PacoSuite and JAsCo: A Visual Component Composition Environment with Advanced Aspect Separation Features

Wim Vanderperren, Davy Suvée, Bart Wydaeghe, and Viviane Jonckers

Vrije Universiteit Brussel, Pleinlaan 2
1050 Brussel, Belgium
{wvdperre,dsuvee,bwydaegh,vejoncke}@vub.ac.be
http://ssel.vub.ac.be

Abstract. This paper presents the visual component composition environment called PacoSuite and the tools needed for the JAsCo aspect-oriented programming language. PacoSuite allows plug-and-play component composition without in-depth technical knowledge of the components. PacoSuite uses three constructs: components, composition patterns and composition adapters. A composition pattern is an abstract and reusable description of a collaboration between components. A composition adapter on the other hand, describes transformations of a composition of components and is used to modularize crosscutting concerns. A composition adapter is able to have an implementation in the JAsCo language in order to invasively alter components. Compatibility of a given collaboration is checked using finite automaton theory and the glue-code to make the composition work is generated automatically.

1 Introduction

Current practice visual component composition environments are still far from reaching the plug and play ideal promised by component based development [4]. Expert technical knowledge of components is required in order to be able to compose them. There's no support whatsoever to verify whether a given composition of components is able to work together. Glue-code still has to be written manually to achieve a more involved collaboration than a mere event-method connection. Moreover, most tools don't even support reusing such a simple collaboration. To solve these problems, we propose a novel visual component composition environment, called PacoSuite. PacoSuite lifts current practice component composition to a higher abstraction level. Composition patterns are introduced as reusable and abstract collaborations. Composition patterns as well as components are documented by a special kind of MSC [2]. PacoSuite automatically validates a given composition using finite automaton theory. In addition, glue-code which enables the collaboration is generated. Recently, composition adapters have been introduced to separate crosscutting concerns [3] that do not fit into our current constructs. Composition adapters describe transformations of a composition pattern independent of a specific API. In addition, a composition adapter is able to have an implementation in the JAsCo aspect-oriented implementation language. This enables a composition adapter to influence the interior behavior of components. We refer to [5,6,7] for more information on the fundamentals of this approach.

The next section describes the PacoSuite tool in more detail. Section 3 shortly sketches the tools required by the JAsCo aspect-oriented programming language.

2 PacoSuite

PacoSuite consists of two tools: a visual documentation editor called PacoDoc and the actual component composition environment called PacoWire. Both tools are written in the Java language. PacoDoc allows the user to construct usage scenarios, composition patterns and composition adapters in a user-friendly manner. Afterwards, the drawn diagrams are exported to an XML file. PacoDoc is also integrated in PacoWire, such that a component composer is able to view the documentation of a component at any time. Fig. 1 shows a screenshot of PacoDoc.

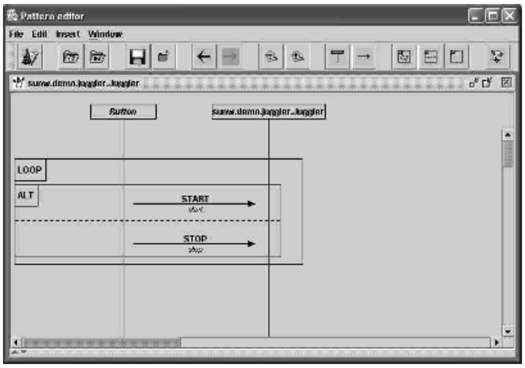

Fig. 1. The documentation of the Juggler component shown in PacoDoc

PacoWire is our actual component composition tool and contains a set of components, composition patterns and composition adapters nicely sorted into different categories. Creating an application is as simple as visually dragging components onto a composition pattern. The drag is refused when a component is detected to be incompatible with the selected composition pattern. Fig. 2 illustrates a screenshot of PacoWire where the component composer is about to drag the juggler component onto the subject role of the ToggleControl composition pattern. The ToggleControl composition pattern specifies a toggling behavior (consecutive starts and stops). The subject role receives the commands and the control role is responsible for sending the toggle commands. As the Juggler component is able to receive consecutive start and stop commands (see Fig. 1), it can fulfill the subject role of the ToggleControl composition pattern. However, when the component composer would drag the Juggler component onto the control role, the drag would be refused because the Juggler component can only receive messages. The JButton component from the Java Swing library for instance, is compatible with the control role. After the JButton is dragged onto the control role, glue-code that implements this collaboration can be generated. The resulting application allows the juggler to be toggled from a single button. Notice that it is impossible to visually wire even this simple collaboration in current component composition environments because state information is required.

