Available online at www.sciencedirect.com
SCIENCE DIRECT®

Electronic Notes in Theoretical Computer Science

ELSEVIER Electronic Notes in Theoretical Computer Science 114 (2005) 153–174
www.elsevier.com/locate/entcs

# There are no Aspects

Davy Suvée[1] , Wim Vanderperren[2] , Dennis Wagelaar and Viviane Jonckers

*System and Software Engineering Lab, Vrije Universiteit Brussel, Pleinlaan 2, 1050 Brussels, Belgium*

{*dsuvee, wvdperre, dwagelaa, vejoncke*}*@vub.ac.be, http://ssel.vub.ac.be*

**Abstract**

In this paper, we claim that a specialized aspect module is not required. Instead, we propose an expressive aspect-oriented composition mechanism which can be applied upon existing modules. At the design level, the CoCompose modeling framework is introduced which is based on Model Driven Development. CoCompose allows step-wise refinement from a high-level design to the lowest level design or code level. Using these refinements, CoCompose postpones the decision concerning the modularization construct that is chosen for a particular concern. At the lowest level design however, a specialized aspect modularization construct still needs to be chosen because current aspect-oriented technologies typically introduce an aspect module. For resolving this issue, the FuseJ programming language is proposed that allows implementing all possible concerns as regular components. FuseJ introduces an expressive component composition mechanism that supports both regular and aspect-oriented compositions between components. As such, a seamless transition from design to implementation is achieved because no specialized aspect modules exist both at the design and implementation level.

*Keywords:* Component-Based Software Development, Aspect-Oriented Software Development, Composition Mechanisms, Model-Driven Development.

[1] The author is supported by a doctoral scholarship from the Flemish Institute for the Improvement of the Scientifical-Technological Research in the Industry (IWT or in Flemish: "Vlaams instituut voor de bevordering van het wetenschappelijk-technologisch onderzoek in de industrie")
[2] The author is supported by a doctoral scholarship from the Fund for Scientific Research (FWO or in Flemish: "Fonds voor Wetenschappelijk Onderzoek")

1571-0661/$ – see front matter © 2005 Published by Elsevier B.V.
doi:10.1016/j.entcs.2004.12.012

## 1 Introduction

Aspect-Oriented Software Development (AOSD) is a new development paradigm that aims at achieving a better separation of concerns than possible using standard object-oriented (OO) software engineering methodologies [15]. AOSD claims that some concerns of an application cannot be cleanly modularized with standard OO technologies as they are scattered over or tangled with the different modules of the system. Such a concern is called *crosscutting* because the concern virtually crosscuts the decomposition of the system. As a result, similar logic is repeated in different modules, with code duplication as a consequence. Due to this code duplication, it becomes very hard to add, edit or remove a crosscutting concern from the system. Typical examples of crosscutting concerns are debugging concerns such as logging [15] and contract verification [30], security concerns [31] such as confidentiality and access control and business rules [6] that describe business-specific logic.

Component-Based Software Engineering (CBSE) is another software engineering paradigm that aims at increasing reusability of individual components and component compositions. CBSE advocates very low coupling between components and high cohesion of single components. Furthermore, components are black-box[3] entities, which are independently deployable [29] . In fact, when CBSE is employed, a component should never explicitly rely onto other specific components in order to increase reusability. As a consequence, CBSE suffers greatly from crosscutting concerns and tangled code because a lot of concerns are spread over and repeated among different components in order to keep the coupling as low as possible. As a result, aspect-oriented ideas are very welcome in the component-based world.

Currently, a wealth of technologies are available that integrate aspect-oriented ideas into component-based software engineering. Examples are JAC [21], JBoss/AOP [13], EAOP [9], OIF [11] and JAsCo [28]. All of these approaches focus at introducing new programming languages or frameworks in order to modularize crosscutting concerns. Support for aspect-oriented ideas during the early cycles of component-based software engineering is still not yet fully explored. Even though several production-quality aspect-oriented technologies exist, it seems to be very difficult to recuperate aspect-oriented ideas in for example the design process. Currently, when designing a software application with aspects in mind, the crucial question is: “when to model concerns as an aspect?”. Indeed, one has to decide which concerns of the application

[3] There is currently no unanimous vision on CBSE. For example, several different approaches exist that motivate why black-box, grey-box or white-box component composition is the better choice. In this paper, we assume the Szyperski vision [29] on CBSE with black-box component composition.