Pre-Packaged Variability for Product Derivation in Product Lines

Thorsten Krebs[1] and **Lothar Hotz**[2] and **Katharina Wolter**[3]

Abstract. Product configurators are widely used to generate products out of the vast amount of potential variants in product families. This is a complex task that is hampered by the fact that configuration models are abstractions of the real world and cannot contain all dependencies between the artifacts and the environment. For embedded systems this is crucial because calibration is used to adjust the product to its operating environment. In this paper we show how pre-packaged variability makes the configuration of complex embedded systems easier. Packages of artifacts known to work well together are introduced into the configuration model and used during product derivation. Functionality of these artifacts is shown to the outside of the package. Based on this functionality the packages are selected and integrated into the product. We give an industrial example and show how the concept of packages can be formalized in configuration models used by the structure-based configuration approach.

1 Introduction

Product lines are a means of large scale reuse of *artifacts* in domains with lots of products that "share a common, managed set of features satisfying the specific needs of a particular customer or market segment" [1]. Features are "prominent or distinctive user-visible aspects of a system or systems" [6]. Substantial production economies are achieved by deriving products from a common set of artifacts in a prescribed way, in contrast to developing separately, form scratch or in arbitrary fashion. Thus, product lines provide reuse strategies and thereby reduce development time and cost. Reusable core artifacts form the basis for routine derivation. However, product-specific adaptations might still be necessary to derive the requested product.

Configuration is a well known approach to support the composition of products out of a set of given domain objects. The configuration of technical systems is one of the most successful application areas of knowledge-based systems [4]. In structure-based configuration, domain objects are represented in *configuration models* to abstract from the "real world". Configuration models describe the admissible combinations of the predefined objects in a specific domain – e.g. the configurable artifacts of a product line. Basic modeling facilities enable the differentiation between three kinds of knowledge:

1. With **conceptual knowledge** domain objects are represented with concepts, taxonomic and compositional relations as well as restrictions between arbitrary concepts and their properties (by means of constraints).
2. **Procedural knowledge** declaratively describes the configuration process – i.e. the order in which decisions are processed.
3. A **task specification** specifies properties and constraints that a product must fulfill.

The configuration process itself is performed in an incremental approach, where each step represents a configuration decision. After each step, optionally constraint propagation and consistency checking can be applied [2, 5]. Detected conflicts[4] are handled by conflict resolution mechanisms.

Product derivation can be divided into two parts, i.e. the configuration process where a description of the product is derived and the realization process where the "real" artifacts are generated and assembled. In this paper we focus on the configuration part. Thereby, configurable artifacts and features are seen as domain objects and represented with the modeling facilities mentioned above.

In a perfect world there may be a perfect model perfectly reflecting reusable artifacts and their dependencies. But in real life it is quite common that not everything is modeled – indeed a model is used to abstract from reality and to make configuring complex products easier. Software for embedded systems often has to be calibrated. Here it is difficult, if not impossible, to foresee, model and configure everything before the software can be installed and tested. To go round this problem, *pre-packaged variability* is used. Next to the single artifacts, a packaged set of artifacts that are known to work together can be used without "reinventing the wheel". This means that packages are used just like single artifacts – i.e. selected to fulfill requested functionality – in the configuration process.

The remainder of this paper is organized as follows: in Section 2 we explain the concept of packages and define requirements for the integration in the structure-based configuration methodology. In Section 3 we give an example application domain and show how we realized pre-packaged variability with the configuration tool KONWERK [3]. Finally, in Section 4 we give a conclusion.

2 Packages

A *package* is a pre-defined, valid combination of artifacts that also exist as single definitions in the configuration model. Therefore, no new functionality is introduced by defining a package. However, the possibility to choose a package instead of choosing multiple artifacts (and configuring their properties and dependencies) makes it easier to reuse commonly used sets of artifacts throughout a large number of configured products. Four concept types are concerned: artifacts, features, packages and the system.

[1] Laboratory for Artificial Intelligence, University of Hamburg, Germany email: krebs@informatik.uni-hamburg.de
[2] HITeC c/o University of Hamburg, Germany email: hotz@informatik.uni-hamburg.de
[3] Laboratory for Artificial Intelligence, University of Hamburg, Germany email: kwolter@informatik.uni-hamburg.de

[4] A conflict is defined as a situation where the decisions made by the user, their logical impacts and the configuration model are not consistent.

