Resolving Architectural Mismatches of COTS Through Architectural Reconciliation

Paris Avgeriou and Nicolas Guelfi

Software Engineering Competence Center (SE2C), University of Luxembourg,
6, rue Richard Coudenhove-Kalergi L-1359 Luxembourg-Kirchberg, Luxembourg
{paris.avgeriou, nicolas.guelfi}@uni.lu

Abstract. The integration of COTS components into a system under development entails architectural mismatches. These have been tackled, so far, at the component level, through component adaptation techniques, but they also must be tackled at an architectural level of abstraction. In this paper we propose an approach for resolving architectural mismatches, with the aid of architectural reconciliation. The approach consists of designing and subsequently reconciling two architectural models, one that is forward-engineered from the requirements and another that is reverse-engineered from the COTS-based implementation. The final reconciled model is optimally adapted both to the requirements and to the actual COTS-based implementation. The contribution of this paper lies in the application of architectural reconciliation in the context of COTS-based software development. Architectural modeling is based upon the UML 2.0 standard, while the reconciliation is performed by transforming the two models, with the help of architectural design decisions.

1 Introduction

The inevitable problem with reusing COTS components is that they simply don't correspond perfectly to the requirements specification and consequently to the envisioned architecture of the system [1]. Even when COTS-based systems are designed by taking into consideration pre-existing components from the market that roughly correspond to the requirements, eventually there will still be disparities when the COTS are integrated. One of the major causes of this problem is **architectural mismatches**: differences between a COTS component and the software system, where it will be integrated, which occur when the former makes the wrong assumptions about the latter [1, 8]. For example, a commercial component can falsely assume that it is in charge of controlling the sequence of interactions between itself and other components, or that other components should comply with specific protocols of interactions. To make matters worse, such assumptions are implicit and are usually in conflict with each other. The consequences are that system-wide properties are diverged from the requirements, both functional and quality ones. Especially quality requirements such as performance, reliability, and flexibility that depend profoundly on the architecture [4, 5, 24] may be to a large extent distressed by the use of COTS components.

X. Franch and D. Port (Eds.): ICCBSS 2005, LNCS 3412, pp. 248 – 257, 2005.

The research community has attempted to tackle the problem of architectural mismatches, focusing on the component level, by means of **component adaptation** techniques, which attempt to incorporate unintended changes in a component for use in a particular application [3]. These techniques are distinguished into **white-box** (e.g. inheritance) and **black-box** (e.g. wrapping), depending on whether the component itself is adapted or whether its interface is adapted [4]. In the case of COTS components, black-box techniques are usually applied since the component's source code is usually prohibited from being inspected or modified [1]. There are several techniques proposed so far [3, 12, 15, 16, 29], and they can be applied according to the context of use and the possible benefits and liabilities they entail [12].

However architectural mismatches cannot only be resolved at the component level since they do not concern an isolated component but they affect a greater part of the system, which collectively includes a number of components and connectors [8, 25]. Architectural mismatches caused by a single component may influence not only the components that communicate with it but may also be propagated further on to other components. Therefore such mismatches may require not only the adaptation of the COTS component but also the modification, addition or removal of other architectural elements. In order to perform these changes we need to examine a greater part of the system's architecture, identify those elements that are affected and subsequently decide on how exactly the architecture should be modified. We thus need to tackle the problem of architectural mismatches from an architectural perspective [8].

This paper proposes an approach to resolve architectural mismatches, caused by integrating COTS, using the technique of **architectural reconciliation**. In specific, it suggests the design and subsequently the reconciliation of two architectural models: one that is forward engineered from the requirements specification and a second that is reverse-engineered from the COTS-based system implementation. The former expresses the architectural decisions in an *ideal* system, which conforms to the requirements. The latter not only grasps the implementation constraints, but also explicitly specifies the architectural impact of COTS that were incorporated in the implementation, making their design assumptions explicit, with respect to the rest of the application. These two models are reconciled into a third model that will combine the two perspectives in the best possible tradeoff, by taking under consideration the design assumptions of the COTS components, but also addressing the requirements, to the best possible extent. The reconciliation is performed by transforming the two models, based on architectural design decisions, depending on which side, requirements or implementation should be more supported. The reconciled model can eventually be used to re-engineer the COTS-based system and also update the requirements. Architectural modeling is based upon the UML 2.0 standard.

The rest of the paper is organized as follows: section 2 provides the details of the proposed approach for resolving architectural mismatches through architectural reconciliation. Section 3 illustrates the implementation of the approach through a case study while Section 4 presents some related research work with respect to architectural reconciliation. Finally Section 5 wraps up with conclusions and future work.