

Analyzing Conversations of Web Services

Tevfik Bultan¹ Xiang Fu² Jianwen Su¹

¹ Department of Computer Science, University of California, Santa Barbara
Santa Barbara, CA 91306, USA. {bultan, su}@cs.ucsb.edu.

² School of Computer and Information Sciences, Georgia Southwestern State University
Americus, GA 31709, USA. xfu@canes.gsw.edu.

Abstract

A conversation is the global sequence of messages exchanged among the components of a distributed system. Conversations provide a promising model for specifying and analyzing the interactions among the peers participating to a composite web service. In this paper we discuss the following question: What is the impact of asynchronous communication on the conversation behavior? We show that the conversation behavior is significantly different for synchronous and asynchronous communication even if the local behaviors of the peers remain the same. We discuss two techniques for analyzing conversations: Synchronizability and realizability analyses. Synchronizability analysis is used to identify bottom-up web service specifications for which asynchronous communication does not change the conversation behavior. Realizability analysis, on the other hand, is used to identify top-down web service specifications which are realizable using asynchronous communication. We show that using the synchronizability and realizability analyses it is possible to automatically verify conversation behavior for bottom-up and top-down web service specifications using model checking.

Categories and Subject Descriptors

H.1.1 [Models and Principles]: Systems and Information Theory—(E.4) *formal models of communication*; D.2.4 [Software Engineering]: Software/Program Verification—*formal methods, model checking*

General Terms

Verification, Design

Keywords

Web services, asynchronous communication, conversations, synchronizability, realizability, model checking.

Introduction

Browser-based web accessible software applications have been the driving force behind electronic commerce. Nowadays one can look for and buy almost anything online, from a book to a car, using such applications. However, the success of the web technology in business-to-consumer applications does not immediately translate to business-to-business applications. Integrating business processes of different organizations through web accessible software components faces several hurdles:

- Different organizations could use different, incompatible implementation platforms. Given that there are competing implementation platforms such as .NET and J2EE, this is very likely to happen.
- Organizations may not want to share the internal details of their applications which can hinder integration.

- No organization would want their application to get stuck due to pauses in availability of a software component residing in another organization. Organizations may not tolerate such pauses in availability and slow data transmission through the Internet.

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 WSDL provides a clear separation between the internals of an application and its interface that is visible to outside organizations.
- Since in asynchronous communication the message exchange among two services does not require their synchronization, use of asynchronous communication lessens the effects of pauses in availability of other services and slow data transmission through the Internet.

In this paper, we will focus on asynchronous communication and its effects on the interaction behavior in composite web services. In asynchronous communication when a message is sent, it is inserted into a FIFO message queue, and the receiver consumes (i.e., receives) 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),¹ Microsoft Message Queuing Service (MSMQ),² IBM's WebSphere, BEA's Web Logic Integration, etc.

Anybody who appreciates the benefits of communication via e-mail instead of instant messaging or communication via voice-mail instead of phone conversations understands the benefits of asynchronous communication. Instant messaging and phone conversations are synchronous communication mediums in which both the sender and the receiver of a message have to be present for a message exchange to occur. In e-mail or voice-mail on the other hand, the sender can send a message while the receiver is busy doing other things. When the receiver is done with other tasks, he can access his voice- or e-mail and receive his messages. In asynchronous communication it is less likely for the sender or the receiver to get stuck due to unavailability of the other party and they are less likely to notice the delays in the delivery of the messages. These attributes make asynchronous communication more robust than the synchronous communication mechanisms such as remote procedure call (RPC)³ and therefore more suitable as the communication mechanism for web services.

Analyzing Interactions of Peers in Composite Web Services

In order to analyze their interactions we first need a model for composite web services. We assume that a composite web service consists of a set of peers which communicate with each other using asynchronous messages. Such a system can be modeled as a set of state machines (one for each peer) which communicate with each other using FIFO message queues. This model fits nicely with the existing web service standards. The messages that are exchanged among the peers are XML documents. WSDL can be used to declare the input, output message types for each peer and abstract BPEL processes can be used as the state machines specifying the behavioral interfaces of the peers.

