An Approach to Checking Compatibility of Service Contracts in Service-Oriented Applications

Surya Nepal, CSIRO ICT Centre, Australia
John Zic, CSIRO ICT Centre, Australia
Thi Chau, CSIRO ICT Centre, Australia

ABSTRACT

In the service-oriented architecture model, a service may be characterized by its exchange of asynchronous messages, and a service contract is a desirable composition of a variety of messages. Though this model is simple, implementing large-scale, cross-organizational distributed applications may be difficult to achieve in general, as there is no guarantee that service composition will be possible because of incompatibilities between the service contracts. This article examines and addresses this problem, first by identifying and specifying contract compatibility conditions based on standards such as WS-BusinessActivity, and second, through the use of a compatibility checking tool that enables application developers to perform checks at design time. [Article copies are available for purchase from InfoSci-on-Demand.com]

Keywords: Compatibility of Web Services; Service Contract; Service-Oriented Architecture; Web Services Transaction

INTRODUCTION

The service-oriented architecture (SOA) model is being promoted by the IT industry for use in the development of the next generation of Internet-scale distributed applications. These applications are comprised of a collection of independent, abstract services that are provided by business partners and third party service providers. Though the model is simple, the Web services vision of large-scale, cross-organizational distributed applications may be difficult to achieve due to the incompatibility of service contracts. Building such applications using the SOA model is getting easier with the good support from Web services standards (BPEL4WS, 2002; SSDL, 2005; WS-BA, 2005) and tools. Because of the availability of
these standards and tools, architects are able to specify service-oriented applications, as well as allow developers the capacity to build service-oriented applications. Nonetheless, as with any distributed systems, issues of deadlocks, race conditions, failures and exceptions, concurrency, and asynchrony remain and need careful attention. Previously, similar issues were faced by the distributed transaction community with partial resolutions being based on the use of ACID-based technologies (Elmagarmid, 1992; Gray & Reuter, 1992). However, as these solutions are for tightly coupled, short-lived applications, they are inappropriate in the autonomous, loosely coupled, cross-trust-domain SOA environment.

Ensuring the design time compatibility between participating service contracts represents a significant difficulty. For example, the merchant and customer services in an e-procurement example are said to be compatible if they can work with each other, and that there are no undesirable or unexpected interactions. The system needs to handle everything from the ideal case, where everything goes smoothly, to service level exceptions (e.g., customer requests for cancellation of the ordering process, late payment, or where there is a delay in the delivery of the goods ordered and paid for).

Analysis of service compatibility issues is highly amenable to model checking and the judicious application of formal methods. A body of work has been done on applying formal methods and model checking in compatibility of service-oriented applications. Compatibility issues are studied in two broad categories: (a) between a service contract and its implementation (Nakajima, 2002) and (b) between contracts of different services (Bultan, Fu, Hall, & Su, 2003; Foster, Uchitel, Kramer, & Magee, 2004; Greenfield, Kuo, Nepal, & Fekete, 2005; Nararaynan & Mellraith, 2002). The latter is the focus of this article.

Previous work (Greenfield et al., 2005) in service contract compatibility defined three compatibility criteria: (a) completeness: services handle all messages; (b) termination: services are always able to terminate in an acceptable consistent state despite failures, races, and other such exceptional events. This article examines these issues further and presents our analysis using an e-procurement example. The completeness criteria can be checked easily using message queues in model checkers (Greenfield et al., 2005). The focus for the rest of the article will be on the latter two criteria, that is, termination and acceptable consistent state.

First, we use WS-BusinessActivity (WS-BA) (WS-BA, 2005) to describe termination patterns in service contracts. Our approach enables application programmers to specify termination patterns and check the termination criterion at design time. Fundamentally, we note that services must terminate in a globally acceptable consistent state. For example, in the e-procurement application, an acceptable consistent state is when both the merchant and customer are satisfied; the merchant receives the payment and delivers the goods, and the customer pays for the invoice and receives the goods. One of the ways of specifying these states is to define global constraints over application states of each of the participating autonomous services. However, this is not possible as application states of an autonomous service are, by definition, internal to each service and not accessible by others. To overcome this, we propose a way of stating acceptable states in contracts using only message-related states. Our approach is based on the realistic assumptions that significant states are reflected in the sequences of messages exchanged, and consequently, global acceptable states can be defined by the conjunction of local acceptable states that can be checked at design time. Finally, we demonstrate the feasibility of our approach by developing a tool. The important aspect of our tool is that it not only detects the compatibility errors as described by Bultan et al. (2003), Foster et al. (2004), and Nararaynan and Mellraith (2002), but also classifies errors and provides reasoning for them. This helps developers to understand the reasons behind the incompatibility, and correct them at design time.
Flexible Probabilistic QoS Management of Orchestrations
[www.igi-global.com/chapter/flexible-probabilistic-qos-management-orchestrations/66960?camid=4v1a](www.igi-global.com/chapter/flexible-probabilistic-qos-management-orchestrations/66960?camid=4v1a)

Adaptive BPEL Service Compositions via Variability Management: A Methodology and Supporting Platform
[www.igi-global.com/article/adaptive-bpel-service-compositions-via-variability-management/220389?camid=4v1a](www.igi-global.com/article/adaptive-bpel-service-compositions-via-variability-management/220389?camid=4v1a)