family of object-oriented product variants. They build the SPLA as a feature model where the dependencies between component variants are based on relations of type *alternative*, OR, AND, *require* and *exclude*. The authors in [5] have proposed an approach for recovering SPLA from software product variants. Compared to our work, the recovered SAPL using our approach is both a feature model and a complete architecture that shows all the architectural connections between components. In addition, our inputs can be system variants or SA variants. The variability is identified starting from the elements in the input architectures. Wille et al. [23] have proposed a variability mining approach for Technical Architecture variants. Our solution can derive new SAs and product variants starting form the reconstructed SAPL.

6 CONCLUSION

Recovering architecture models of large-sized software products is an important activity in software maintenance and evolution. SPL Reverse Engineering (SPL-RE) processes enable to recover models with a better structure, since they factorize the variable part in the product variants and enable to see the variability points. In this work we focused on component- and service-based systems and proposed in this paper: i) a (meta-)model for architectures of component/service-based software product lines, ii) the design of an adapter of a generic SPL-RE process (But4Reuse) for building architecture models (SAPL models) by analyzing product variants, iii) an implementation of this adapter specific to OSGi-based applications, and iv) an experimentation of this recovery process on a set of Eclipse releases.

As perspectives to this work, we plan to study the enrichment of SPL reverse engineering of large component/service-based systems by including a learning module which exploits existing SPLs and their variants/features. In addition, we envisage the instantiation of the process for other component/service frameworks, or just investigate its use with Java modules for exploring variability in Java SE, EE, ME, TV, etc. From a tool-support point of view, we intend to enrich our implementation by capabilities such as software product configuration and derivation to complete the “loop".

REFERENCES

[1] Simon Allier, Salah Sadou, Houari A. Sahraoui, and Régis Fleurquin. 2011. From Object-Oriented Applications to Component-Oriented Applications via Component- Oriented Architecture. In *Proc. of the 9th WICSA, USA*. IEEE.

[2] Sven Apel, Don S. Batory, Christian Kästner, and Gunter Saake. 2013. *Feature-Oriented Software Product Lines - Concepts and Implementation*. Springer.

[3] Wesley Klewerton Guez Assunção and Silvia Regina Vergilio. 2014. Feature location for software product line migration: a mapping study. In *18th SPLC, Companion Volume, Italy*.

[4] Stéphane Ducasse and Damien Pollet. 2009. Software Architecture Reconstruction: A Process-Oriented Taxonomy. *IEEE TSE* 35, 4 (2009), 573–591.

[5] Hamzeh Eyal-Salman and Abdelhak-Djamel Seriai. 2016. Toward Recovering Component-based Software Product Line Architecture from Object-Oriented Product Variants. In *Proc. of SEKE*.

[6] Gang Huang, Hong Mei, and Fu-Qing Yang. 2006. Runtime recovery and manipulation of software architecture of component-based systems. *Journal of ASE* 13, 2 (2006), 257–281.

[7] Mohamed Lamine Kerdoudi, Chouki Tibermacine, and Salah Sadou. 2016. Opening web applications for third-party development: a service-oriented solution. *Journal of SOCA* 10, 4 (2016), 437–463.

[8] Mohamed Lamine Kerdoudi, Chouki Tibermacine, and Salah Sadou. 2018. Spotlighting Use Case Specific Architectures. In *Proc. the 12th ECSA*. Springer.

[9] Li Li, Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Mining families of Android applications for extractive SPL adoption. In *Proceedings of the 20th SPLC 2016, Beijing, China*.

[10] Thibaud Lutellier, Devin Chollak, Joshua Garcia, Lin Tan, Derek Rayside, Nenad Medvidović, and Robert Kroeger. 2018. Measuring the impact of code dependencies on software architecture recovery techniques. *IEEE TSE* 44, 2 (2018), 159–181.

[11] Onaiza Maqbool and Haroon Babri. 2007. Hierarchical clustering for software architecture recovery. *IEEE TSE* 33, 11 (2007), 759–780.

[12] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Automating the Extraction of Model-Based Software Product Lines from Model Variants (T). In *30th IEEE/ACM, ASE , Lincoln, NE, USA,*. 396–406.

[13] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2015. Bottom-up adoption of software product lines: a generic and extensible approach. In *Proceedings of the 19th SPLC, Nashville, TN, USA*. 101–110.

[14] Jabier Martinez, Tewfik Ziadi, Tegawendé F. Bissyandé, Jacques Klein, and Yves Le Traon. 2016. Name suggestions during feature identification: The VariClouds approach. In *Proceedings of the 20th SPLC , Beijing, China*.

[15] Jabier Martinez, Tewfik Ziadi, Tegawendé F Bissyandé, Jacques Klein, and Yves Le Traon. 2017. Bottom-up technologies for reuse: automated extractive adoption of software product lines. In *Proc. of the 39th ICSE Companion*. IEEE Press, 67–70.

[16] Mari Matinlassi. 2004. Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA. In *Proceedings of the 26th ICSE (ICSE '04)*. IEEE Computer Society, Washington, DC, USA, 127–136.

[17] Jeff McAffer, Paul VanderLei, and Simon Archer. 2010. *OSGi and Equinox: Creating highly modular Java systems*. Addison-Wesley Professional.

[18] Gilles Perrouin, Jacques Klein, Nicolas Guelfi, and Jean-Marc Jézéquel. 2008. Reconciling automation and flexibility in product derivation. In *Proc of. the 12th SPLC*. IEEE.

[19] Abderrahmane Seriai, Salah Sadou, and Houari A Sahraoui. 2014. Enactment of Components Extracted from an Object- Oriented Application. In *Proc. ECSA*.

[20] Anas Shatnawi, Abdelhak-Djamel Seriai, and Houari Sahraoui. 2017. Recovering Software Product Line Architecture of a Family of Object-oriented Product Variants. *J. Syst. Softw.* 131, C (Sept. 2017), 325–346.

[21] Zipani Tom Sinkala, Martin Blom, and Sebastian Herold. 2018. A mapping study of software architecture recovery for software product lines. In *Companion Proceedings of ECSA*.

[22] Thomas Thüm, Christian Kästner, Fabian Benduhn, Jens Meinicke, Gunter Saake, and Thomas Leich. 2014. FeatureIDE: An extensible framework for feature-oriented software development. *Science of Computer Programming* 79, 0 (2014).

[23] David Wille, Kenny Wehling, Christoph Seidl, Martin Pluchator, and Ina Schaefer. 2017. Variability Mining of Technical Architectures. In *Proceedings of the 21st SPLC - Volume A*. ACM.

[24] Tewfik Ziadi and Lom Messan Hillah. 2018. Software Product Line Extraction from Bytecode based applications. In *Proc. of the 23rd (ICECCS)*. IEEE, 221–225.