A promising component of the web services framework that facilitates integration and interoperability is the conversation model.^{4,5,6,7} 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 provide a convenient model for 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 implementation

details about the peers. For this reason, a similar model for interactions is also a part of the recently proposed Web Service Choreography Description Language (WS-CDL).¹⁴

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 Logic¹¹ 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: $G(\text{payment} \Rightarrow F \text{ receipt})$. Other examples are: $\neg \text{receipt} U \text{ payment}$ i.e., a receipt message cannot be sent before a payment message is sent, and $G(\text{payment} \Rightarrow X \text{ receipt})$, i.e., the next message after a payment message is always a receipt message.

Model checking¹¹ is a technique for automated verification of temporal logic properties on finite state systems. There are tools, such as the Spin model checker,¹⁰ 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.

There are some theoretical results which show that automated verification of conversation behavior is not possible in the presence of asynchronous communication with unbounded queues. Communicating Finite State Machines¹² is a formal model for finite state machines which interact with asynchronous messaging. Addition of an asynchronous communication mechanism to finite state machines makes them as powerful as Turing machines.¹² Based on this result it is possible to show that model checking conversations of composite web services is an undecidable problem in the presence of asynchronous communication.^{8,16}

The above discussion demonstrates that asynchronous communication makes the verification of interactions of web services a more difficult problem. If we can find a way to eliminate asynchronous communication during verification and analysis of conversations while keeping it as a part of the implementation due to its benefits discussed earlier, we would have the best of both worlds. Web service implementations would be more robust due to flexibility brought by asynchronous communication but at the same time we would not pay the price for the added complexity asynchronous communication brings to verification and analysis of interactions. The synchronizability analysis we will discuss below achieves this by identifying the web service compositions for which asynchronous communication does not influence the interaction behavior.

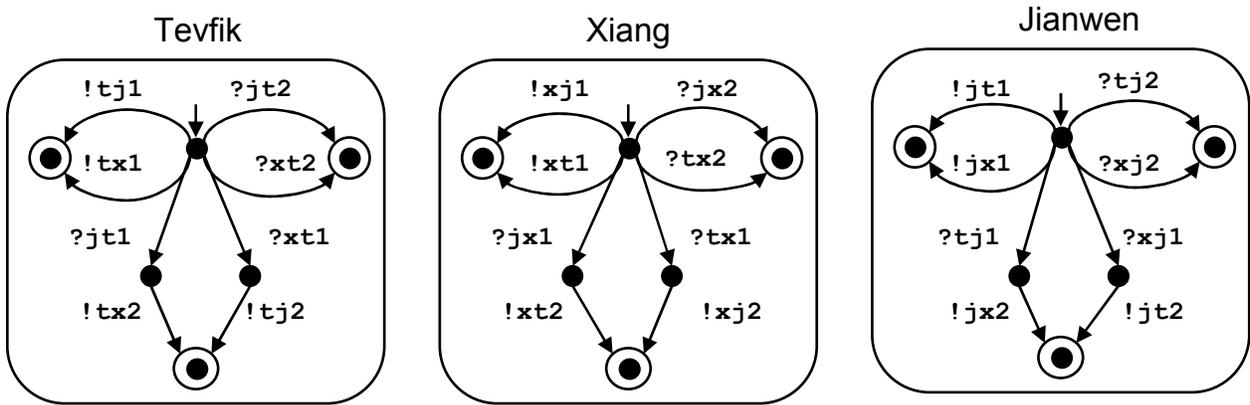


Figure 1. State machines for the three peers (Tevfik, Xiang, and Jianwen) for the First Caller Decides (FCD) Protocol for going to lunch. The name of each message consists of the initial of the sender, the initial of the receiver, and the message number, in that order. Send operations are labeled with “!” and receive operations are labeled with “?”. The initial states are marked with arrows and the double circles denote the final states.

