Feature Modeling

A. Speck [†] E. Pulvermüller[◇]

[†]Wilhelm-Schickard-Institut für Informatik
Universität Tübingen, 72076 Tübingen, Germany
http://www-pu.informatik.uni-tuebingen.de/users/speck/

[◇]Fakultät für Informatik, IPD
Universität Karlsruhe, 76128 Karlsruhe, Germany
http://i44w3.info.uni-karlsruhe.de/~pulvermu/

1 Introduction

Component-driven approaches for software development are increasingly gaining importance. As with objects there exist different definitions for components, the most well-known is given in [3]. Components and their composition are supported by various technologies such as CORBA, COM/DCOM or Java Beans. Components are more than issue of research. Different commercial vendors provide components as building blocks (e.g. Enterprise Beans) for industrial systems.

Now the question rises how to build up (or model, design and verify) interactions between the composed components. We consider that a component provides one feature or a set of features. The design of a system of components is driven by the desired features and their interactions.

2 Features

Features are properties of components. A component may provide a single feature or a set of multiple features. A feature itself may consist of other features. Examples of features are: an algorithm performed or an attribute of a component. Features are either basic features (usually representing a single method or attribute) or combined features (c.f. section 3).

The features or respectively the names of the features have to be annotated to the objects, methods or attributes of the components. In the most simple case this has to be done manually by the developer of the component as documentation of the feature. Composed features are expressed by their sub-features.

3 Feature driven Composition

In the same way as components may be combined in new larger components features may be combined too.

Features may be combined from (sub-)features with logical operators. E.g. the feature *text editor* consists of several features:

- *editing algorithms* like *insert*, *undo*, *save* and *load*, ...
- *text attributes* e.g. text which itself may contain other attributes (or features) like *words*, *sentences* or *paragraphs*

The feature *text editor* may be expressed as:

$$text\ editor = editing\ algorithms \wedge text\ attributes$$

$$editing\ algorithms = insert \wedge undo \wedge save \wedge load$$

$$text\ attributes = words \wedge sentences \wedge paragraphs$$

Of course all basic logical operations are available to express feature combination.

The application of a logic description of feature combinations enables to control and verify systems. A system may be designed as logical dependencies of different features. Such a logic design can automatically be processed and may serve as a base for automatic combination tools (generators [2, 1]).

The design may be verified by adding rules, e.g. *Feature A* requires *Feature B*, or *Feature A* excludes *Feature C*. When a specific design has to be reused the appropriate rules may simply be reused. In order to improve reuse the rules may be stored in a database.

4 Future Work

We presented a method to describe the design of a system by the logical combination of features. In future we will concentrate on the automatic parsing of simple features of components.

Another interesting area is the feature composition guided by artificial intelligence. This is an improvement of the feature design data base which may help to reuse existing design partially or to detect critical combinations which do not necessarily lead to explicit errors (e.g. at compile time).

References

[1] K. Czarnecki und U. Eisenecker. *Generative Programming.* Addison-Wesley, New York, 2000.

[2] J. Smaragdakis und D. Batory. Implementing Layered Designs with Mixin Layers. In *Proceedings of ECOOP'98, European Conference on Object-Oriented Programming*, pages 550 – 570, Brussels, July 1998.

[3] C. Szyperski. *Component Software.* Addison-Wesley, ACM-Press, New York, 1997.