Proceedings of the Federated Conference on Computer Science and Information Systems pp. 1715–1718
DOI: 10.15439/2016F170
ACSIS, Vol. 8. ISSN 2300-5963

A Development Process Based on Variability Modeling for Building Adaptive Software Architectures

Ngoc-Tho Huynh
IRISA / Université Européenne de Bretagne / TELECOM Bretagne, Brest, France
tho.huynh@telecom-bretagne.eu

Maria-Teresa Segarra
IRISA / Université Européenne de Bretagne / TELECOM Bretagne, Brest, France
mt.segarra@telecom-bretagne.eu

Antoine Beugnard
IRISA / Université Européenne de Bretagne / TELECOM Bretagne, Brest, France
antoine.beugnard@telecom-bretagne.eu

***Abstract*—Adaptive software is a class of software which is able to dynamically modify at runtime its own internal structure and hence its behavior in response to changes in its operating environment. Adaptive software development has been an emerging research area of software engineering in the last decade. Many existing approaches use techniques issued from software product lines (SPLs) to develop adaptive software. They propose tools, frameworks or languages to build adaptive software architectures (ASAs) but do not guide developers on the process of using them. In this paper, we propose an adaptive software architecture development process to guide developers building an ASA. One of the important activities of this development process is software specification based on models. In our process, we propose to use the models and basic tools of Common Variability Language (CVL, proposed as an OMG standard) to generate an ASA and a subprocess to specify these models.**

I. Introduction

MAINTENANCE phase is one important stage of a software development process. Maintenance aims at evolving and updating software to meet new requirements or to satisfy new conditions in the software execution context. In order to be maintained and evolved, software is usually stopped, then updated, and finally restarted. However, in certain circumstances, stopping the software is unacceptable, e.g., cloud gaming, medical, and finance systems as stopping the software has consequences for business, or even dangerous for humans.

One solution to solve this problem is to add dynamic reconfiguration mechanisms to modify the software architecture at runtime. Several works are interested in determining when the architecture reconfiguration can occur [1], [2]. Other works are interested in tools and methods to develop an ASA [3], [4], [5]. Particularly, they allow specifying variation points in the software architecture. A variation point is a particular point in the architecture specification where choices can be realized. In SPL, variation points are specified in a variability model. Such a model describes commonality and variability of the product line. Commonality represents common parts of all products in the family. Variability represents the parts that may be different in different software products [6]. A product contains all the common parts and the choices made on all the variation points. Once all the variation points are resolved (a choice has been made for all of them), the variability model is said to be configured.

Many existing works use techniques issued from SPLs to develop adaptive software [7], [8]. These works use a variability model to specify the software variability and propose the mapping between the variability model and the software architecture. When the variability model is configured, the corresponding component architecture is deduced. During software execution, a new configuration of the variability model can be decided and the corresponding software architecture deduced. Then, by calculating the differences between the current and the new architecture, reconfiguration actions are identified. However, these approaches do not specify a software development process to guide developers to build the adaptive software architecture.

To deal with the above limitation, we are working on a development process to guide developers to specify information needed to generate an ASA. In this paper, we focus on identifying the information to be specified in a development process. All along the development process we use CVL metamodels [9] to manage variability and propose a subprocess to describe how to specify this variability.

The remainder of the paper is structured as follows. Section II describes the CVL approach. Our general development process for building an ASA is presented in Section III. Section IV focuses on the variability modeling stage of our process and how a variability model is configured in order to generate an adaptive software. Related work is discussed in Section V then the paper concludes.

II. Common Variability Language

CVL [9] is a domain-independent language, and also an approach for specifying and configuring variability. We use CVL in our approach for two main reasons:

- CVL has been proposed as an OMG standard and we think it will be largely used in the near future.

978-83-60810-90-3/$25.00

- It proposes a MOF-based variability language which means that any MOF-based product model can be easily extended with variability information using CVL.

An overview of the CVL approach is depicted in Figure 1. The base model is used to specify the elements of the architecture that does not contain any information about variability. Variability information is specified in the variability model. In order to generate the configuration of a specific product, the resolution model consists of VSpecResolutions each determining a decision for a VSpec.

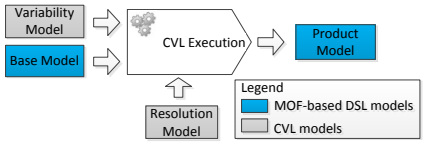

Fig. 1. CVL approach

In CVL, a variability model consists of three main parts:

- **VSpec tree.** A VSpec tree consists of VSpecs which are similar to features in feature models (FMs) [10]. There are four types of VSpecs: *Choice*, *Variable*, *VClassifier* and *CompositeVSpec*. A *Choice* allows to specify binary selections (true/false). A *Variable* should be used to specify a parameter which value may change. A *VClassifier* allows to specify a min and max number of instances of the VSpec. Finally, *CompositeVSpec* is used for modularity purposes.
- **Variation points.** A variation point links a VSpec to the corresponding elements in the base model.
- **OCL constraints.** CVL supports the definition of OCL constraints among VSpecs of a VSpec tree that cannot be directly captured by hierarchical relations.

