An Introduction to Context-Oriented Programming with ContextS

Robert Hirschfeld[1], Pascal Costanza[2], and Michael Haupt[1]

[1] Hasso-Plattner-Institut, Universität Potsdam, D-14482 Potsdam, Germany
{robert.hirschfeld,michael.haupt}@hpi.uni-potsdam.de
[2] Programming Technology Lab, Vrije Universiteit Brussel, B-1050 Brussels, Belgium
pascal.costanza@vub.ac.be

Abstract. Context-oriented Programming, or COP, provides programmers with dedicated abstractions and mechanisms to concisely represent behavioral variations that depend on execution context. By treating context explicitly, and by directly supporting dynamic composition, COP allows programmers to better express software entities that adapt their behavior late-bound at run-time. Our paper illustrates COP constructs, their application, and their implementation by developing a sample scenario, using ContextS in the Squeak/Smalltalk programming environment.

1 Introduction

Every intrinsically complex application exhibits behavior that depends on its context of use. Here, the meaning of context is broad and can range from obvious concepts such as location, time of day, or temperature over more technical properties like connectivity, bandwidth, battery level, or energy consumption to a user's subscriptions, preferences, or personalization in general.

Besides these examples of context that are often associated with the domain of ambient computing, the computational context of the program itself, for example its control flow or the sets or versions of libraries used, can be an important source of information for affecting the behavior of parts of the system.

Even though context is a central notion in a wide range of application domains, there is no direct support of context-dependent behavior from traditional programming languages and environments. Here, the expression of variations requires developers to repeatedly state conditional dependencies, resulting in scattered and tangled code.

This phenomenon, also known as crosscutting concerns, and some of the associated problems were documented by the aspect-oriented programming (AOP [16]) and the feature-oriented programming (FOP [2]) communities. The focus of AOP is mainly on the establishments of inverse one-to-many relationships [17] to achieve their vision of quantification and obliviousness [10]. FOP's main concern is the compile-time selection and combination of variations, and the necessary algebraic means to reason about such layer compositions [3].

R. Lämmel, J. Visser, and J. Saraiva (Eds.): GTTSE 2007, LNCS 5235, pp. 396–407, 2008.

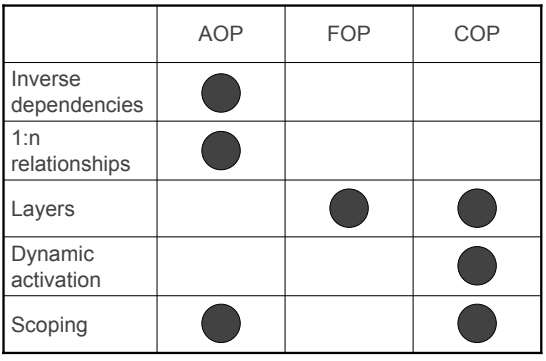

Fig. 1. Properties of AOP, FOP, and COP

Context-oriented programming (COP [6,14]) addresses the problem of dynamically composing context-dependent concerns, which are potentially crosscutting. COP takes the notion of FOP layers and provides means for their selection and composition at run-time. While FOP mechanisms are applied at compile-time—with the effect that, during program execution, layers as a distinct entity are no longer available—, COP preserves layers, adds the notion of dynamic layer activation and deactivation, and provides dynamic scoping to delimit the visibility of their composition as needed (Figure 1). With the dynamic scoping mechanisms offered by COP implementations, layered code can be associated with the units it belongs to and can be composed into or removed from the system depending on its context of use.

There are several COP extensions to popular programming languages such as ContextL for Lisp [6], ContextS for Squeak/Smalltalk [14], ContextR for Ruby, ContextPy for Python, and ContextJ* for Java [14]. Here, we will focus on ContextS. Our paper is meant to be used mainly as a tutorial, describing ContextS in how it can be applied to the implementation of context-dependent behavioral variations.

The remainder of our paper is organized as follows: We give an overview of COP in Section 2. In Section 3 we introduce some of the COP extensions provided with ContextS which are applied to an example presented in Section 4. After some recommendations for further reading in Section 5 we conclude our paper with Section 6.

2 Context-Oriented Programming

COP, as introduced in [6,14], facilitates the modularization of context-dependent behavioral variations. It provides dedicated programming abstractions and mechanisms to better express software entities that need to change their behavior depending on their context of use.