Adaptor synthesis for protocol-enhanced component based architectures

Massimo Tivoli
University of L'Aquila
via Vetoio 1, 67100 L'Aquila
tivoli@di.univaq.it

David Garlan
Carnegie Mellon University
5000 Forbes Avenue, Pittsburgh, PA 15213-3891
garlan@cs.cmu.edu

Abstract

Correct assembly of software components is an important issue in Component Based Software Engineering. Composing a system from reusable components often introduces a set of problems related to communication and compatibility. In particular, one of the main problems in component assembly is that components may have incompatible interaction behavior. In this paper, we address this problem using an architecture-based approach that can detect integration mismatches, and semi-automatically synthesize a suitable adaptor, or glue code, to bridge them.

1 Introduction

Correct automatic assembly of software components is an important issue in Component Based Software Engineering. Building a system from reusable or from *Commercial-Off-The-Shelf* (COTS) components introduces a set of potential problems related to communication and compatibility. One of the main problems in component assembly is that components may have incompatible interaction behavior. This, in turn, can affect the correct functioning of the composed system. There are two common approaches dealing with this problem. One is to disallow the composition of incompatible components, or possibly restrict their behaviors to a behavioral subset that *is* compatible [1]. The second approach is to modify one or more of the components to remove the incompatibilities. Neither is ideal. The former limits the ability to combine components or achieve the maximum benefit from their combination. The latter, is often not possible with COTS components. In this paper we present an approach that addresses the goal of automatic synthesis of interaction mechanisms. These mechanisms are synthesized to bridge component incompatibilities and to enhance the components interaction protocol in order to introduce more sophisticated interactions among the components. Starting from earlier work in synthesizing failure-free (e.g., deadlock-free) adaptors for component composition by restricting their behaviors [1], we show here how to augment that technique with protocol-enhancing extensions similar to the protocol transformers introduced in [2]. The key idea is to encapsulate protocol enhancements as policies that can be applied automatically, and incrementally to synthesize new behaviors that eliminate sources of incompatibility and add new component interactions. We implemented the approach in our *SYNTHESIS* tool [3] (*http://www.di.univaq.it/tivoli/SYNTHESIS/synthesis.html*). This paper is a short version of an existent technical report [4] where we report and compare related work, and formalize the approach giving also a formal proof of its correctness. Moreover, in [4], we validate the approach by means of an industrial case-study. The paper is organized as follows: Section 2 describes the formal architectural model underlining our approach. Section 3 discusses our adaptor synthesis approach for protocol-enhanced component based architectures. Section 4 summarizes the contributions of the paper and outlines future work.

2 The reference architectural style

The starting point for our work is the use of a formal architectural model of the system representing the components to be integrated and the connectors over which the components will communicate. To simplify matters we will consider the special case of a generic layered architecture in which components can request services of components above them, and notify components below them. Specifically, we assume each component has a top and bottom interface. The top (bottom) interface of a component is a set of top (bottom) ports. Connectors between components are synchronous communication channels defining top and bottom ports. Components communicate by passing two types of messages: notifications and requests. A notification is sent downward, while a request is sent upward. We will also distinguish between two kinds of components (i) *functional components* and (ii) *coordinators*. Functional components implement the system functionality. Coordinators, on the other hand, simply route messages and each input they receive is strictly followed by a corresponding output. We make this distinction in order to clearly separate components that are responsible for the functional behavior of a system and components that are introduced to aid the integration/communication behavior. Within this architectural style, we will refer to a system as a *Coordinator-Free Architecture* (CFA) if it is defined without any coordinators. Conversely, a system in which coordinators appear is termed a *Coordinator-Based Architecture* (CBA) and is defined as *a set of functional components directly connected to one or more coordinators, through connectors, in a synchronous way*.

3 Adaptor synthesis for protocol enhancement

We recall that we want to automatically synthesize interaction mechanisms to bridge component incompatibilities and add more sophisticated interactions among the components. This is done by modifying and augmenting the failure-free coordinator synthesis approach presented in [1] with protocol-enhancing extensions. This extension starts with a deadlock-free CBA system S that exhibits only a set P of desired behaviors (S is the output of the old approach) and produces the corresponding protocol-enhanced CBA system S'.

