A Top-Down Approach to Modeling Global Behaviors of Web Services

Xiang Fu Tevfik Bultan Jianwen Su

Department of Computer Science
University of California at Santa Barbara
{fuxiang, bultan, su}@cs.ucsb.edu

Abstract

Due to the distributed nature of modern composite web services, designers are facing new challenges in both requirement specification as well as logic validation. This paper proposes a top-down design/verification strategy that helps construct composite web services to meet preset system goals. The key to this approach is to specify desired global behaviors with a "conversation protocol" and verify preset system goals on the global protocol. Then peer implementations are synthesized from the conversation protocol. Three realizability conditions are provided to guarantee that the composition of synthesized peers will satisfy the previously verified system goals.

1 Introduction

Web services are revolutionizing the way that many e-commerce, consumer software, and telecommunication applications are provided, as indicated by the rapid growing development in the industry standards (e.g., SOAP, UDDI, WSDL, BPEL4WS) and technology (e.g., IBM's Web services Toolkit, Sun's Open Net Environment and JiniTM Network technology, Microsoft's .Net and Novell's One Net initiatives, HP's e-speak). Research communities are providing complimentary technologies from different perspectives. Modeling at a more fundamental level both e-services themselves, and frameworks for combining them have been studied in [5, 8, 14, 15, 29, 16, 2, 1, 17]. New languages for defining services were proposed in [3, 19]. Specialized type systems were considered in [22]. Finally, tools were developed for annotating e-services and for planning, aiming at combining web services automatically to achieve a specified functionality [25, 4, 27, 13]. In this paper, we discuss the issues and techniques in the design, specification, and verification of composite web services.

Since each component of a composite web service is autonomous, no single peer has the control over the global interaction process. Such a distributed nature makes it extremely hard to ensure the correctness of the composite web service merely through the design of each peer individually. In this paper, we argue for a top-down approach from a global perspective in specification and design of web services. On one hand, we show that the bottom-up approach of designing composite web services may result in more complex global behaviors. On the other hand, we illustrate that the top-down approach may further enable existing tools for verification of web services.

In this paper we extend a web service model introduced in [10] and further studied in [20]. A composite web service in this model consists of a set of peers that communicate via asynchronous message passing. In particular, each peer is modeled using a guarded finite state automaton, which abstracts emerging web service choreography standards (e.g. BPEL4WS [6], WSCI [30], BPML [7], ebXml [18]) to characterize behaviors of complex long running services. The asynchronous message passing is achieved by associating each peer with a queue for storing its input messages. This FIFO queue based model resembles many industry efforts like Java Message Service (JMS) [24] and Microsoft Message Queuing Service (MSMQ) [26]. Unlike JMS and MSMQ, there is a virtual "global watcher" in our model that "records" the sequence of messages as they are sent by the peers. A central focus on the global behavior of a web service is to study the set of message sequences generated by the web service, where temporal logics such as LTL [28] can be extended to this framework to specify "good" behaviors. Our previous work in [10] and [20] concentrates on a contentless message model and on how to design a "realizable" global specification, from which (FSA) peers can be synthesized to ensure specified global behaviors without a global coordinator.

Specifically, we define a *conversation protocol* as a set of permissible sequences of messages observed by the global watcher. In [10, 20], we show that it is possible to realize a