- Artifacts *realize* features. Multiple artifacts can together realize one feature ($n-1$) or each realize a single feature ($1-1$) and one artifact can realize multiple features ($1-n$).
- Packages *contain* artifacts and *supply* the corresponding features. One package can contain multiple artifacts. In the definition artifacts can be referred to in multiple packages – artifact instances belong to exactly one package instance.
- Systems *have packages*, *have artifacts* and *have features*. One system can have multiple packages but (also for reasons of structuring and maintenance) one package belongs to exactly one system.

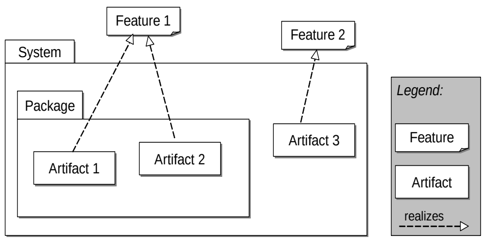

Figure 1. Interplay between System, Package, Artifact and Feature

In Figure 1, a package containing two artifacts that together realize a feature is depicted. Furthermore, single artifacts can coexist next to packages. Therefore, the system represented in this figure consists of three artifacts and offers two features.

For including packages in the configuration model, the compositional relation as it is defined in structure-based configuration has to allow for shared objects. When a package contains an artifact, the system that contains this package, also contains the artifact – i.e. the relation is *transitive* and the artifact is part of both, the package and the system. Thus, for the compositional relation a $n-m$ relation is used – i.e. one feature can be *realized by* n artifacts that together realize $m-1$ other features.

For a *realizes* relation between an artifact and a feature this goes even one step further. The same feature can be realized by multiple artifacts but when multiple instances of that artifact exist, the feature still is realized only once. *Constraints* are needed to ensure this: when an artifact that realizes a feature is instantiated and an instance of that feature already exists, then this feature instance is automatically taken for the *realizes* relation of that artifact. Therefore, this feature is shared by multiple artifacts.

3 Example Application Domain

As an example we give the product family of Car Periphery Supervision (CPS) systems introduced by [9]. A CPS system consists of automotive systems that are based on sensors installed around the car to monitor its local environment. Sensor measurement methods and evaluation mechanisms provide information for various kinds of high-level comfort and safety related applications like Parking Assistance and Pre-Crash Detection. In such systems, usually sets of sensors are mounted on the vehicle (e.g. ultrasonic sensors hidden in the bumper).

The general ideas presented in Figure 1 are formalized in a configuration model for KONWERK. Additionally, the CPS-specific artifacts have been placed under these concepts. Parts of this example model also shown in Figure 2 are:

Hardware Typical hardware artifacts in CPS systems are sensors. These are grouped into front and rear sensors as well as into ultrasonic and short range radar (in the figure there is only US).

Software The only Software in our example model are applications, i.e. parking assistance and pre-crash detection.

Features Two features are modeled: front supervision and rear supervision.

System The CPS system is the goal concept (i.e. the task specification). It contains hard- and software artifacts and packages, and realizes features.

Package One package – a rear sensor set – is defined. It contains four ultrasonic rear sensors and realizes rear supervision.

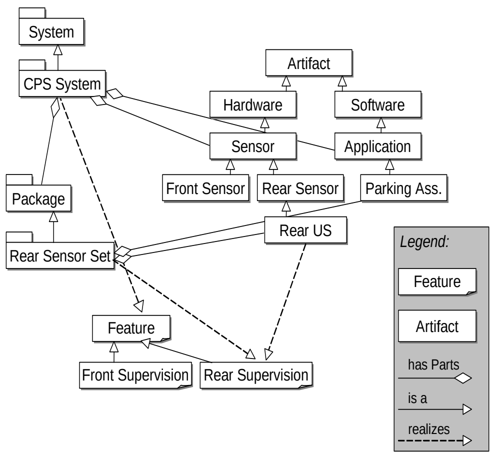

Figure 2. Example from the CPS Domain

3.1 Implementation

All sensors in one package have to be of the same type – e.g. no ultrasonic and radar sensors should be mixed. This can be easily modeled with a constraint that sets the type of all instances of the concept sensors in one package to be equal so that different sensors in one package are explicitly ruled out.