Let *CVL model* denote the three models: the base model, the variability model, and the resolution model.

CVL offers tools and meta-models to specify variability of a product family but it does not offer a method to specify the variability model and the base model. Additionally, the product is generated without runtime variability.

III. A DEVELOPMENT PROCESS FOR BUILDING AN ADAPTIVE SOFTWARE ARCHITECTURE

In order to help engineers on developing adaptive software, we propose a development process that includes variability as a first-class stage. Although our process is based on CVL models and meta-models, other tools and frameworks may be used. The process we propose encompasses the variability specification issue to generate the architecture of the product. On the other hand, a reconfiguration process (at runtime) is out of the scope of this paper, but appears in the process in order to give an overview of the whole picture.

Our process is based on SPL engineering. The SPL engineering distinguishes two phases: domain engineering and application engineering [11] (see Figure 2).

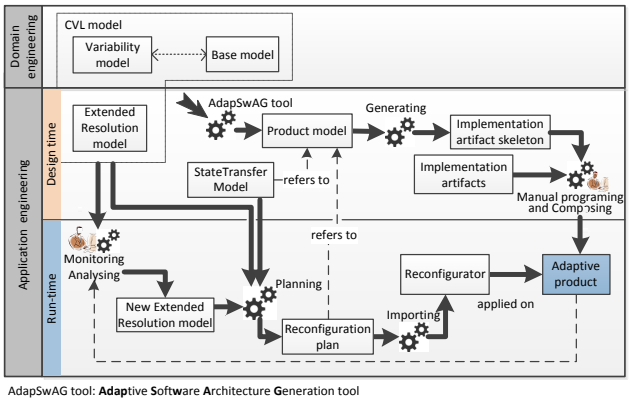

Fig. 2. General Adaptive Software Architecture Development Process

Domain engineering is the process responsible for defining the commonality and variability of the product line. The variability model and the base model are artifacts of this phase. The base model in our approach corresponds to a component-based architecture model specified by using an architecture description language (ADL).

Application engineering refers to the process of actually combining the artifacts obtained during the domain engineering phase, in order to generate a software product. In our approach, this phase is partitioned into two subprocesses: one at design time and another one at runtime.

At design time, a resolution model is specified to configure the variability model. This resolution model is extended from the CVL resolution meta-model for generating the necessary elements in an ASA. Generating the architecture is realized by our **Adap**tive **S**oft**w**are **A**rchitecture **G**eneration (AdapSwAG) tool. As depicted in Figure 2, the input of this tool is the CVL model and its output is a particular architecture (the product model). Compared to the CVL execution of the CVL approach, the software architecture generated by AdapSwAG tool includes part of its initial variability, and is specified in the same language as the base model. Based on the product model and a given target platform, a text generation module generates implementation artifacts skeleton. Each of them corresponds to a component specified in the product model and consists of component implementations, a component specification, and a configuration file. Once the implementation artifacts skeleton is generated, implementation artifacts that are either available or should be developed are integrated into it to build operational components. This task is performed by an application engineer. The result is an adaptive product that can be executed in the target platform.

If the product embeds variability, then, its architecture may change at runtime, and the components state affected by the changes should be migrated to the new ones. As the semantic of this state is component-specific, the state migration task is never completely automated. Thus, engineers have to specify a state transfer model that gives actions to effectively migrate state between components. The state transfer model represents the state mapping between previous components and the new

ones in the product model. It is specified at design time and used at runtime.

At runtime, because of new user requirements or execution environment changes, a change decision may be made. In this case, a new resolution model (new configuration) must be specified. This configuration may be either defined by engineers or computed thanks to analysis activities by a decision module integrated into the adaptive product. The current resolution model, the new one, and the state transfer model are processed to generate a reconfiguration plan that refers to the product model. Finally, the reconfiguration plan is injected into the reconfigurator to execute the reconfiguration actions. The moment when these actions are executed at runtime is important since the components must be driven to safe state (e.g., quiescent [1] or tranquility [2]), but this is out of the scope of this paper.

In this paper, we focus on the domain engineering phase and the design subprocess of the application engineering and give details on how to specify the VSpec tree, the base model, and the variation points to generate an ASA.

IV. Variability Modeling and Configuration

Variability modeling plays an important role in our approach. This section presents strategies to specify variability models according to CVL meta-model.