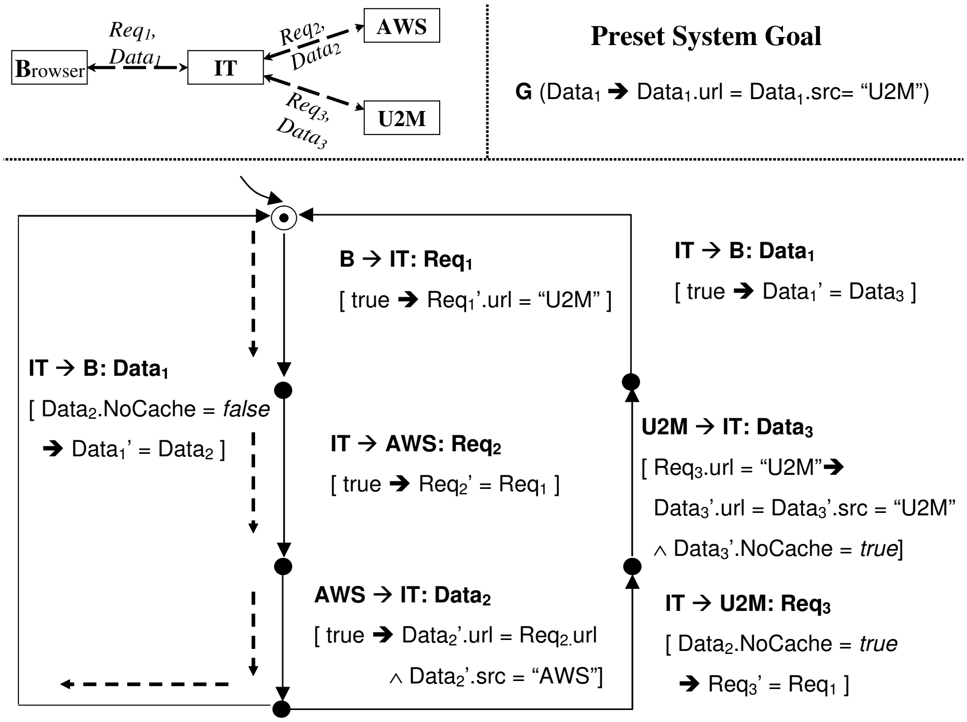

Figure 1. Global conversation specification

conversation protocol using a set of finite state peers, if the protocol satisfies three conditions. Our framework enables a top-down verification strategy where

1. A conversation protocol is specified by a realizable Büchi automaton [9],
2. The properties of the protocol are verified on the Büchi automata specification, and
3. The peer implementations for the conversation protocol are synthesized from the Büchi automaton via projection.

In contrast, we also present a negative fact about the alternative bottom-up approach of specifying the peers of a composite web service in isolation. We show that the composition of finite state peers may result in a non-ω-regular behavior set globally, which makes it difficult to use model checking techniques. In this paper, we generalize the framework of [10, 20], which considers only message classes (names), by allowing messages to have contents. We show that this technique can be used to tackle the U2M problem described in the workshop announcement.

This paper is organized as follows. Section 2 illustrates the conversation protocols with the U2M example in the workshop announcement. Section 3 defines a variation of LTL logic to express system goals such as the "freshness" requirement in U2M scenario. We apply formal model checking techniques in Section 4. In Section 5, we synthesize each peer based on the global conversation protocol in Section 4. As a comparison, Section 6 shows a negative result concerning the bottom up approach. Section 7 concludes the paper.

2 Conversation Protocols

Consider the UpToTheMinuteNews.com (U2M) example: A user accessing the pages of U2M using a web Browser has to go through a Corporate IT (IT) web proxy on the corporate firewall. A company called Acme Web Speedup Services (AWS) provides a caching proxy web service which is used by IT for accelerating web access. This has the undesirable effect of displaying stale web pages from U2M at the user's Browser.

We argue that the system goal that Browser always receives fresh web pages from U2M is fundamentally a global constraint. Although one could derive ad hoc solutions that are local, it is more desirable to obtain a more general global solution, depending on the properties of IT, AWS, and U2M. In Figure 1, we present a conversation protocol specifying the global web service, which consists of four peers: Browser, IT, AWS, and U2M.

A *conversation protocol* is a guarded Büchi automaton enhanced with message contents, and each transition of the automaton consists of two parts:

1. a *message* transmitted from one peer to another, and

2. a *transition guard* that specifies the condition to take the transition as well as assigns the contents of the message being sent.

We use a conversation protocol to characterize the set of *conversations*, i.e., all possible sequences of messages communicated between peers. Then we check whether the conversations meet some preset goals.

As shown in Figure 1, there are two types of messages in the U2M scenario: `Req` and `Data`. Note that we use subscripts to distinguish the same message class transmitted on different channels, e.g. $\texttt{Req}_1$ and $\texttt{Req}_3$. The two message classes are declared in the following.

```
class Req{              class Data{
   string url;             string url;
   ...                     string src;
}                          bool   NoCache;
                           string htmlPage;
                           ...
                        }
```

Message class `Req` represents an http request, and its attribute `url` contains the address (original source) of the requested web page. Message class `Data` is the response, where `htmlPage` is the web page content, `src` is the actual address it is retrieved from (e.g., a cache server), and attribute `NoCache` is set to `true` if the header of `htmlPage` contains a "no-cache" tag.

In Figure 1, each transition guard is written in the form of "*condition* → *assignment*". Take as an example the transition labeled with "$\texttt{IT} \rightarrow \texttt{U2M: Req}_3$". The condition "$\texttt{Data}_2\texttt{.NoCache} = \textit{true}$" means that only if the web page returned from AWS contains a "no-cache" tag can the transition take place. The assignment "$\texttt{Req}_3{}' = \texttt{Req}_1$" means that IT simply relays the request $\texttt{Req}_1$. Note that here primed variables refer to the contents of the current message being sent, and non-primed variables denote the corresponding fields of the latest transmitted message of that message class.