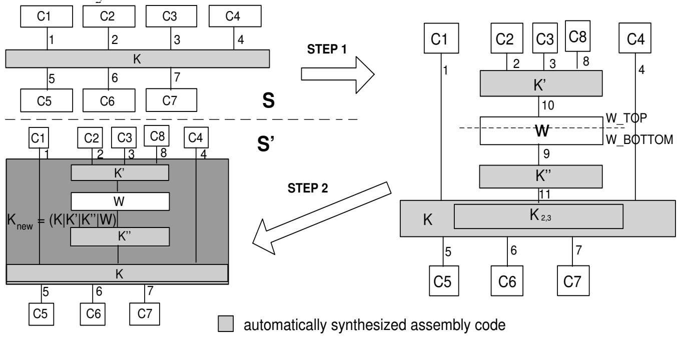

Figure 1. Adaptor synthesis for protocol enhancement

Our method assumes as input: i) S in terms of the *Labeled Transitions Systems* (LTSs) of each component and deadlock-free coordinator forming it, and of each component interaction specified by P; ii) a *basic Message Sequence Chart* (bMSC) and *High-level MSC* (HMSC) specification of the set E of protocol-enhancements that must be applied. Indeed, by combining our method with the old approach, the user does not provide the input (i), that is, the user does not have to provide the LTS of the deadlock-free coordinator[1]. In fact, by referring to [1], he/she gives as input to the *SYNTHESIS* tool only a MSC specification of the interaction behavior of the functional components forming S and a LTS specification of the desired interactions in P. From these specifications and by performing the approach presented in [1], *SYNTHESIS* automatically derives the input (i) required by the extension that we discuss in this paper. In the following, we discuss our method proceeding in two steps as illustrated in Figure 1. In the first step, by starting from the specification of P and S, if it is possible, we apply each protocol enhancement in E. By referring to Figure 1, this is done by inserting a wrapper component W between K and the portion of S concerned with a specified protocol enhancement (i.e., $C2$ and $C3$). Depending on the purposes of the enhancement, some wrapper might require to use an extra component to accomplish its tasks (i.e., $C8$). We first decouple K, $C2$ and $C3$ to assure that they no longer synchronize directly. Then we automatically derive a behavioral model of W and $C8$ (i.e., a LTS) from the MSC specification of E. Finally, if the insertion of W in S allows the resulting composed system (i.e., S' after the execution of the second step) to still satisfy each desired behavior in P, W is interposed between K, C_2 and C_3; and $C8$ is assembled with them. To insert W, we automatically synthesize two new coordinators K' and K''. In the second step, we derive the implementation of the synthesized glue code used to insert W in S and add possible extra components. By iterating the whole approach, K_{new} (see Figure 1) may be treated as the initial coordinator K with respect to the enforcing of new desired behaviors and the application of new enhancements. This allows us to be compositional in the automatic synthesis of the enhanced glue code.

[1]Providing this LTS would be impossible for real-scale systems.

4 Conclusion and future work

In this paper, we combined the approaches of protocol transformation formalization [2] and of automatic coordinator synthesis [1] to produce a new technique for automatically synthesizing protocol-enhanced coordinators for component-based systems. The two approaches take advantage of each other: while the approach of protocol transformations formalization adds compose-ability to the automatic coordinator synthesis approach, the latter adds automation to the former. The key results are: (i) our approach is compositional in the automatic synthesis of the enhanced coordinator; (ii) this, in turn, allows us to enhance a coordinator with respect to a useful set of protocol transformations such as the set of transformations described in [2]. The automation and applicability of our approach is supported by our tool called *SYNTHESIS*. As future work, we plan to: (a) support a specification of the protocol-enhancement policies which is more used (than bMSCs and HMSCs) in the practice of software development (e.g., UML2 Interaction Overview Diagrams and Sequence Diagrams); (b) validate the applicability of the approach to large-scale examples.

References

[1] P. Inverardi and M. Tivoli. *Software Architecture for Correct Components Assembly - LNCS 2804*. Springer, 2003.

[2] B. Spitznagel and D. Garlan. A compositional formalization of connector wrappers. In *proceeding of ICSE'03*.

[3] M. Tivoli and M. Autili. Synthesis: a tool for synthesizing correct and protocol-enhanced adaptors. *To appear on L'Object journal*.

[4] M. Tivoli and D. Garlan. Adaptor synthesis for protocol-enhanced component based architectures. Technical report, Dep. of Computer Science, Carnegie Mellon University - http://www.di.univaq.it/tivoli/trcs_08.pdf, 2005.