Asynchronous vs. Synchronous Communication

An interesting issue is the influence of asynchronous communication on conversation behavior. Consider the following (fictional) example:

Before Xiang graduated from UCSB, we were using the following protocol for going to lunch. Sometime around noon one of us would call another one by phone and tell him where and when we would meet for lunch. The receiver of this first call would call the remaining peer and pass the information. Let’s call this protocol the First Caller Decides (FCD) protocol. The FCD protocol worked fine until the university installed a voice-mail system. After the installation of the voice-mail system we would sometimes show up at different restaurants without knowing where the others were. This got us thinking about the effect of asynchronous communication on the FCD protocol and we realized that the FCD protocol behaved differently with synchronous and asynchronous communication. Jianwen suggested that we change our lunch protocol as follows: As the most senior researcher among us he would make the first call to either Xiang or Tevfik and tell when and where we would meet for lunch. Then, the receiver of this call would pass the information to the other peer. Let’s call this protocol the Jianwen Decides (JD) protocol. When we switched to the JD protocol we realized that this protocol was working the same way with or without the use of voice-mail, i.e., the protocol behaved the same with synchronous and asynchronous communication. This led us to investigate techniques for recognizing such protocols. We were able to develop a set of conditions for identifying such protocols which we will explain below.

The example above can be modeled as a composite web service consisting of three peers: Tevfik (T), Xiang (X) and Jianwen (J). Figure 1 shows the state machines for the three peers for the FCD protocol. Transitions are labeled with send (the ones with the prefix “!”) or receive (the ones with the prefix “?”) operations. Each message is labeled with the sender, the receiver and a number denoting if the message is the first message sent or the second message sent. Let us first consider the behavior of the FCD protocol with synchronous communication. Consider the peer T. Initially T can do one of the following, send $tj1$ or $tx1$, or receive $jt1$, $xt1$, $jt2$, or $xt2$. If he receives $jt1$ he has to send $tx2$ to X and if he receives

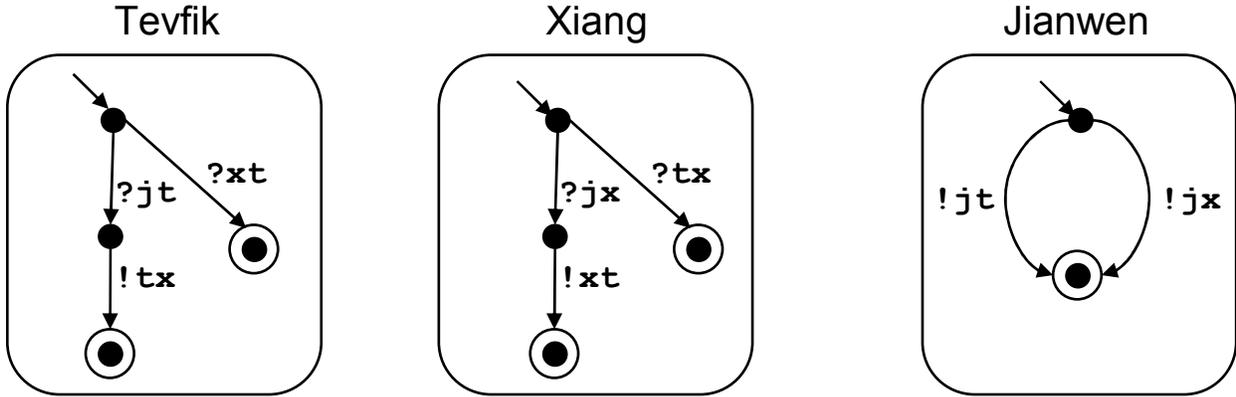


Figure 2. State machines of the three peers for the Jianwen Decides (JD) Protocol.

x_{t1} he has to send t_{j2} to J. Consider the scenario in which T sends t_{x1} . After receiving t_{x1} , X will send x_{j2} to J. Hence, this execution generates the following conversation: (t_{x1}, x_{j2}) . When synchronous communication is used, the set of all conversations generated by the state machines shown in Figure 1 is: $\{(t_{x1}, x_{j2}), (t_{j1}, j_{x2}), (x_{t1}, t_{j2}), (x_{j1}, j_{t2}), (j_{t1}, t_{x2}), (j_{x1}, x_{t2})\}$.

