Source Code Composition with the Reuseware Composition Framework

Jendrik Johannes*
Technische Universität Dresden
Institut für Software- und Multimediatechnik
D-01062 Dresden, Germany
jendrik.johannes@tu-dresden.de

Abstract

The Reuseware Composition Framework is a tool-supported framework that aids developers of new composition techniques with integrating them into programming languages. In this tool-demo proposal we explain the usage and benefits of the framework by defining an extension of the Java language.

1. Introduction and Background

In software engineering complex systems are managed by splitting them into components using different techniques [4, 6, 1]. While the techniques of the object-oriented paradigm—object interaction and class inheritance—are the most prominent, they have proven to be insufficient for many structuring needs. Therefore, new techniques, like aspect-oriented programming [3], were developed.

When such new techniques are developed and discovered in research, they are often prototypically implemented as extensions of existing general-purpose languages like Java or Prolog. The execution is often performed by preprocessors rewriting the source code.

Such composition techniques are referred to as *gray-box* compositions. In contrast to compiled *black-box* components that are not altered at composition time, gray-box source code components are internally changed during composition. Yet, they provide a clear composition interface, which disinguishes them from *white-box* components which are arbitrarily modifiable and thus are easy to break.

Reuseware supports the extension of existing languages with component models for gray-box components [2, 5]. It is a Java framework with an Eclipse-based GUI that assists the user in defining such extensions and generating tooling for extended languages. Most importantly, it includes a generic pre-processesor which, extended by automatically generated language-specific tooling (like parsers), can execute compositions utilizing source code rewriting. Through this tool support, experimentation and prototyping for newly developed composition techniques becomes much easier.

*This research has been co-funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`), as well as through the 6th Framework Programme project Modelplex contract number 034081 (cf. `http://www.modelplex.org`).

2. Tool Demo Description

In this demonstration, we will explain the extension of Java with a gray-box component model, which has its origin in Invasive Software Composition [1]. This will show how composition systems for newly developed composition techniques can quickly be build on top of the Reuseware Composition Framework.

In the following, we refer to an original language (Java in this example) as the *core language*; the extended as *reuse language*. Components—which are fragments of source code—are called *fragment components* or simply *fragments*. Composition techiques are implemented in so-called *composers*.

Using Reuseware is a two step process: First, the language at hand has to be extended and tool support has to be generated. Second, the generated systems can be used to execute source code compositions. We first perform a simple extension of the language and then repeat the process to realize an aspect-oriented composition technique.

Languages are described by grammars (Figure 1). Reuseware supports the developer to extended a language with additional constructs for *variation point* declaration. Two types of variation points are supported: *Slots* are holes in a fragment that have to be filled by other fragments. *Hooks* mark positions, where additional fragments can be inserted.

Once the desired extension has been performed, the composition system can be generated without further expense.

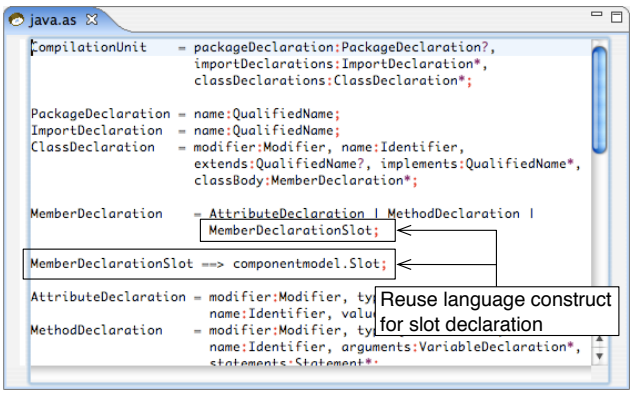

Figure 1. Extended grammar of Java

Figure 2. (a/b) Two fragments and (c) a composition program

When running the generated system, fragments can be written in the extended language (Figure 2a/b).

Compositions of those fragments are described in composition programs. These are written in Reuseware's Fragment Composition Language (FraCoLa). It is a generic composition language that can be used to define compositions of fragments written in arbitrary reuse languages. Beside ordinary functionality of scripting languages (e.g., conditions and loops) the language allows to call the primitive composers *Bind* and *Extend*. These are used to bind fragments to slots and extend hooks with fragments respectively. The composition program in Figure 2c binds the `Method` fragment from Figure 2b to the `Class` fragment from Figure 2a.

Another feature of the Reuseware Composition Framework is the possibility to define customized *complex composers* and tailored composition languages to call them. In the following we will shortly demonstrate how this can be utilized to realize a simple aspect-oriented weaving operator and a corresponding composition language.

A tailored composition language (here the weaving language) is defined by a grammar, in which we reuse constructs from FraCoLa. The weaving language also contains a new construct (`Weave`) for calling the weaving composer defined in Figure 3. Such complex composer definitions consist of successive calls to Bind and Extend and are also written in FraCoLa.

Note, that the defined weave composer is language independent. That is, it can perform cross-cutting source code composition of fragments written in an arbitrary reuse language. It could also be defined in a more Java specific way, making it less flexible but easier to use for the composition of Java fragments.

This short description only highlights the most prominent features of Reuseware. In the demonstration the explained features will be shown in more detail on the introduced, and also on other more advanced, examples.

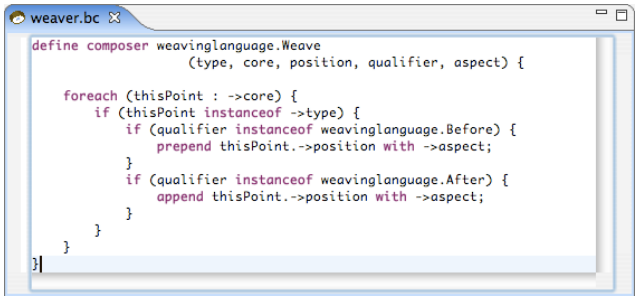

Figure 3. The Weave composer

References

[1] U. Aßmann. *Invasive Software Composition*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[2] J. Henriksson, J. Johannes, S. Zschaler, and U. Aßmann. Reuseware – adding modularity to your language of choice. *Proceedings of TOOLS EUROPE 2007: Special Issue of the Journal of Object Technology (to appear)*, 2007.

[3] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect-oriented programming. In *Proceedings of the European Conference on Object-Oriented Programming (ECOOP 97)*, volume 1241 of *LNCS*, pages 220–242. Springer, Heidelberg, 1997.

[4] D. McIlroy. Mass-produced software components. In *Proceedings of the 1st International Conference on Software Engineering, Garmisch Pattenkirchen, Germany*, pages 88–98, 1968.

[5] Software Technology Group of TU Dresden. Reusware webpage. `http://www.reuseware.org`, Apr. 2007.

[6] C. Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, New York, 1998.