Towards the Automatic Resolution of Architectural Variability in Software Product Line Architectures through Model Transformations

Jesús Benedé

ISSI Research Group, Universitat Politècnica de València
Camino de Vera, s/n, 46022, Valencia, Spain
jesbegar@inf.upv.es

Abstract. Modelling variability in software product lines (SPL) development is receiving a lot of attention in current years, building on the idea that product derivation could be automatically derived from a product line through model transformations. Software Product Line development involves the explicit management of variability that has to be encompassed by the software artifacts, in particular by the software architecture. This variability has to be supported and represented by the architecture. The Common Variability Language (CVL) allows to represent such variability independently of the Architecture Description Language (ADL). In this research work we define a set of model transformations to support the resolution of the product architecture variability in an automated way by using the Atlas Transformation Language (ATL). The transformation takes as input a multimodel which allows us to represent the relationships between the external variability, represented by a feature model, the architectural variability, represented by the CVL model, and the non-functional requirements, represented in a quality model and generates as output the CVL resolution model that represents the product architecture's variability resolution.

Keywords: Variability Resolution; Architecture Derivation; Software Product Lines; Model Transformations; Atlas Transformation Language.

1 Research Problem and Motivation

A Software Product Line (SPL) is a set of software systems which share a group of specific features and they are developed from a collected and pre-established software assets. One of the key aspects in SPL is the variability management [1].

Variability in the SPL development has to be defined, represented, exploited and implemented [2]. Variability is the ability of a software system or software artifact to be extended, customized or configured for (re-)use in a specific context. This variability should be resolved in derivation time to obtain a specific product from a set of SPL assets. Feature models represent the external variability, which is the "variability of domain artifacts that is visible to customers", and internal variability which is the "var-

iability of domain artifacts that is hidden from customers" [2]. Both external and internal variability are important to the success of developing software artifacts. However, external variability deserves special attention as it is visible to users and relates to requirements defined at early development stages where errors or inaccuracies are relatively inexpensive and easy to detect and correct.

In SPL software architectures play a dual role: on the one hand, the product line architecture (PLA) should provide variation mechanisms which will help us to achieve a set of allowed variations, and on the other hand, the derived product architecture (PA) from the PLA by exercising its built-in architectural points [1].

For enabling the automatic resolution of the PLA variability is required not only that the architectural description languages provide variation mechanisms, but also to explicitly represent how the different variants realize the external variability usually represented in feature models.

The Common Variability Language (CVL) [3] is a language that allows the specification of variability over any EMF-based model, and supports the automatic resolution of the variability and the derivation of resolved models. In CVL approach, we can distinguish three models: i) The base model, which is a model described in a Domain Specific Language; ii) The variability model, which is the model that defines variability on the base model; iii) The resolution model, that describes how the variability has been resolved with the aim to create a new model in the base DSL.

For a given base model, we can derive several resolution models, that, through a generic CVL model-to-model transformation can generate the product specific models, which conforms to the base DSL.

CVL incorporates its own variability mechanisms, and concepts like *type, composite variability, constraint and iterator,* will help us to mimic feature diagrams. Moreover, there are lower level concepts (e.g., fragment substitution, object existence) that can be used to extend those provided by the Architectural description languages (ADLs).

The main goal of this research work is the development of ATL model transformations which will allow us to automate the resolution of variability on PLAs, obtaining the CVL resolution models of the architecture of the product under development. These transformations take as input a product configuration which comprises not only functional but also non-functional requirements (NFRs).

Such transformations are developed by using ATL [4], an imperative-declarative language, based in OCL and developed by ATLAS group in response to the QVT standard from the OMG. Once obtained the CVL resolution models will be used to obtain the PA through a CVL model transformation.

2 Background and Related Work

In a Software Product Line (SPL) scenario it is difficult to manage the derivation of the architectural specification of a product (PA), especially when the SPL allows a wide range of variability. CVL has been used in several approaches to support the resolution of the architectural variability and the derivation of product architectures in SPL development (e.g., [5], [6], [7])

Nascimento et al. [5] present an approach for defining product line architectures using CVL. They apply the Feature-Architecture Mapping Method (FArM) to filter the feature models in order to consider only the architectural-related features. These features will form the CVL specification that will allow obtaining the COSMOS* architectural models. They do not define relationships between the external variability model (features model) and the architectural variability expressed in CVL and thus the derivation of the product architecture taking as input the configuration is not supported. They explicitly omit the non-functional requirements when applying the FArM method.

Svendsen et al.[6] present the applicability of CVL for obtaining the product models for a Train Control SPL that are defined using a DSL. They only consider the explicit definition of the internal variability and consequently, the configuration should be made directly over the CVL specification of the internal variability.

Combemale et al. [7] present an approach to specify and resolve variability on Reusable Aspect Models (RAM), a set of interrelated design models. They use CVL to resolve the variability on each model and then compose the corresponding reusable aspects by using the RAM weaver. They also consider just the internal variability, and the configuration should be made over the CVL specification.