Now, consider the FCD protocol with asynchronous communication. Assume that first T sends the message t_{x1} to X and then J sends the message j_{x1} to X. J and T have no way of knowing that they both sent the first message. X, on the other hand, cannot block J or T since based on the asynchronous communication semantics when a message is sent it is inserted to the message queue even if the receiver may not be able to receive it. During this execution, the generated conversation will be (t_{x1}, j_{x1}, x_{j2}) and the messages j_{x1} and x_{j2} will be left in the message queues of X and J without being consumed. Note that this message sequence cannot be generated when synchronous communication is used. If synchronous communication is used, then T and X would execute the send and receive actions for the message t_{x1} at the same time. After that J cannot send the message j_{x1} to X since X will not be in a state where it can synchronize with J to receive it. Hence, the only operation that J can execute at that point is to wait and receive x_{j2} . This example shows that the conversation behavior of the FCD protocol is different for synchronous and asynchronous communication.

Now, let us investigate the JD protocol. Figure 2 shows the state machines for the JD protocol. Even if we replace synchronous communication with asynchronous communication, the set of conversations generated by the JD protocol remains the same, which is $\{(j_{t1}, t_{x1}), (j_{x1}, x_{t1})\}$. Note that this is a very useful property since the state space of a composite web service is finite if synchronous communication is used and if the peer specifications are finite state. This means that we can automatically verify the properties of conversations for a composite web service if we can show that its conversation set does not change when synchronous communication is replaced with asynchronous communication.

Synchronizability Analysis

We call a composite web service *synchronizable* if the set of conversations it generates is the same under synchronous and asynchronous communication. Synchronizability analysis identifies synchronizable composite web services.^{9,15,16}

We define the *synchronous product* of a composite web service as a state machine whose set of states is the product of the sets of states of the state machines for each peer. The transitions of the product machine are defined based on the synchronous communication semantics. A transition of the product machine

changes the states of both the sender and the receiver of the message according to their state machine specifications. The initial state of the product machine is the product state where all the peers are in their initial states. A product state is a final state if all the peers are in their final states. Figure 3 shows the product machines for the FCD and JD protocols.

We identified sufficient conditions for synchronizability. We call these conditions *synchronous compatibility* and *autonomy* conditions.^{8,9,15,16} If a composite web service is synchronous compatible and autonomous, then it is synchronizable.

- **Synchronous Compatibility Condition:** A web service composition is synchronous compatible if its synchronous product does not contain a state in which there exists a peer which has a send transition from its current local state, however, the corresponding receiver peer is not in a state where it can receive that message.
- **Autonomy Condition:** A web service composition is autonomous if, given any state of any of the peers, only one of the following three conditions hold 1) all the transitions from that state are send transitions, 2) all the transitions from that state are receive transitions, or 3) that state is a final state and there are no send or receive transitions from that state.

The JD protocol satisfies the synchronous compatibility condition which can be checked by investigating the corresponding product machine shown in Figure 3. JD protocol also satisfies the autonomy condition. Hence our synchronizability conditions are able to show that JD protocol is synchronizable.

FCD protocol, however, does not satisfy either of the synchronizability conditions. For example the first state of the peer T is the source of a send transition for message $\tau_j 1$ and also the source of a receive transition for message $j \tau 1$, violating the autonomy condition.

As we discussed above synchronizability analysis can be used to verify properties of conversations of composite web services with respect to unbounded message queues. It can also help in verification of specifications with bounded message queues by reducing the size of the state space by removing the configurations of the message queues.