A. Variability Modeling Process

There may be different strategies to specify the variability and the base models.

- "Variability-driven process" (top-down approach): in this strategy the VSpec tree is specified first from information collected by engineers such as documentation or already existing products. Following this model, a base model may be built. Finally, variation points may be specified. This strategy allows specifying variability at high level of abstraction towards the diversity of components at concrete level.
- "Architecture-driven process" (bottom-up approach): in this strategy the base model is specified first. Then, VSpec tree is deduced and finally, variation points identified. This strategy allows specifying components at concrete level towards high level abstraction.
- "VSpec tree - base model independent process" (hybrid approach): in this strategy the VSpec tree and the base model are independently specified. Once the VSpec tree and the base model are specified, variation points can be identified to do the mapping between them. The advantage of this strategy is to allow independent specifications. Unlike the two first strategies, there is no guarantee to have a variation point for each VSpec.

As our approach is focused on guiding developers to identify information for generating adaptive products, we consider the first strategy as the process to be followed by engineers. In this strategy, variability modeling plays an important role. We focus on this task to represent the changes that an ASA may undergo. Based on the variability model, the base model can be created appropriately. The adopted strategy is depicted in Figure 3. In order to specify the VSpec tree and the base model, engineers need to identify variability and commonality. To do this, they can use documentation of existing applications of a product line, domain experts may also participate on the specification (step 1 in Figure 3). Once commonality and variability are identified, they are represented as a VSpec tree (step 2 in Figure 3). In step 3, a base model is defined with respect to the VSpec tree and the set of commonality and variability in the first step. Finally, step 4, variation points are defined to map VSpecs in the VSpec tree and elements in the base model. Steps 1 and 2 have to be manually realized by variability engineers. Step 3 should be realized by a software architecture expert. Step 4 can be manually (by an expert) or semi-automatically done thanks to similar characteristics of elements in the two models. To this end, in our development process, the domain engineering process encompasses all activities of this variability modeling process.

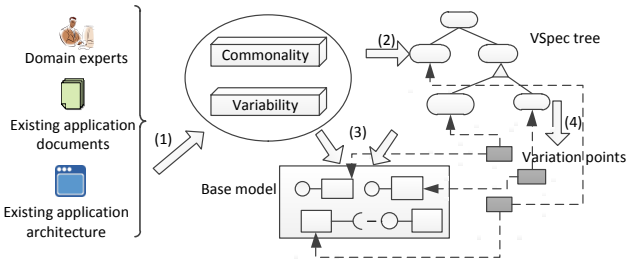

Fig. 3. "Variability-driven process" strategy

B. Product model generation

The first activity in Figure 2 shows the stage of the development process where the variability model is configured and the product model is generated. As previously mentioned, the product model is generated by configuring the variability model based on the resolution model. The task is realized by the AdapSwAG tool. It considers the CVL model as its input. Its output is a product model that contains the necessary components for adaptation. The AdapSwAG tool generates this product model by applying the following rules:

1) A component in the base model will be present and activated in the product if the corresponding *Choice* VSpec is resolved by a True value in the VSpecResolution.
2) A component in the base model will be present in the product at runtime if the corresponding *Choice* VSpec is not resolved by a True value in the VSpecResolution.
3) An attribute in a component in the architecture is assigned the value that is specified to the *value* attribute of its VSpecResolution.
4) Connections between components that are activated in the product are maintained in the product model.

Unlike the CVL approach, the generated product model contains all components in the base model and only connections between components which corresponding *Choice* VSpecs are resolved by True values in the resolution model.

V. Related work

SPL engineering has proven to be a well-suited methodology for developing a diversity of software products and software-intensive systems [11]. Several approaches are based on SPL engineering to develop adaptive software. Parra et al. [3] propose an approach to build a dynamic SPL. In this approach, activities such as analysis, composition generation, transformation, and runtime reconfiguration are separated into domain engineering and application engineering processes as in SPL engineering. Moreover, the role of developers in the process is defined such as application architect, platform architect, and asset developer. Lee et al. in [4] propose an approach to develop dynamically reconfigurable products. They introduce the activities of product line development, e.g., feature and feature binding analysis, behavior and functional specifications of feature binding units, product line architecture, and component design. They provide guidelines for designing dynamically reconfigurable architectures. Authors in [8] represent a process to automatically build a dynamic SPL from a feature model based on the assumption that a feature can be modeled as a component. This process consists of four steps: defining the core and dynamic architecture, adding the configurator, and defining the initial product. The reconfiguration action at runtime is simple to activate or deactivate components. The MADAM approach [5] is based on SPL techniques to build adaptive systems. They propose five steps to develop adaptive applications as follows: identifying fixed and varying user needs and resource constraints, designing the architecture, implementing the components identified by the architecture and deriving runtime plan objects, designing the utility function and property predictors for the components and composition.