Summarizing, the approaches described above do not establish relations among the SPL external variability and the architectural variability nor consider non-functional requirements in the process. The establishment of connections between the SPL external variability, the non-functional requirements, represented in a quality model, and the architectural variability, represented by using CVL allows us to automatically solve the architectural variability by using ATL model transformations.

3 Contribution

Our approach for the automatic resolution of the PLA in architectural models through ATL transformation is supported by the use of a multimodel [8, 9] that allows the explicit representation of relationships among entities in different viewpoints. The problem of representing and automatically resolving the architectural variability taking into account functional and non-functional requirements requires (at least) three viewpoints:

- **The variability viewpoint** represents the SPL external variability expressing the commonalities and variability within the product line. It has a main element, *feature*, expressed by means of a variant [10] of the cardinality found in the feature model.
- **The architectural viewpoint** represents the architectural variability of the Product Line architecture that realizes the external variability of the SPL expressed in the variability viewpoint. It is expressed by means of the Common Variability Language (CVL) and its main element is the Variability Specification (VSpec) [3].
- **The quality viewpoint** represents the hierarchical decomposition of quality into sub-characteristics, quality attributes, metrics and the impacts and constraints among quality attributes. It is expressed by means of a quality model for software product lines [11], that allows the representation of NFRs as constraints.

In the multimodel we can also specify how a given feature *is_realized_by* [12] a set of VSpecs or how a given NFR *is_realized_by* [12] a set of VSpecs or how a given VSpec *impacts* [8] positively or negatively on a quality attribute.

By using the multimodel and the product configuration, which comprises features and NFRs that the product must fulfill together with the quality attributes priorities, in this research work we focus on the definition of a set of ATL model transformation to support the resolution of the architectural variability and the automatic generation of CVL resolution models.

The quality attribute priorities together with the impacts that VSpecs have on the quality attributes are used to choose from a set of architectural variants that having the same functionality differ in their quality attribute levels. Fig. 1 shows the structure of the ATL transformation rules. Although the transformations have been defined using ATL, Fig. 1 shows the structure using QVT [13] graphical syntax due to clarity and simplicity reasons. Fig. 1(a) shows how we use the *is_realized_by* relationship between features and VSpec to positively resolve the $VSpec_b$ when we select F_a in the configuration. Fig. 1(b) shows how we use the *is_realized_by* relationship between NFRs and VSpec to positively resolve the $VSpec_b$ when we select NFR_{is} in the configuration. Similarly, we positively resolve a given VSpec when it is the alternative that best fits the quality attributes priorities combined with the the *impact* relationships between VSpecs and quality attributes by implementing the AHP [14] method.

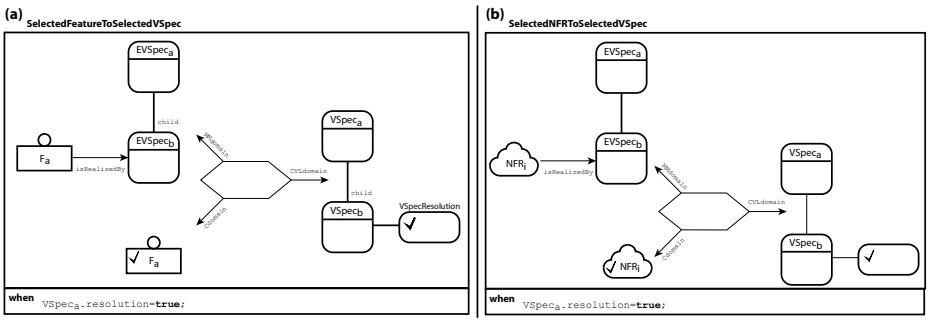

Fig. 1. Transformation Rules Graphical Syntax

These transformations are tested in an automotive case study for obtaining the product architecture of vehicle control systems[1]. Fig. 2 shows an excerpt of an example of the CVL resolution model generation of this case study. The configuration consists on the following features: *ABS*, *Traction control*, *Stability Control* and *Cruise Control* and a prioritized quality attribute: *Latency Time*. This *Cruise Control* feature can be realized by two different architectural variants *Simple* (which impacts positively on *Latency Time*) and *Adaptive* (which impacts negatively on *Latency Time*). The transformation process positively resolves *SimpleCruisseControl* System VSpec since has a positive impact on the prioritized quality attributes.