Conversation Protocols and Realizability Analysis

A composite web service can be specified in two different ways :

- 1) In the *bottom-up* approach each peer participating to the web service composition is specified separately as a state machine and then the composed system is studied by analyzing the combined behaviors of individual peer specifications. This is the specification approach used in the examples we gave above.
- 2) In the *top-down* approach first the desired global behavior is specified and the detailed peer implementations are left blank. Any peer implementation that conforms to the desired global behavior is an acceptable implementation of a peer.

Conversation protocols are a top-down specification formalism for composite web services.^{8,16} 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 all the conversations accepted by the conversation protocol are generated by the composite web service and any conversation that is not accepted by the conversation protocol is not 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.

Consider the state machines given in Figure 3. These two state machines are the conversation protocols for the FCD and JD examples. Note that, these state machines do not specify the behaviors of the

individual peers. Rather, they specify the global ordering of the send events. It is easy to see that the conversation set specified by the FCD protocol in Figure 3 is: $\{(tx1, xj2), (tj1, jx2), (xt1, tj2), (xj1, jt2), (jt1, tx2), (jx1, xt2)\}$ and the conversation set specified by the JD protocol in Figure 3 is: $\{(jt, tx), (jx, xt)\}$.

We call a conversation protocol *realizable* if there exists a web service composition which generates the same conversation set using asynchronous communication.^{8,16} Based on our earlier discussions we know that the composite web service shown in Figure 2 realizes the JD protocol shown in Figure 3, i.e., JD protocol is realizable. It is also possible to show that the composite web service shown in Figure 1 does not realize the FCD protocol shown in Figure 3. However, this does not necessarily mean that FCD protocol is not realizable. In our earlier work⁸ we have shown that if a conversation protocol is realizable then it is realizable by its projections to each peer. We project a conversation protocol to a peer P by replacing all the transitions that have a send or a receive operation for which P is neither the sender nor the receiver with ϵ -transitions. The ϵ -transitions can later be removed by determinization. The state machines shown in Figure 2 are the projections of the JD protocol shown in Figure 3 and the state machines shown in Figure 1 are the projections of the FCD protocol shown in Figure 3. Since we showed that the conversation set generated by the web service composition in Figure 1 is not same as the conversation set generated by the FCD protocol in Figure 3 we conclude that the FCD protocol is not realizable.

There is a close relation between the synchronizability and realizability analysis. A conversation protocol is realizable if its projections to peers satisfy the synchronous compatibility and autonomy condition and, additionally, if the conversation protocol satisfies the lossless join condition.^{8,15,16}

A conversation protocol satisfies the *lossless join* condition if its conversation set is equal to the join of its projections to each peer. The *projection* of a conversation to a peer is the message sequence generated by removing all the messages that are not related to that peer from the given conversation. For example, consider the JD protocol from Figure 3. As we discussed above, the conversation set for this protocol is $\{(jt, tx), (jx, xt)\}$. Projection of this conversation set to peer T is $\{(jt, tx), (xt)\}$, to peer X is $\{(tx), (jx, xt)\}$ and to peer J is $\{(jt), (jx)\}$. Note that we removed each message for which a peer is neither the sender nor the receiver from its projection to that peer. Given a set of conversation sets (one conversation set for each peer), the *join* of them includes all conversations whose projection to each peer is included in the conversation set of that peer. When we take the join of the sets $\{(jt, tx), (xt)\}$ for T, $\{(tx), (jx, xt)\}$ for X, and $\{(jt), (jx)\}$ for J, we get the set $\{(jt, tx), (jx, xt)\}$ back, which shows that the JD protocol is lossless join.

