WEB SERVICE INTERACTIONS: ANALYSIS AND DESIGN

Jianwen Su
Tevfik Bultan
Department of Computer Science
University of California at Santa Barbara

Xiang Fu
School of Computer & Information Sciences
Georgia Southwestern State University

Abstract The conversation model captures interactions among the components of a composite web service. A conversation is the global sequence of messages exchanged among the components. We discuss the impact of asynchronous communication on the conversation behavior. It turns out that the conversation behavior is significantly different for synchronous and asynchronous communication and demands new techniques for static analysis of composite web services. We outline strategies of model checking service composition for both bottom-up and top-down design approaches.

Keywords: Process interactions, conversations

Introduction

The success of business-to-consumer applications (e.g. in electronic commerce) over the Internet and the web has already lead to the need of the development of business-to-business applications. Integrating business processes of different organizations through network accessible software components faces several hurdles: (1) Different organizations could use different, incompatible implementation platforms. (2) Organizations may not want to share the internal details of their applications which can hinder integration. (3) No organization would want their application to get stuck due to pauses in availability of a software component residing in another organization.

Web services standards and technologies provide a framework for integration and interoperability of web accessible software applications by addressing these challenges as follows:

- Standardized data transmission via XML enables interaction among software components that are implemented using different platforms.
- Loose coupling of interacting services through standardized interfaces such as Web Service Definition Language (WSDL) provides a clear separation between the internals of an application and its interface visible to outside organizations.
- Use of asynchronous communication to lessen the effects of pauses in availability of other services and slow data transmission through the Internet.

In this short paper, we focus on asynchronous communication and its effects on the interaction behavior of composite web services. In asynchronous communication when a message is sent, it is inserted into a FIFO message queue, and the receiver consumes the message when it reaches to the front of the queue. This type of asynchronous communication is supported by message delivery platforms such as Java Message Service (JMS) [9], Microsoft Message Queuing Service (MSMQ) [10], WebSphere, Web Logic Integration, etc.

We focus on the behavior of composite web services that usually consist of many interacting component services. We consider two design approaches: "bottom-up" that starts from developing (identifying through discovery) component services, and "top-down" that derives the individual services from the specification for the composite service. A main goal is to analyze composite web services (obtained either through bottom-up or top-down) against properties specified in some logic language.

Modeling Interactions of Web Services

A *composite web service* consists of a set of *peers* (or component services) which communicate with each other using asynchronous messages. The messages exchanged among the peers are XML documents. This model fits nicely with the existing web service standards such as WSDL, BPEL (Business Process Execution Language for Web Services), and WSCL (Web Service Conversation Language [12]). In particular, we can view that each peer is defined as a BPEL service, while messages are received through its WSDL ports. As a starting point, we restrict each peer to a finite state machine that communicates with other peers through FIFO message queues.

A promising component of the web services framework that facilitates integration and interoperability is the conversation model [12][7][3][1]. A *conversation* is the sequence of messages exchanged among web services recorded in the order they are sent. Note that a conversation does not specify when the "receive" events occur, it only specifies the global ordering of the "send"

events. Conversations allows specification and analysis of interactions among web services. It is an intuitive model that is easy to understand and it allows specification and analysis of interactions without exposing the implementations details about the peers. For this reason, such a method of synchronizing interactions is also a part of the recently proposed Web Service Choreography Description Language (WS-CDL) [11].

Approaches to Design of Composite Web Services

There are two fundamentally distinguished approaches to composite web service design:

- In the *bottom-up* approach each peer participating in the web service composition is specified separately (as a state machine) and then the composed system can be studied by analyzing the combined behaviors of these individual peer specifications.

- In the *top-down* approach the desired global behavior is specified first and the detailed peer implementations are left blank initially. Any peer implementation that conforms to the desired global behavior is an acceptable implementation of a peer.

Conversation protocols are a top-down specification for composite web services. A *conversation protocol* is a finite state machine that specifies the desired set of conversations for a composite web service. A composite web service *realizes* a conversation protocol if the conversations accepted by the conversation protocol are exactly those generated by the composite web service. In other words, if the conversation sets of a conversation protocol and a composite web service are equal then we say that the composite web service realizes the conversation protocol.

A conversation protocol is *realizable* if there exists a composite web service that realizes it. It is known that not every conversation protocol is realizable; however, if a conversation protocol is realizable then it is realizable by its projections to each peer [5]. We project a conversation protocol to a peer p by replacing all transitions that have a send or a receive operation for which p is neither the sender nor the receiver with empty transitions. A conversation protocol is realizable if its projections to peers satisfy the "synchronous compatibility" and "autonomy" conditions and, additionally, if the conversation protocol satisfies the "lossless join" condition [5].

On the other hand, if we start from the peer implementation as in the bottom-up approach, the set of all possible conversations may not be identical to any conversation protocols. The reason is that the conversation sets of some composite web services may not be regular nor even context-free [3]. A sufficient

condition, called *synchronizability*, of the conversation sets of composite web services identical to conversation protocols is developed [5].