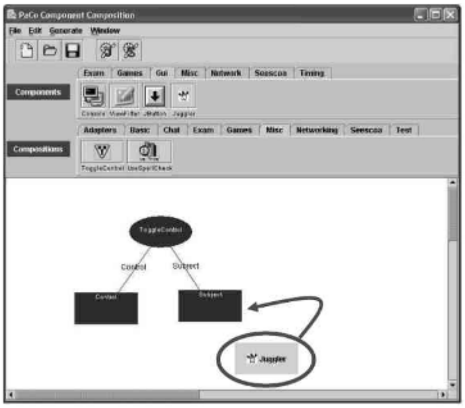
PaCo Component Composition
File Edit Generate Window
Components
Exam Games Gui Misc Network Seescoa Timing
Compositions
Adapters Basic Chat Exam Games Misc Networking Seescoa Test
ToggleControl UseSpellCheck
ToggleControl
Control
Subject
Juggler

Fig. 2. Screenshot of PacoWire. The component composer is about to drag the Juggler component onto the subject role of the ToggleControl composition pattern

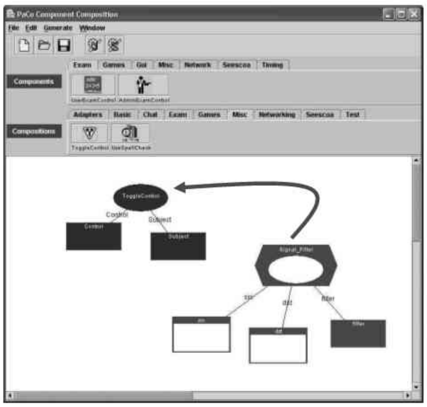
PaCo Component Composition
File Edit Generate Window
Components
Exam Games Gui Misc Network Seescoa Timing
Compositions
Adapters Basic Chat Exam Games Misc Networking Seescoa Test
ToggleControl UseSpellCheck
ToggleControl
Control
Subject
Signal_Filter
src
dst
filter

Fig. 3. Applying the invasiveTimer composition adapter onto the ToggleControl composition pattern

Applying a composition adapter is also achieved by a simple drag and drop. The tool takes care of inserting the transformations the composition adapter describes. When the composition adapter is implemented using JAsCo, the JAsCo tools are executed transparently for the user. In fact, a component composer doesn't even have to know whether a composition adapter has a JAsCo implementation or not. Fig. 3 illustrates a screenshot of PacoWire, where a component composer is about to map the SignalFilter composition adapter onto the ToggleControl composition pattern. The SignalFilter composition adapter describes a logging aspect. The communication between the source and destination roles is trapped and re-routed through the filter role. In this way, the filter role is able to log the events the component composer is interested in. To apply the SignalFilter composition adapter onto the ToggleControl composition pattern, the component composer can simply drag one onto the other. The source and destination roles of the SignalFilter composition adapter are then automatically mapped onto roles of the composition pattern using an algorithm based on dynamic programming ideas [1]. The tool issues a warning when the application of this composition adapter onto the selected composition pattern is not valid. Afterwards, the JButton and Juggler components are mapped onto the roles of the ToggleControl composition pattern as before. A logging component that writes received events onto disk can for instance be mapped onto the filter role. When the glue-code is generated, the juggling application works just as it did before. However, every signal from the button is first rerouted through the logging component before it is sent to the Juggler component.

3 JAsCo

The JAsCo-language has been implemented to allow composition adapters to affect the internal behavior of components. The JAsCo-framework provides 4 tools which are required to deploy aspects on components.

The key tool of the JAsCo-package is the *BeanTransformer*. To enable interaction between aspects and components, we propose a new component model where each public method of a component is provided with a *trap*. These traps reroute control-flow at run-time, which enables the execution of aspect behavior. The *BeanTransformer*-tool is responsible for transforming a regular Java Bean into a JAsCo bean component. This tool employs state-of-the-art Java byte code adaptation techniques for inserting *traps* at the appropriate places.

The JAsCo-language itself stays as close as possible to the regular Java syntax and introduces two concepts: aspect beans and connectors. Aspect beans are used for describing crosscutting behavior. Deploying an aspect bean within an application is done by making use of connectors. The definition of both aspect beans and connectors is preprocessed to a Java source code file. Afterwards, this definition is compiled to its Java class-representation by making use of the standard Java Compiler. Both the *CompileAspect*- and *CompileConnector*-tool are responsible for managing this compilation-process.

The fourth tool contained within the JAsCo-package is the *Introspector*-tool, which is a GUI environment that allows introspecting what connectors are loaded. Connectors can be added and removed at run-time, which enables to dynamically change the properties of the system. The tool displays the various hooks that are instantiated by the connectors and the targets on which these hooks are applied.

References

[1] Bellman R.E. & Dreyfus S.E. Applied Dynamical Programming. Princeton University Press, 1962.

[2] ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart (MSC). ITU-TS, Geneva, September 1993.

[3] Kiczales, G., Lamping, J., Lopes, C.V., Maeda, C., Mendhekar, A. and Murphy, A. Aspect-Oriented Programming. In proceedings of the 19th International Conference on Software Engineering (ICSE), Boston, USA. ACM Press. May 1997.

[4] Short, K. (1997). Component Based Development and Object Modeling. Available at: `http://www.cool.sterling.com/cdb/whitepaper/2.htm`

[5] Suvée, D., Vanderperren, W., and Jonckers, V. JAsCo: an Aspect-Oriented approach tailored for CBSD. In Proc. of AOSD int. Conf., Boston, USA, march 2003.

[6] Vanderperren, W. Localizing crosscutting concerns in visual component based development. In proc. of SERP international conference, Las Vegas, USA, june 2002.

[7] Wydaeghe, B. and Vandeperren, W. Visual Component Composition Using Composition Patterns. In Proceedings of Tools 2001, July 2001.