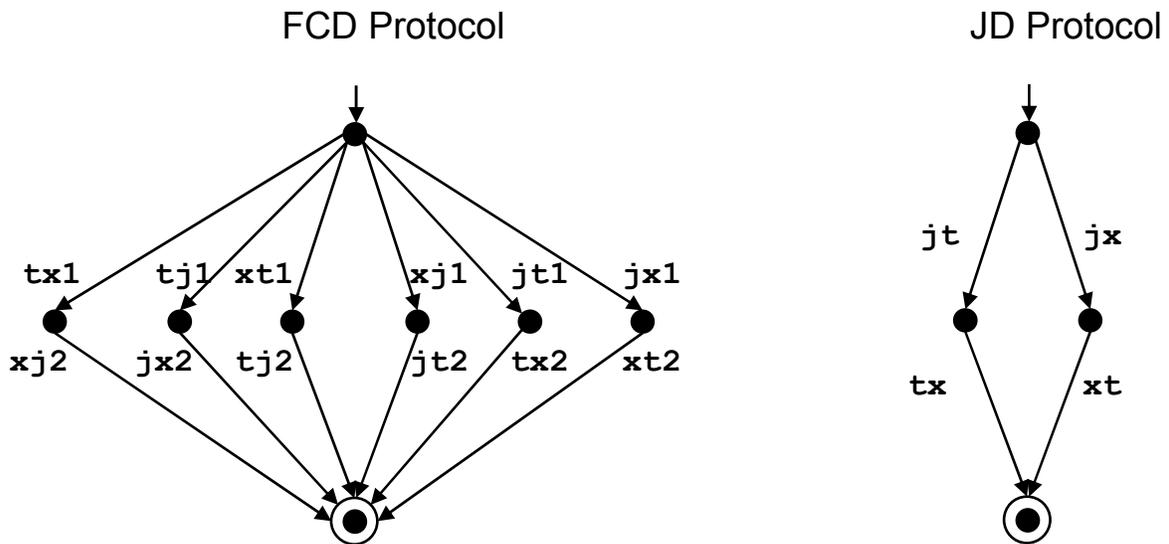


Figure 3. Conversation protocols for FCD and JD.

Another Example

The effect of asynchronous communication can be observed even in very simple interactions.⁹ Consider the example in Figure 4 which describes a simple Client-Server interaction. The Server replies to each request message (`req`) sent by the Client with an acknowledgement message (`ack`). The interaction terminates when the Client sends an `end` message. Figure 4(a) and Figure 4(b) show two different implementations by the Client where the Server implementation is the same for both of them. In Figure 4(a) the Client does not wait for an `ack` message after it sends a `req` message. It arbitrarily interleaves the sending of the `req` messages and the receiving of the `ack` messages. In Figure 4(b), on the other hand, the Client waits for an `ack` message before it sends the next `req` message. The implementation shown in Figure 4(b) is synchronizable and its conversation set is characterized by the conversation protocol shown in Figure 4(c). However, the implementation shown in Figure 4(a) is not synchronizable and it is not possible to characterize its conversation set using a finite state machine. Note that, in the conversations generated by the peers shown in Figure 4(a), the `ack` messages can lag behind the `req` messages. In fact, the difference between the number of `req` messages and the number of `ack` messages can be arbitrarily large. However, in each prefix of any conversation, the number of `req` messages is greater than or equal to the number of `ack` messages, and for a complete conversation the number of `req` messages is equal to the number of `ack` messages. It is not possible to characterize such a conversation set using a finite state machine since a finite state machine cannot keep track of the number of unacknowledged `req` messages which can be arbitrarily large.