Almost all these approaches use a feature model to specify variability. The feature model is configured to generate an initial product. However, they do not define how to specify variability and build an ASA. In our approach, an adaptive software development process based on SPL engineering with a set of specific steps has been proposed to guide engineers building an ASA. Activities to specify the variability model and the base model in our approach are considered as steps in the domain engineering process. The other activities are separated into two subprocesses in the application engineering process. Moreover, our approach identifies an explicit activity for maintaining the software state integrity based on specifying a state transfer model at design time.

VI. Conclusion

In this paper, we have presented a variability-driven development process for building an ASA. The process is based on CVL tools and meta-models to model variability of the software architecture but identifies an ordered set of tasks to be performed by engineers from variability specification up to the software architecture.

In order to validate our process, we have implemented a simple client-server application. The server is implemented with two different versions and can dynamically switch. In order to represent the variability of this application, a VSpec tree in CVL that conforms to the CVL meta-model is finally specified. Its architecture (base model) is specified by using ACME - an ADL. In order to generate a product, a resolution model is specified. Then, AdapSwAG tool generates a product model as an ASA. In this example, we have used the iPOJO component model and the Apache CXF framework to develop components. Based on their specification, a text generation module is implemented by using Xpand generator framework to generate implementation artifacts skeleton that includes packages. Each of them corresponds to a component specified in the product model and consists of component implementations, a component specification represented in XML documents, and a configuration file to create iPOJO components. To this end, the generated packages can be manually completed by an engineer, or the implementation artifacts developed independently by an engineer are integrated into them to build the iPOJO components of the adaptive product. Due to space constraints we can not give more details of our implementation, but the example is available at https://github.com/nthohuynh/.

References

[1] J. Kramer and J. Magee, "The evolving philosophers problem: Dynamic change management," *IEEE Transaction on Software Engineering*, vol. 16, no. 11, pp. 1293–1306, 1990. doi: 10.1109/32.60317

[2] Y. Vandewoude, P. Ebraert, Y. Berbers, and T. D'Hondt, "Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates," *IEEE Transaction on Software Engineering*, vol. 33, no. 12, pp. 856–868, 2007. doi: 10.1109/TSE.2007.70733

[3] C. Parra, X. Blanc, A. Cleve, and L. Duchien, "Unifying design and runtime software adaptation using aspect models," *Science of Computer Programming*, vol. 76, no. 12, pp. 1247 – 1260, 2011. doi: 10.1016/j.scico.2010.12.005

[4] J. Lee and K. C. Kang, "A feature-oriented approach to developing dynamically reconfigurable products in product line engineering," in *Proceedings of the 10th International on Software Product Line Conference*, ser. SPLC '06, Washington, DC, USA, 2006. doi: 10.1109/SPLINE.2006.1691585. ISBN 0-7695-2599-7 pp. 131–140.

[5] S. Hallsteinsen, E. Stav, A. Solberg, and J. Floch, "Using product line techniques to build adaptive systems," in *Proceedings of the 10th International on Software Product Line Conference*, ser. SPLC '06. Washington, DC, USA: IEEE Computer Society, 2006. doi: 10.1109/SPLINE.2006.1691586. ISBN 0-7695-2599-7 pp. 141–150.

[6] R. Capilla, J. Bosch, and K.-C. Kang, "Systems and software variability management: Concepts, tools and experiences," in *Systems and Software Variability Management*, 2013. doi: 10.1007/978-3-642-36583-6. ISBN 978-3-642-36582-9

[7] G. Pascual, M. Pinto, and L. Fuentes, "Self-adaptation of mobile systems driven by the common variability language," *Future Generation Computer Systems*, 2014. doi: 10.1016/j.future.2014.08.015

[8] P. Trinidad, A. R. Cortés, J. Peña, and D. Benavides, "Mapping feature models onto component models to build dynamic software product lines," in *1st SPLC Workshop on Dynamic Software Product Line (DSPL)*, Kyoto, Japan, 2007, pp. 51–56.

[9] O. Haugen, A. Wkasowski, and K. Czarnecki, "Cvl: Common variability language," in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC '13. New York, NY, USA: ACM, 2013. doi: 10.1145/2491627.2493899. ISBN 978-1-4503-1968-3 pp. 277–277.

[10] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, Tech. Rep. CMU/SEI-90-TR-021, 1990.

[11] K. Pohl, G. Böckle, and F. van der Linden, "Software product line engineering: Foundations, principles, and techniques." Springer-Verlag Berlin Heidelberg, 2005. doi: 10.1007/3-540-28901-1. ISBN 978-3-540-28901-2 p. 467.