A *number restriction constraint* ensures that features *realized by* the artifacts inside a package are also shown to the outside of this package. This constraint sets the number of features in the *supplies* relation of the package to the number of features that the artifacts together *realize*. The abstract package concept can *supply* an arbitrary number of features; a specific package contains a fixed number of artifacts and therefore *supplies* a fixed number of features (e.g. the rear sensor set *supplies* exactly one feature: rear supervision). This means, each sensor of the rear sensor set realizes rear supervision – but all sensors together still realize this feature only once, not once for every sensor. To get the same feature instance in the *realizes* relation of all sensors we extended this constraint such that it relates the feature instances and the artifact instances. This is done by setting the corresponding *realizes* and *realized by* entries accordingly.

3.2 Process

In the structure-based configuration process the order of configuration decisions can be defined with procedural knowledge. For using pre-packaged variability, however, certain aspects have to be restricted:

1. In a feature-driven configuration approach (see e.g. [5]) first the product capabilities are selected. This is achieved by selecting the system as goal concept and focusing on the *has feature* relation of the system in the first configuration step.

2. Based on this selection, the mandatory artifacts are inferred. This is achieved by mappings defined between features and artifacts in the configuration model. After this, the artifacts can be further configured.

Our approach first checks whether a package is defined that realizes the requested product capabilities. When this is the case, the complete package is instantiated and included in the *partial configuration*. When no package can be found such that a subset of the requested features can be satisfied, single artifacts are selected, instantiated and integrated into the configuration. This is achieved by defining a focus on the *supplied by* relation (inverse of *supplies*) that is defined for packages – single artifacts and features are connected via the *realizes* relation. Thus, with this focus packages are preferred over single artifacts. This approach can be applied after the features of the CPS system have been determined.

Conflict situations regarding the configuration of packages can be handled with the standard conflict resolution mechanisms from structure-based configuration since these are defined for all concept types and a package is a concept definition.

We successfully implemented the approach shown here with the CPS example (as a configuration model for KONWERK). The configuration solution presented in Figure 3 shows a CPS system that realizes rear supervision. This rear supervision is supplied by a rear sensor set containing four rear ultrasonic sensors. It is clearly visible how only one feature instance (`Feature-2`) has been instantiated and propagated to all hardware artifacts and the package and system.

4 Conclusion

We have shown how pre-packaged variability can be integrated into structure-based configuration. Therefore we provide a configuration model with a special type of concept named *package*. Packages contain an arbitrary number of artifacts and show the functionality these artifacts realize as also their own functionality. We have further described how to cope with the fact that one feature can be realized by multiple artifacts and how the same feature instance can be propagated to the instances of packages and systems. Finally, we have sketched how the configuration process can be modeled to prefer packages over single artifacts.

The idea of composed variability as a (former) subproblem is not new as it was e.g. previously addressed as the composite constraint satisfaction problem [7]. Having packages of artifacts next to the definition of these single artifacts – as some sort of combination of former cases and the current configuration problem – in structure-based configuration models, however is a novel approach.

New packages can be defined by simply configuring them. However, tool support for maintaining package definitions in the configuration model is not yet available. Methods from *case-based* reasoning can be applied to achieve this (see e.g. [8] for case-based reasoning in software reuse).

Another topic of future research is how the selection of packages and single artifacts can be improved. Selecting one package e.g. can rule out the selection of two further packages that together would supply all the requested product capabilities. When the one package only supplies part of this functionality, the remaining artifacts have to be configured "the traditional way".

ACKNOWLEDGEMENTS

This research has been supported by the European Community under the grant IST-2001-34438, ConIPF - Configuration in Industrial Product Families.

REFERENCES

[1] J. Bosch, G. Florijn, D. Greefhorst, J. Kuusela, H. Obbink, and K. Pohl, 'Variability Issues in Software Product Lines', in *Proc. of the Fourth International Workshop on Product Family Engineering(PFE-4)*, Bilbao, Spain, (October 3-5 2001).

[2] A. Günter, *Wissensbasiertes Konfigurieren*, Infix, St. Augustin, 1995.

[3] A. Günter and L. Hotz, 'KONWERK - A Domain Independent Configuration Tool', *Configuration Papers from the AAAI Workshop*, 10–19, (July 19 1999).

