Binding Time Based Concept Instantiation in Feature Modeling

Valentino Vranić and Miloslav Šípka

Institute of Informatics and Software Engineering
Faculty of Informatics and Information Technology
Slovak University of Technology, Ilkovičova 3, 84216 Bratislava 4, Slovakia
vranic@fiit.stuba.sk, miloslav.sipka@gmail.com

Abstract. In this paper, we address the issue of concept instantiation in feature modeling with respect to binding time. We explain the impact of such instantiation on applying constraints among features expressed in feature diagrams and as additional constraints and propose a way to validate a concept instance under these new conditions.

1 Introduction

Feature modeling aims at expressing concepts by their features as important properties of concepts taking into account feature interdependencies and variability in order to capture the concept configurability [3]. A concept is an understanding of a class or category of elements in a domain [3]. Individual elements that correspond to this understanding are called *concept instances*. While a concept represent a whole class of systems or parts of a system, an instance represents a specific configuration of a system or a part of a system defined by a set of features. Concept instances may be used for feature model validation and manual or automatic configuration of other design models or program code of specific products in a domain [2, 3].

When designing a family of systems, we have to balance between statically and dynamically bound features. In general, dynamic binding is more flexible as we may reconfigure our system at run time, while static binding is more efficient in terms of time and space. Although they often embrace the information on feature binding time, contemporary approaches to feature modeling do not consider the time dimension during concept instantiation.

This paper focuses on the issue of concept instantiation (Sect. 2) and validation of concept instances with respect to binding time (Sect. 3). The paper is closed by a discussion (Sect. 4).

2 Concept Instantiation in Time

Binding time describes *when* a variable feature is to be *bound*, i.e. selected to become a mandatory part of a concept instance. The set of possible binding times depend on a solution domain. For compiled languages they usually include source time, compile time, link time, and run time [1].

An instance I of the concept C at time t is a concept derived from C by selecting its features which includes the C's concept node and in which each feature f whose parent is included in I obeys the following conditions:

1. If f is a mandatory feature, f is included in I.
2. If f is a variable feature whose binding time is earlier than or equal to t, f is included in I or excluded from it according to the constraints of the feature diagram and additional constraints associated with it. If included, the feature becomes mandatory for I.
3. If f is a variable features whose binding time is later than t, f may be included in I as a variable feature or excluded from it, or the constraints (both feature diagram and additional ones) on f may be made more rigid as long as the set of concept instances available at later instantiation times is preserved or reduced.

As follows from this definition,[1] a feature in a concept instance may be bound, in which case it appears as a mandatory feature, or unbound, in which case it stays variable. Mandatory features and features bound in previous instantiations are considered as bound. A concept instance may be instantiated further at later instantiation times.

The constraints—both feature diagram and additional ones—on a variable features whose binding time is later than the instantiation time may be made more rigid as long as the set of concept instances available at later instantiation times is preserved or reduced. An example of this is a transformation of a group of mandatory or-features (Fig. 1a) into a group of alternative features (Fig. 1b).

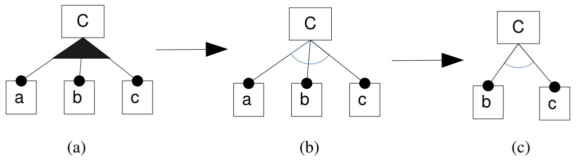

Fig. 1. Reducing the set of concept instances.

Variable features with binding times later than the instantiation time are potentially part of concept instances at later binding times. Again, such features may be excluded at instantiation times earlier than their binding times as long as the set of concept instances available at later instantiation times is preserved or reduced. Consider a group of three alternative features (Fig. 1b) with run-time binding. At source time, one of these features may be excluded (Fig. 1d). However, none of the two remaining features may be excluded since preserving

[1] The definition is based on our earlier concept instance definition [7].

only one of them will force us to make it mandatory, which is illegal, or optional, which will allow an originally unforeseen concept instance to be created: the one with no features from the group.