Intuitively, the desired conversations specified by the protocol in Figure 1 are as follows. In each round of a conversation, the first message is a request ($\texttt{Req}_1$) from Browser to IT. IT relays this request to cache service AWS, and waits for its response $\texttt{Data}_2$. AWS guarantees that $\texttt{Data}_2$ is a matching response for $\texttt{Req}_2$, by ensuring that their `url` are equal; and AWS also sets the actual source `src` of the response to the value "AWS". IT then examines the contents of $\texttt{Data}_2$ from AWS, if the page does not contain a "no-cache" tag, IT just sends this cached page to Browser; otherwise, it will fetch the page directly from U2M. Note that U2M guarantees that each page it sends contains the "no-cache" tag, and their `url` and `src` are properly set.

3 Using LTL to State the System Goal

Now the immediate question is how to express the preset system goal that Browser should always get non-cached U2M news pages from IT. We extend the linear temporal logic (LTL) [28] to fit into our message passing framework. To facilitate the discussion, we clarify some of the technical notions first. Given a conversation $w = w_0, w_1, w_2, \ldots$, a sequence of *messages* with contents, let w_i denote the i-th message in w, and $w^i = w_i, w_{i+1}, w_{i+2}, \ldots$ denote the i-th suffix of w. An *atomic proposition* is either in the form of c where c is a message class, or $c.pred$ where $pred$ is a predicate over the attributes of c.

Let AP be the set of atomic propositions. A message m is said to *satisfy* an atomic proposition $\psi \in AP$, written as $m \models \psi$, if

1. when ψ is a message class, the type of m is ψ, and

2. when ψ is in the form of $c.pred$, then the type of m is c and $pred(m) = true$.

LTL properties are constructed from such atomic propositions, logical operators $\wedge, \vee, \neg$, and LTL operators **X**, **G**, **U**, **F**. Given LTL formulas ϕ, and φ, and an atomic proposition $\psi \in AP$,

$$\begin{array}{lll}
w \models \psi & \text{iff} & w_0 \models \psi \text{ if } \psi \in AP \\
w \models \neg\phi & \text{iff} & w \not\models \phi \\
w \models \phi \wedge \varphi & \text{iff} & w \models \phi \text{ and } w \models \varphi \\
w \models \phi \vee \varphi & \text{iff} & w \models \phi \text{ or } w \models \varphi \\
w \models \mathbf{X}\phi & \text{iff} & w^1 \models \phi \\
w \models \mathbf{G}\phi & \text{iff} & \text{for all } i \geq 0, w^i \models \phi \\
w \models \mathbf{F}\phi & \text{iff} & \text{there exists } i \geq 0, w^i \models \phi \\
w \models \phi\mathbf{U}\varphi & \text{iff} & \text{there exists } j \geq 0, \text{ s.t. } w^j \models \varphi \\
 & & \text{and, for all } 0 \leq i < j, w^i \models \phi
\end{array}$$

Intuitively temporal operator **X** means "next", **G** means "globally", **F** means "eventually", and **U** means "until". We give some examples of LTL properties and their semantics in the following.

1. **G** `Data` : every message appeared in the conversation is of type `Data`.

2. **G**(`Req.url="U2M"` ⇒ **F** `Data.url="U2M"`) : for each message `Req` with `url` equal to "`U2M`" eventually there is a matching response `Data` with `url` equal to "`U2M`".

Similarly, the "freshness" system goal of U2M scenario can be expressed as

$$\mathbf{G}(\texttt{Data}_1 \Rightarrow \texttt{Data}_1\texttt{.url=Data}_1\texttt{.src="U2M"}) \quad (1)$$

That is, every U2M news page retrieved by IT should be a non-cached fresh page.