[4] A. Günter and C. Kühn, 'Knowledge-based Configuration - Survey and Future Directions', in *XPS-99: Knowledge Based Systems, Proceedings 5th Biannual German Conference on Knowledge Based Systems*, ed., F. Puppe, Springer Lecture Notes in Artificial Intelligence 1570, Würzburg, (March 3-5 1999).

[5] L. Hotz and T. Krebs, 'Supporting the product derivation process with a knowledge-based approach', in *Proc. of Software Variability Management Workshop at ICSE 2003*, Portland, Oregon, USA, (May 3rd 2003).

[6] K. Kang, S. Cohen, J. Hess, W. Novak, and S. Peterson, 'Feature-oriented Domain Analysis (FODA) Feasibility Study', *Technical Report CMU/SEI-90-TR-021*, (1990).

[7] D. Sabin and E.C. Freuder, 'Configuration as Composite Constraint Satisfaction', pp. 153–161. AAAI Press, (1996).

[8] M. Sasikumar, 'Case-based Reasoning for Software Reuse', in *Knowledge Based Computer Systems-Research and Applications (International Conference on Knowledge-Based Computer Systems)*, pp. 31–42, Bombai, India, (December 12-15 1996). Narosa Publishing House, London.

[9] S. Thiel, S. Ferber, T. Fischer, A. Hein, and M. Schlick, 'A Case Study in Applying a Product Line Approach for Car Periphery Supervision Systems', in *Proceedings of In-Vehicle Software 2001 (SP-1587)*, pp. 43–55, Detroit, Michigan, USA, (March, 5-8 2001).

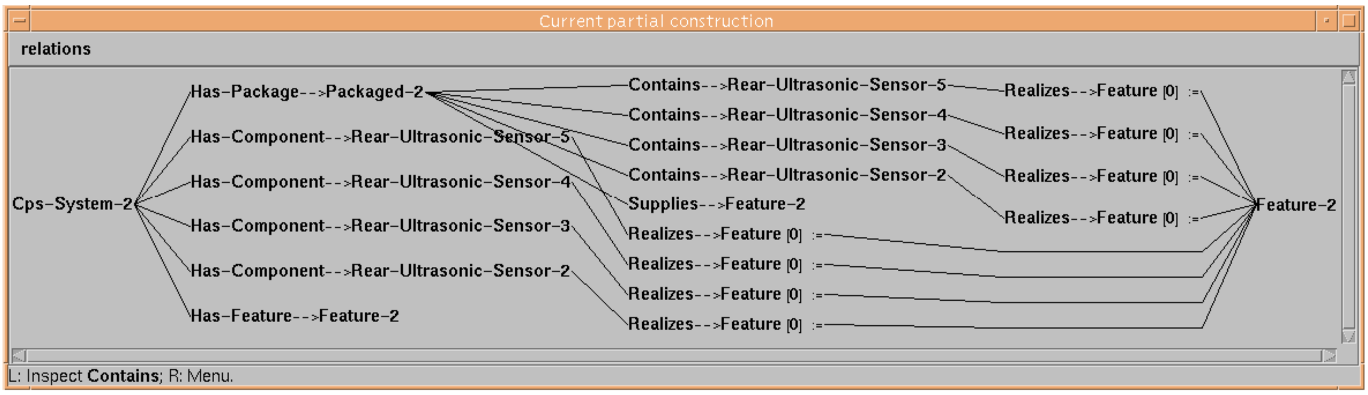
Current partial construction
relations
Cps-System-2
Has-Package-->Packaged-2
Has-Component-->Rear-Ultrasonic-Sensor-5
Has-Component-->Rear-Ultrasonic-Sensor-4
Has-Component-->Rear-Ultrasonic-Sensor-3
Has-Component-->Rear-Ultrasonic-Sensor-2
Has-Feature-->Feature-2
Contains-->Rear-Ultrasonic-Sensor-5
Contains-->Rear-Ultrasonic-Sensor-4
Contains-->Rear-Ultrasonic-Sensor-3
Contains-->Rear-Ultrasonic-Sensor-2
Supplies-->Feature-2
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Realizes-->Feature [0] :=
Feature-2
L: Inspect Contains; R: Menu.

Figure 3. Example Configuration Solution