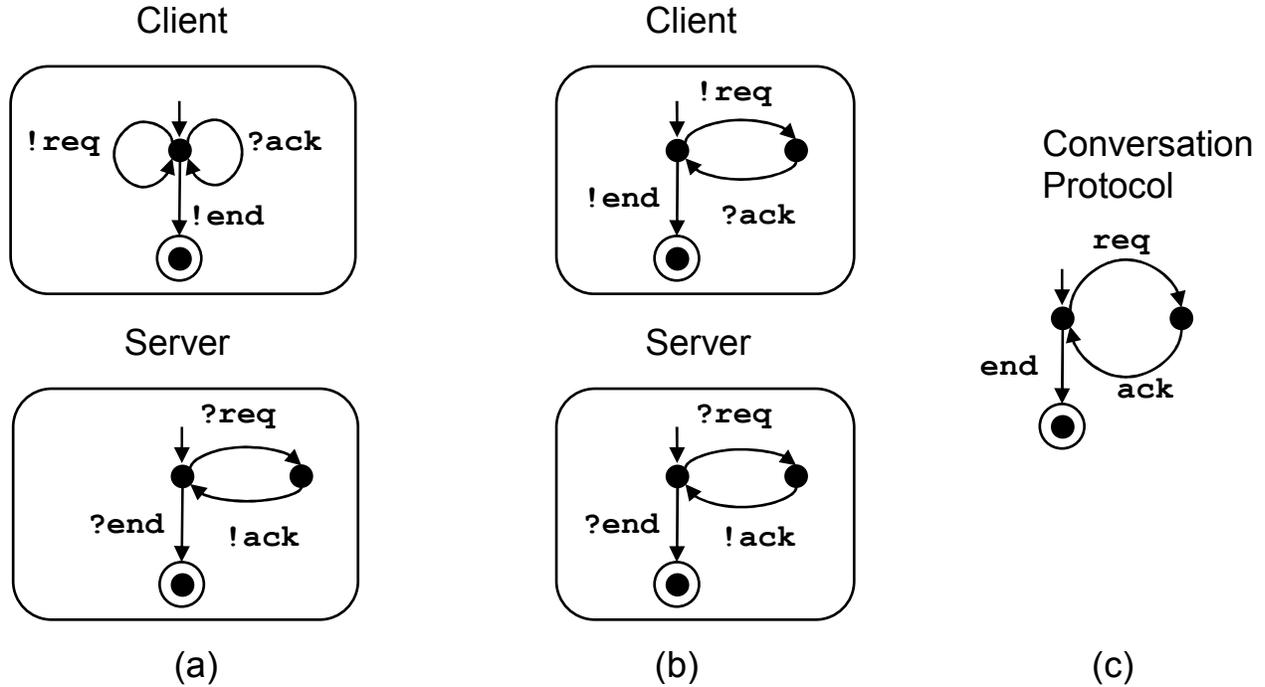


Figure 4. Two Client-Server implementations where (a) is not synchronizable and (b) is synchronizable. The implementation shown in (b) is a realization of the conversation protocol shown in (c).

Model Checking Conversations

The synchronizability and realizability results discussed above lead to following verification strategies. For bottom-up specifications:

1. We first check the synchronizability of the composite web service.
2. If the web service is synchronizable we verify the LTL properties about its conversations using synchronous communication semantics. 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 about its conversations by bounding the sizes of the communication channels. In this case the verification results we obtain are guaranteed as long as the message queues remain within the bounds. However if we find that a property is violated, then the counter-example generated using the model checking techniques provide a concrete counter-example demonstrating the error.

For top-down specifications:

- 1) We first check the realizability of the conversation protocol. If the conversation protocol is not realizable it should be modified until a realizable conversation protocol is obtained.
- 2) We check the LTL properties about the conversations on the conversation protocol.
- 3) After the verification step is done, the conversation protocol is projected to each peer, and the composition of these peer implementations will preserve the LTL properties that are verified, even for the asynchronous communication with unbounded queues.

The synchronizability and realizability analyses discussed in this paper have been implemented as a part of our Web Service Analysis Tool (WSAT).^{13,17} The front-end of WSAT accepts industry 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 the Spin model checker for verification. At the front-end, a translation algorithm from BPEL4WS 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. At the core analysis part, the synchronizability and realizability analyses are implemented. The synchronizability and realizability analyses are also extended to handle XML data manipulation. At the back-end, translation algorithms are implemented from the internal state machine representation to Promela, the input language of Spin. Based on the results of the realizability and the synchronizability analyses, the LTL verification at the back-end can be performed using the synchronous communication semantics instead of asynchronous communication semantics.