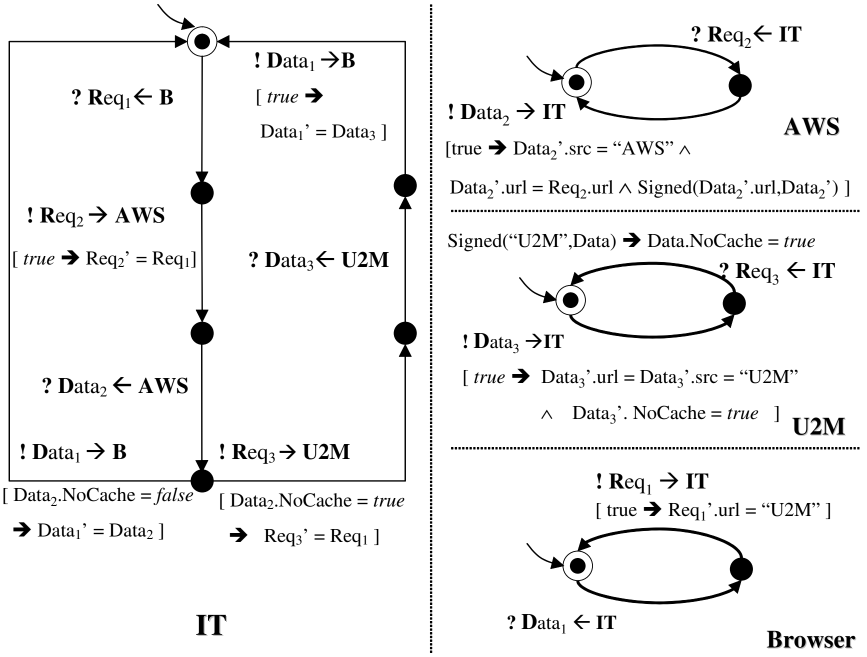
? Req$_1$← B
! Data$_1$ →B
[*true* →
Data$_1$' = Data$_3$]
! Req$_2$→ AWS
[*true* → Req$_2$' = Req$_1$]
? Data$_3$← U2M
? Data$_2$ ← AWS
! Data$_1$ → B
! Req$_3$→ U2M
[Data$_2$.NoCache = *false*
→ Data$_1$' = Data$_2$]
[Data$_2$.NoCache = *true*
→ Req$_3$' = Req$_1$]
IT
? Req$_2$← IT
! Data$_2$ → IT
AWS
[true → Data$_2$'.src = "AWS" ∧
Data$_2$'.url = Req$_2$.url ∧ Signed(Data$_2$'.url,Data$_2$')]
Signed("U2M",Data) → Data.NoCache = *true*
? Req$_3$ ← IT
! Data$_3$ →IT
[*true* → Data$_3$'.url = Data$_3$'.src = "U2M"
∧ Data$_3$'. NoCache = *true*]
U2M
! Req$_1$ → IT
[true → Req$_1$'.url = "U2M"]
? Data$_1$ ← IT
Browser

Figure 2. Synthesized implementation of each peer

4 Model Checking the U2M Design

Given a Büchi specification of conversation protocol, it is possible to transform it into the language of a model checker, such as Spin [23], SMV [12], and Action Language Verifier [11]. Note that, depending on the restrictions on data types and domains imposed by model checkers, the translation may require abstractions. After the translation, we can verify whether the proposed system goal is satisfied by the conversation protocol using model checking. For the example presented in Figure 1, model checking can reveal that the LTL property (1) is not guaranteed by the initial design, and an error trace is marked using dashed arrows in Figure 1. The problem with the initial design is that AWS may forge a page whose attribute NoCache is false, which is later relayed by IT to the Browser. Thus we need to require that AWS is always "honest". To express this concept, we introduce a predicate Signed(url: site, Data: page), which means intuitively that page is digitally signed by the web service at url site. Then the following formula can be inserted into the guard of the transition $\texttt{AWS} \rightarrow \texttt{IT} : \texttt{Data}_2$ in Figure 1.

$$\texttt{Signed(Data}_2\texttt{.url',Data}_2\texttt{')} = \texttt{true} \qquad (2)$$

Interestingly, even if AWS makes the "no-deception" promise, it still cannot guarantee the freshness requirement. For example, if at some point, U2M forgets to insert "no-cache" tag into its web page, and somehow this page happens to be stored in AWS. When IT requests the page, AWS can send this digitally signed "bad" page to IT which causes the failure of freshness. Therefore if we strengthen the design of U2M with the following system assumption:

$$\texttt{Signed("U2M", data)} \Rightarrow \texttt{data.NoCache} = \texttt{true} \qquad (3)$$

we can safely reach the conclusion that the LTL property (1) is satisfied. Model checking of the new composed system with guard (2) and system assumption (3) requires the ability to handle first order formulas.

5 Synthesis of Peers

Synthesis of peers is obtained by projecting the conversation protocol to each peer by removing non-relevant transitions for each peer. Then we detach guards for those transitions that are labeled with incoming messages, since a peer cannot control the contents of its incoming messages. The projection results in a guarded Büchi automaton for each peer. As an example, in Figure 2, we present the synthesized peers for the refined version (enhanced with Equations 2 and 3) of the U2M example in Figure 1.

It can be verified that, in an asynchronous message passing environment (where a FIFO queue is used to store incoming messages), the composition of finite state peers in Figure 2 produces exactly the same conversation set as described by the refined protocol of Figure 1. However, not every conversation protocol has this "realizable" property.

