Chapter X

Temporal OCL: Meeting Specification Demands for Business Components

Stefan Conrad
Ludwig-Maximilians-University Munich, Germany

Klaus Turowski
Otto-von-Guericke-University Magdeburg, Germany

INTRODUCTION

Compositional plug-and-play-like reuse of black box components requires sophisticated techniques to specify components, especially if we combine third-party components, which are traded on component markets, to customer-individual business application systems. As in established engineering disciplines like mechanical engineering or electrical engineering, we need a formal documentation of business components that becomes part of contractual agreements. Taking this problem as a starting point, we explain the general layered structure of software contracts for business components and show shortcomings of common specification approaches. Furthermore, we introduce a formal notation for the specification of business components that extends the Object Constraint Language (OCL) and that allows a broader use of the Unified Modeling Language (UML) with respect to the layered structure of software contracts for business components.

The remainder of the chapter is as follows. After providing background information in the next section, we discuss the necessity of a multi-level notation standard. Thereafter, we explain how the OCL can be used to specify business components. Taking this as a basis, we proceed to the main thrust of our chapter the temporal extension of OCL. Finally, we present our conclusions and give an outlook.

SOFTWARE CONTRACTS FOR BUSINESS COMPONENTS

Combining off-the-shelf software components offered by different vendors to customer-individual business application systems is a goal that is followed-up for a long time.
By achieving this goal, advantages of individually programmed software with those of standardized off-the-shelf software could come together. In this context, we need compositional reuse techniques. Compositional reuse is a special reuse technique as generative techniques or code and design scavenging (Sametinger, 1997, pp. 25-28). The emphasis on compositional reuse stems from our guiding model, which is the compositional plug-and-play-like reuse of black box components that are traded on a component market. In general, a guiding model is an ideal future state that might not completely be reached.

Corresponding to our guiding model, a company, which, e.g., needs new software for stock keeping, could buy a suitable software component on the component market and further integrate it into its business application system with little effort. Brown and Wallnau (1996, pp. 11-14) explain the steps that are generally necessary to do so, e.g., technical and semantic adaptation or composition. Expected improvements, which should come along with using software components, concern cost efficiency, quality, productivity, market penetration, market share, performance, interoperability, reliability or software complexity (cf. e.g., Orfali, Harkey, & Edwards, 1996, S. 29-32).

According to Fellner and Turowski (2000, pp. 3-4), we understand by a component a reusable, self-contained and marketable software part, which provides services through a well-defined interface, and which can be deployed in configurations unknown at development time. A business component is a component that implements a certain set of services out of a given business domain. Refer to Szyperski (1998, pp. 164-168) and Fellner and Turowski (2000) for an in depth discussion of various other component approaches given in literature.

To use business components according to our guiding model, it is necessary to standardize them (for a detailed discussion on standardizing business components cf. Turowski, 2000). Furthermore, we have to describe their interface and behavior in a consistent and unequivocal way. In short, we have to specify them. Specification becomes more and more important with respect to third-party composition of business components, since the specification might be the only available support for a composer who combines business components from different vendors to an application system.

Software contracts offer a good solution to meet the special requirements of specifying business components. Software contracts go back to Meyer, who introduced contracts as a concept in the Eiffel programming language. He called it programming by contract (Meyer, 1988) and extended it later to the concept of design by contract (Meyer, 1992). Furthermore, similar concepts are described in Wilkerson and Wirfs-Brock (1989) or Johnson ad Wirfs-Brock (1990).

Software contracts are obligations to which a service donator (e.g., a business component) and a service client agree.
Augmenting UML to Support the Design and Evolution of User Interfaces
www.igi-global.com/chapter/augmenting-uml-support-design-evolution/4939?camid=4v1a