We applied WSAT to a range of examples, including six conversation protocols converted from the IBM Conversation Support Project, five BPEL4WS services from BPEL4WS standard and Collaxa.com. We applied the synchronizability and the realizability analyses to these examples, and except for two conversation protocols, all examples were shown to be either realizable or synchronizable.¹⁵ These experiments demonstrate that the conditions in our synchronizability and realizability analyses are not too restrictive and they are able to capture most practical applications.

Conclusions

We believe that the synchronizability and realizability analyses are useful tools for achieving a better understanding of interaction behavior for asynchronously communicating systems. It is important to understand the effects of asynchronous communication on the behavior in order to avoid hard to find bugs that depend on the ordering of messages in a distributed system. Since testing and debugging of web services is especially difficult due to their distributed nature, techniques for automated verification of their interactions could be very valuable in practice.

Conversation model provides a promising framework for analyzing interactions among web services. However, as we discussed in this paper, 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.

As a final note, the synchronizability and realizability conditions for conversations we discussed in this paper are *sufficient* conditions. Finding *necessary and sufficient* conditions for synchronizability and realizability of conversations is an open problem. We also do not know yet if synchronizability and realizability of conversations are decidable problems. However, for most example web services we analyzed we were able to show synchronizability or realizability using the sufficient conditions discussed above. There is also more work to be done in extending the synchronizability and realizability analyses to specifications with XML data manipulation and in which XML data content influences the control flow.

References

1. Java Message Service. <http://java.sun.com/products/jms/>.
2. Microsoft Message Queuing Service. <http://www.microsoft.com/msmq/>.

3. D. A. Menasce, "MOM vs. RPC: Communication Models for Distributed Applications," *IEEE Internet Computing*, March–April 2005, pp. 90–93.
4. Web Services Conversation Language (WSCL). <http://www.w3.org/TR/2002/NOTE-wscl10-20020314/>
5. J. E. Hanson, P. Nandi and S. Kumaran, "Conversation Support for Business Process Integration," *Proc. 6th IEEE Int'l. Enterprise Distributed Object Computing Conf.*, 2002, pp. 65–74.
6. B. Benatallah, F. Casati and F. Toumani, "Web Service Conversation Modeling." *IEEE Internet Computing*, January–February 2004, pp. 46–54.
7. T. Bultan, X. Fu, R. Hull, and J. Su. "Conversation Specification: A New Approach to Design and Analysis of E-Service Composition," *Proc. 12th Int'l World Wide Web Conference (WWW 2003)*, May 2003, pp. 403–410.
8. X. Fu, T. Bultan and J. Su. "Conversation Protocols: A Formalism for Specification and Verification of Reactive Electronic Services," *Theoretical Computer Science*, vol. 328, no. 1-2, November 2004, pp. 19–37.
9. X. Fu, T. Bultan and J. Su. "Analysis of Interacting BPEL Web Services," *Proc. 13th Int'l World Wide Web Conference (WWW 2004)*, May 2004, pp. 621–630.
10. G. Holzmann, *SPIN Model Checker: Primer and Reference Manual*, Addison-Wesley, 2003.
11. E.M. Clarke, O. Grumberg and D. A. Peled, *Model Checking*, The MIT Press, 1999.
12. D. Brand and P. Zafiropulo, "Communicating Finite-State Machines," *Journal of the ACM*, vol. 30, no. 2, 1983, pp. 323–342.
13. X. Fu, T. Bultan and J. Su. "WSAT: A Tool for Formal Analysis of Web Services," *Proc. 16th Int'l Conference on Computer Aided Verification (CAV 2004)*, July 13-17, 2004, pp. 510-514.
14. Web Services Choreography Description Language (WS-CDL). <http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217/>
15. X. Fu, T. Bultan and J. Su. "Synchronizability of Conversations Among Web Services," Technical Report 2005-28, Department of Computer Science, University of California at Santa Barbara, November 2005.
16. X. Fu, "Formal Specification and Verification of Asynchronously Communicating Web Services," Ph.D. Thesis, University of California, Santa Barbara, June, 2004.
17. Web Service Analysis Tool (WSAT). <http://www.cs.ucs.edu/~su/WSAT/>.