

# Tool for Configuring Product Individuals from Configurable Software Product Families

Varvana Myllärniemi, Timo Asikainen, Tomi Männistö, and Timo Soininen

Helsinki University of Technology  
Laboratory of Software Business and Engineering  
`varvana.myllarniemi@hut.fi`

**Abstract.** This position paper presents a tool for configuring product individuals from configurable software product lines. The product derivation is done by making selections based on a configuration model in a way that best satisfies customer requirements at hand. The tool produces a description of the product individual as an output. The implementation is based on both component-based and feature-based modelling techniques. The tool employs techniques from traditional product configuration to ensure the validity of the product individual against the model. The implementation of the tool is still in progress.

## 1 Introduction

This position paper presents a tool for configuring product individuals from configurable software product lines. The implementation is still in progress, so this paper describes the current status of the tool. This section covers the theoretical background and basics of the tool. Section 2 discusses the implementation of the tool, and section 3 draws conclusions and suggestions for future work.

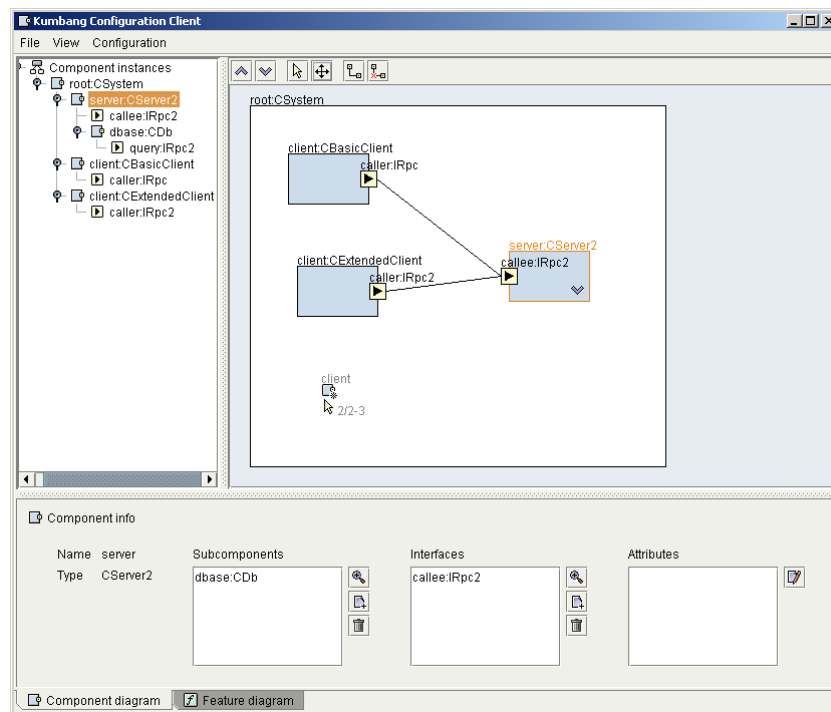
This work is based on the concept of a *configurable software product family* (see e.g. [1]). Configurable software product families correspond to the highest maturity level of software product lines [2]: all development effort has moved from application engineering towards domain engineering. During product derivation, there is no need for additional product-specific coding. Instead, product individuals are derived from predesigned software components and a predesigned configuration model in a routine manner. The output of this configuration task is a *configuration*, which is a description of the product individual.

The tool is based on two modelling languages that describe the structure and variability of the family. Koalish [3] is a architecture-driven modelling language that is derived from architecture description language Koala by adding mechanisms for variability. A Koalish system consists of components, interfaces, bindings between interfaces, attributes and constraints. Components form a compositional hierarchy that constitutes the configuration. (For more information about Koalish, please refer to [3] or [4].) The other language employed is feature-driven language called Forfamel that synthesises existing feature modelling languages. Forfamel offers features, attributes and constraints between them, as well as a compositional hierarchy of features. This language is still under construction.

The tool helps the user in deriving a product individual that conform both to the product family architecture and to the customer requirements. The user



Our tool is based on two modelling languages for configurable software product families (see section 1). Thus the tool employs both architecture-driven and feature-driven approaches: the configuration consists of features, software components, attributes, interfaces and bindings between these interfaces. For example, the user might say “I want component **client** connected to component **server** through interface **caller**” or “I want attribute **bandwidth** in feature **connection** to have value **high**”. The user then makes these selections by modifying the configuration with the graphical user interface (see figure 2). There is no pre-set order for these modifications, except that one must add components and features in a top-to-bottom order in the compositional hierarchy.



**Fig. 2.** A screen shot from the configuration client. The client shows a compositional hierarchy of the current configuration. This is the component view that shows the components, interfaces and their connections.

The tool offers a distributed client-server architecture: clients offer a graphical user interface and are connected to the configuration server. This enables centralized management of the models: the user of the configuration client does not need to possess the configuration model in order to use the tool.

For the selected configuration, the tool must check the validity of the configuration, which includes both completeness (all the necessary selections are made) and consistency (no rules of the model are violated). This reasoning uses the inference tool *smodels* [7]. The same reasoning mechanism is used in an aca-

demic product configurator for traditional products [8]. In addition, the server can deduct some of the consequences of the selections so far, for example, to tell which instances must be present in the configuration and which instances must not be present.

In addition, the tool produces a description of the complete configuration, both as text and extension markup language (XML).

### 3 Conclusions and Future Work

In this position paper we presented a configurator tool for configurable software product lines. The tool is still under construction, so there is some planned functionality that has not been yet implemented.

In order to validate our tool, we are working on a real-world case for configuring product individuals. This requires that we construct a configuration model that represents the case product family, and then derive products using our configurator tool.

A large part of the tool support for configurable software product families is the modelling tool. At the moment, all configuration models have to be written by hand. This is not a feasible approach, since it requires the user to be familiar with the syntax of the language. In addition, when models are large, writing them by hand might become too demanding. A semi-graphical modelling tool would ease this task and provide a visual aid in model creation.

### References

1. Männistö, T., Soininen, T., Sulonen, R.: Product configuration view to software product families. In: International Workshop on Software Configuration Management (SCM-10) at ICSE 2001. (2001)
2. Bosch, J.: Maturity and evolution in software product lines: Approaches, artefacts and organization. In Chastek, G.J., ed.: Proceedings of the Second Software Product Line Conference (SPLC2). (2002) 257–271
3. Asikainen, T., Soininen, T., Männistö, T.: A Koala-based ontology for configurable software product families. In: IJCAI 2003 Configuration workshop. (2003)
4. Asikainen, T., Soininen, T., Männistö, T.: A Koala-based approach for modelling and deploying configurable software product families. In: Proceedings of the Fifth International Workshop on Product Family Engineering (PFE-5). (2003)
5. van der Hoek, A., Heimbigner, D., Wolf, A.: Capturing architectural configurability: variants, options and evolution. Technical Report CU-CS-895-99, Department of Computer Science, University of Colorado, Boulder, Colorado (1999)
6. Hotz, L., Krebs, T.: Supporting the product derivation process with a knowledge-based approach. In: Proceedings of Software Variability Management ICSE 2003 Workshop. (2003) 24–29
7. Simons, P., Niemelä, I., Soininen, T.: Extending and implementing the stable model semantics. Artificial Intelligence **138** (2002) 181–234
8. Tiihonen, J., Soininen, T., Niemelä, I., Sulonen, R.: A practical tool for mass-customising configurable products. In: Proceedings of the 14th International Conference on Engineering Design (ICED'03). (2003)