[1] The whole specification of the case study is available for download at: http://users.dsic.upv.es/~jagonzalez/CarCarSPL/links.html

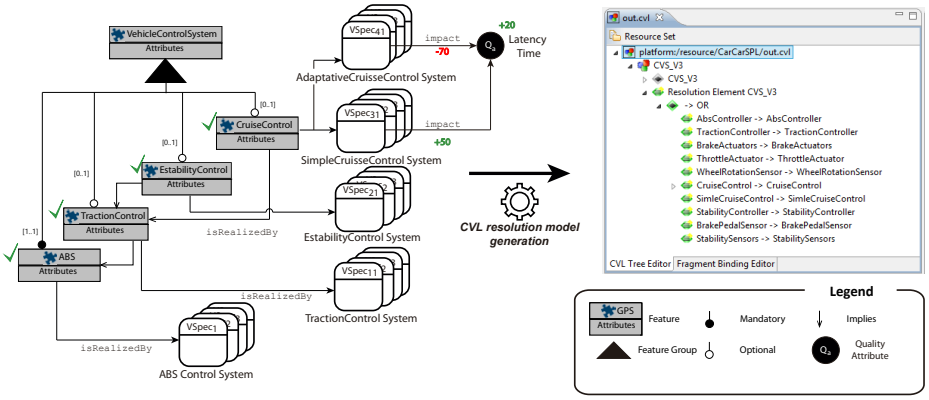

Fig. 2. CVL Resolution Generation Excerpt

These ATL transformations will be also employed in a cloud system service context, with the purpose of performing cloud applications derivation from other existing cloud services that incorporate variability. This variability could respond to the existence of multiple services with the same functionality and different non-functional properties, or because the very same services still include the variability through configuration interfaces. The transformation generates as out choreography and orchestration models of our new service.

At the moment, we are not providing a full coverage of CVL language. We are now focusing on the *Fragment Substitution* and *Object Substitution* variability types [15] to express the architectural variability.

4 Conclusions

We have presented our approach to automatically derive the CVL resolution models for a given product configuration by using an ATL transformation.

We also show how quality assurance techniques can be introduced in SPL product architecture derivation, and how the Model Driven Software Development principles are going to help us to resolve the automatic resolution of variability in product line architectures. A model transformation chain (ATL+CVL transformations) allow us to obtain the first version of the architecture of the product under development.

We are now working on the test of the ATL transformations in real scenarios and on the integration with the CVL supporting tool so as to enable the CVL transformation that generates the models that represent the architecture of the product under development. We also plan to incorporate other CVL variation mechanisms that will enrich the expressiveness of the architectural variability.

5 References

1. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley Professional (2001).
2. Van der Linden, F., Schmid, K., Rommes, E.: Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering. Springer Berlin Heidelberg (2007).
3. Object Management Group: Common Variability Language (CVL) OMG Revised Submission. (2012).
4. Jouault, F., Allilaire, F., Bézivin, J.: ATL: a QVT-like transformation language. Companion to 21st ACM SIGPLAN Symp. Object-oriented Program. Syst. Lang. Appl. 719–720 (2006).
5. Nascimento, A.S., Rubira, C.M.F., Burrows, R., Castor, F.: A Model-Driven Infrastructure for Developing Product Line Architectures Using CVL. 7th Brazilian Symposium on Software Components, Architectures and Reuse. pp. 119–128. , Brasilia, Brazil (2013).
6. Svendsen, A., Zhang, X.: Developing a software product line for train control: a case study of CVL. 14th Software Product Line Conference. pp. 106–120. , Jeju Island, South Korea (2010).
7. Combemale, B., Barais, O., Alam, O., Kienzle, J.: Using CVL to operationalize product line development with reusable aspect models. Workshop on Variability Modeling Made Useful for Everyone. pp. 9–14. , Innsbruck, Austria (2012).
8. González-Huerta, J., Insfran, E., Abrahão, S.: A Multimodel for Integrating Quality Assessment in Model-Driven Engineering. 8th International Conference on the Quality of Information and Communications Technology. pp. 251–254. , Lisbon, Portugal (2012).
9. González-Huerta, J., Insfrán, E., Abrahão, S.: Defining and Validating a Multimodel Approach for Product Architecture Derivation and Improvement. 16th International Conference on Model-Driven Engineering Languajes and Systems. pp. 388–404. , Miami, USA (2013).
10. Gómez, A., Ramos, I.: Cardinality-Based Feature Modeling and Model-Driven Engineering : Fitting them Together. International Workshop on Variability Modelling of Software-Intensive Systems. pp. 61–68. , Linz, Austria (2010).
11. González-Huerta, J., Insfran, E., Abrahão, S., McGregor, J.D.: Non-functional requirements in model-driven software product line engineering. Proceedings of the Fourth International Workshop on Nonfunctional System Properties in Domain Specific Modeling Languages. pp. 1–6. , Innsbruck, Austria (2012).
12. Janota, M., Botterweck, G.: Formal approach to integrating feature and architecture models. 11th Conference on Fundamenteal Approaches to Software Engineering. pp. 31–45. , Budapest, Hungary (2008).
13. Object Management Group: Meta Object Facility (MOF) 2.0 Query / View / Transformation Specification. (2008).
14. Saaty, T.: Decision making with the analytic hierarchy process. Int. J. Serv. Sci. 1, 83–98 (2008).
15. Haugen, Ø., Moller-Pedersen, B., Olsen, G.K., Svendsen, A., Fleurey, F., Zhang, X.: Consolidated CVL language and tool. , MoSiS Project, D.2.1.4., SINTEF, Univeristy of Oslo (2010).