3 Concept Instance Validation

A concept instance is valid if its features satisfy the constraints. In general, a constraint—be it a feature diagram constraint or an additional one— may be evaluated only if all the features it refers to are bound. However, some logical expressions can be evaluated without knowing the values of all of their variables. Suppose we are instantiating a simple concept in Fig. 2a at source time (with no additional constraints). If we bind the x feature, the or-group constraint will be satisfied regardless of the y feature binding. Thus, we may omit this constraint transforming the y feature into an optional one as shown in Fig. 2b.

It is also possible to omit x. The only possibility for y is to leave it optional, as shown in Fig. 2c, but it has to be assured it will finally be bound (which can be done only at run time). For this purpose, we must add a trivial constraint to this instance: y (y has to be true, i.e. bound).

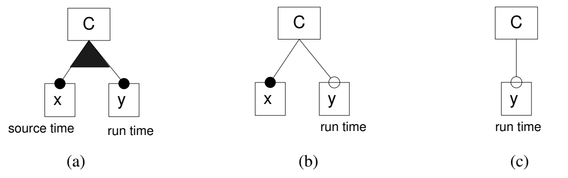

Fig. 2. Dealing with features whose binding time is later than the instantiation time.

By excluding features from feature diagrams, the feature diagram constraints are gradually relinquished. After a successful concept instance validation, all additional constraints that refer to the features whose binding time is not later than the instantiation time can be safely removed from the model. All other constraints have to be postponed for further instantiation.

4 Discussion

In this paper, we presented an approach to concept instantiation with respect to binding time. We analyzed the impact of introducing the time dimension into concept instantiation on concept instance validation with respect to both feature diagram and additional constraints. We have also developed a prototype tool that supports such instantiation (available at `http://www.fiit.stuba.sk/~vranic/fm/`).

Concept instantiation with respect to feature binding time is similar to staged configuration of feature models proposed in conjunction with cardinality-based feature modeling [5, 6]. Although consecutive work [4] mentions a possibility of defining configuration stages in terms of the time dimension, this approach does not elaborate the issue of feature binding time with respective consequences on validation of concept specializations.

Concept instantiation with respect to binding time can be used to check for “dead-end” instances that may result into invalid configurations of a running system. Such configuration may miss some features required by other, bound features, which will lead to a system crash if such features are activated. Similarly as staged feature model configuration, concept instantiation with respect to binding time could be used for creating specialized versions of frameworks [5], which would represent a source time instantiation, and in software supply chains, optimization, and policy standards [4].

Partial validation of the constraints that incorporate unbound features may be improved by transforming them into the normal conjunctive form. This would enable to extract parts of such a constraint with bound features, while conjuncts with unbound features would be simple enough to directly determine whether they can be evaluated or not. As a further work, we plan to explore consequences of applying this approach to cardinality-based feature models [5].

Acknowledgements The work was supported by Slovak Science Grant Agency VEGA, project No. 1/3102/06, and Science and Technology Assistance Agency of Slovak Republic under the contract No. APVT-20-007104.

References

[1] James O. Coplien. *Multi-Paradigm Design for C++*. Addison-Wesley, 1999.

[2] Krzysztof Czarnecki and Michal Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In Robert Glück and Michael R. Lowry, editors, *Proc. of Generative Programming and Component Engineering, 4th International Conference, GPCE 2005*, LNCS 3676, pages 422–437, Tallinn, Estonia, October 2005. Springer.

[3] Krzysztof Czarnecki and Ulrich W. Eisenecker. *Generative Programing: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[4] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10:7–29, January/March 2005.

[5] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. Staged configuration through specialization and multi-level configuration of feature models. *Software Process: Improvement and Practice*, 10:143–169, April/June 2005.

[6] Krzysztof Czarnecki and Chang Hwan Peter Kim. Cardinality-based feature modeling and constraints: A progress report. In *International Workshop on Software Factories, OOPSLA 2005*, San Diego, USA, October 2005.

[7] Valentino Vranić. Reconciling feature modeling: A feature modeling metamodel. In Matias Weske and Peter Liggsmeyer, editors, *Proc. of 5th Annual International Conference on Object-Oriented and Internet-Based Technologies, Concepts, and Applications for a Networked World (Net.ObjectDays 2004)*, LNCS 3263, pages 122–137, Erfurt, Germany, September 2004. Springer.