Model Checking Conversations

Given a composite web service where each peer is specified as a state machine, an interesting problem is to verify if the conversations generated by the composite web service satisfy certain properties. For example, it would be very useful to verify properties such as a *payment* message is always eventually followed by a *receipt* message. Such properties can be specified in Linear Temporal Logic (LTL) [4] using the temporal operators G (globally), F (eventually), X (next), and U (until). For example, the property above can be expressed in LTL as follows: $\mathsf{G}(payment \rightarrow \mathsf{F}receipt)$.

Model checking [4] is a technique for automated verification of temporal logic properties on finite state systems. There are tools such as the Spin model checker [8] which provide efficient implementation of the model checking techniques. However, most model checkers can only handle finite state systems, whereas asynchronous communication with unbounded message queues makes the state space of a composite web service infinite. One approach is to put an upper bound on the sizes of the message queues and transform the system to a finite state system. However, the state space of the composite web service can increase exponentially with the increasing queue sizes. This exponential increase in the state space can make verification of composite web services infeasible for large queue sizes even for a highly optimized finite state model checker such as Spin.

Known results indicate that automated verification of conversation behavior is not always possible in the presence of asynchronous communication with unbounded queues [2][5].

The synchronizability and realizability results lead to following verification strategies. For example, in the case of bottom-up specifications (design), the analysis strategy consists of the following steps:

1 We first check the synchronizability of the composite web service.

2 If the web service is synchronizable we verify the LTL properties on its conversations. (In this case the results we obtain hold for all conversations generated by the composite web service even in the presence of unbounded message queues.)

3 If the web service is not synchronizable we verify the LTL properties on its conversations by bounding the sizes of FIFO queues. In this case the verification results we obtain are guaranteed as long as the message queues remain within the specified bound. However if we find that a property is violated, then a counter-example generated using the model

checking techniques provide a concrete counter-example demonstrating the error.

A slightly different 3-step strategy is developed for top-down specification.

The synchronizability and realizability analysis have been implemented as a part of the Web Service Analysis Tool (WSAT) [6]. The front-end of WSAT accepts web service standards such as WSDL and BPEL. The core analysis engine of WSAT is based on an internal state machine representation. The back-end employs model checker Spin for verification. At the front-end, a translation algorithm from BPEL to the internal state machine representation is implemented, and support for other languages can be added without changing the analysis and the verification modules of the tool. WSAT also supports XML data manipulation by extending its internal state representation using transition guards written as XPath expressions. The synchronizability and realizability analysis are also extended to handle XML data manipulation. At the back-end, translation algorithms are implemented from the internal state machine representation to Spin. Based on the results of the realizability and the synchronizability analysis, the LTL verification at the back-end can be performed using the synchronous communication semantics instead of asynchronous communication semantics.

Conclusions

Conversation model provides a promising framework for analyzing interactions among web services. Asynchronous communication can effect the conversation behavior and if unbounded queues are used to model asynchronous communication then the verification of temporal logic properties of conversations becomes undecidable. We outlined two approaches to overcome the difficulties that arise in verification due to asynchronous communication. Synchronizability analysis identifies web service compositions for which the conversation behavior does not change when synchronous communication is replaced with asynchronous communication. This enables us to verify properties of conversations using the simpler synchronous communication semantics without giving up the benefits of asynchronous communication. On the other hand realizability analysis helps us to make sure that for top-down web service specifications asynchronous communication does not create unintended behaviors. This enables us to verify the conversation properties at a higher level of abstraction without considering the asynchronous communication semantics.

References

[1] B. Benatallah, F. Casati, and F. Toumani. Web service conversation modeling: A cornerstone for e-business automation. *IEEE Internet Computing*, 8(1):46–54, 2004.

[2] D. Brand and P. Zafiropulo. On communicating finite-state machines. *Journal of the ACM*, 30(2):323–342, 1983.

[3] T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. Int. World Wide Web Conf. (WWW)*, May 2003.

[4] E.M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 2000.

[5] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. *Theoretical Computer Science*, 328(1-2):19–37, 2004.

[6] X. Fu, T. Bultan, and J. Su. WSAT: A tool for formal analysis of web services. In *Proc. 16th Int. Conf. on Computer Aided Verification (CAV)*, pages 510–514, Boston, MA, July 2004.

[7] J. E. Hanson, P. Nandi, and S. Kumaran. Conversation support for business process integration. In *Proc. 6th IEEE Int. Enterprise Distributed Object Computing Conference*, 2002.

[8] G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, Boston, Massachusetts, 2003.

[9] Java Message Service. `http://java.sun.com/products/jms/`.

[10] MicroSoft Message Queuing Service. `http://www.microsoft.com/msmq/`.

[11] Web Services Choreography Description Language Version 1.0. `http://www.w3.org/TR/ws-cdl-10/`, December 2004.

[12] Web Services Conversation Language (WSCL) 1.0. `http://www.w3.org/TR/2002/NOTE-wscl10-20020314/`, March 2002.