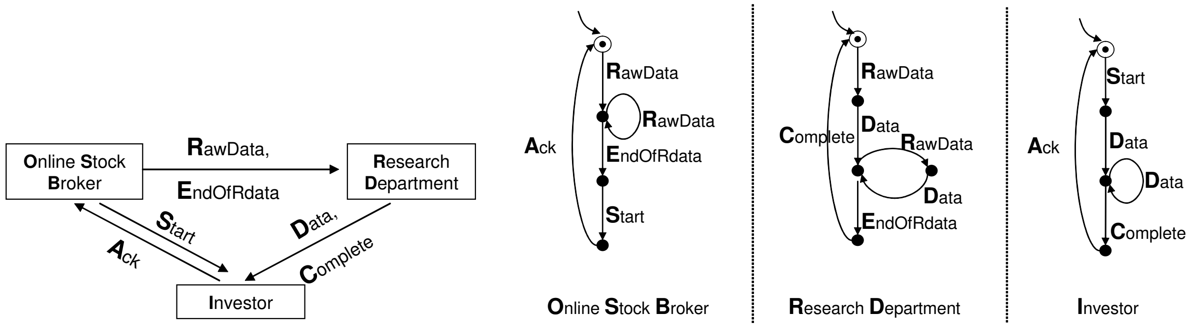

Figure 3. Fresh Market Update Service

In [20], we presented three conditions that can ensure the realizability of a conversation protocol. We briefly introduce them below:

Lossless join property requires that a conversation protocol should be equivalent to the Cartesian product of its projections to each peer.

Autonomous property requires that at any moment according to the protocol, each peer can make a deterministic decision on whether to wait, or to send, or to terminate.

Synchronous compatible property requires that there is no "illegal" state in a conversation protocol where some peer is ready to send a message that is not expected by its receiver.

We argue in [20] that conversation protocols satisfying these three realizability conditions can still capture a large category of web service patterns. However results in [20] cannot be directly applied to conversation protocols with message contents. In [21] we show that by employing the state space exploration technique, for a conversation protocol with finite domains, we can always construct a standard guardless protocol which bisimulates the original protocol. Running realizability check on its guardless bisimulation usually suffices to justify a realizable guarded conversation protocol with message contents.

6 Problems of Bottom-up Approach

One natural question concerning the bottom-up specification of composite web services, i.e., to specify each single peer first and then compose them, is whether we can always construct such a global conversation specification recognized by a finite state automaton? A positive answer would imply that many verification techniques become immediately available. Unfortunately, we show that the answer is negative, even when message contents and guards are not considered. There are composite web services whose conversation set cannot be recognized by finite state automata.

Consider the scenario shown in Figure 3. There are three participants, namely OSB (Online Stock Broker), RD (Research Department), and Investor, involved in a "Fresh Market Update" (FMU) service. We describe each service using a Büchi automaton, and note that each service is equipped with a FIFO queue to store incoming messages under the asynchronous message passing environment like the Internet.

The interaction pattern between the three peers is described as follows. In each round of message exchange, OSB first collects "Rawdata" (e.g. the market price and volume of each stock) from the market, and then sends them to RD for further analysis. After all "Rawdata" are collected and sent, OSB sends the message "EndofRdata" to mark the end of "RawData", and it sends the message "Start" to inform Investor about the planned arrival of a sequence of "Data". RD processes each "Rawdata" and generates a corresponding polished report named "Data". After all "RawData" have been processed, RD sends the message "Complete" to Investor. Once informed by the "Complete" message, Investor sends the message "Ack" to OSB so that OSB can start another round of market information collection and analysis.

The seemingly simple FMU scenario produces a non ω-regular language. To see why this is the case, consider its intersection with an ω-regular language[1] $(\mathbf{R}^*\mathbf{ESD}^*\mathbf{CA})^\omega$. One can infer that the result is $(\mathbf{R}^i\mathbf{ESD}^i\mathbf{CA})^\omega$. By an argument similar to pumping lemma, we can show that this intersection cannot be recognized by any Büchi automaton, and hence the set of conversations is not ω-regular. In fact, given a set of finite state peers, the problem of checking if all conversations generated by them satisfy an LTL property is undecidable due to the unbounded input queues associated with peers. This negative result is one of the motivations for our top-down approach to specification of web services.

[1] We denote each message by its first letter. For example, **R** is the "Rawdata".

7 Discussions

While using the top-down approach enables us to take advantage of model checking techniques, there are other challenges. One possible drawback of the top-down approach may be that for the same design, the global specification can be much larger than its bottom-up counterpart. Another drawback can be that the top-down approach does not work well when we try to compose a service from existing services which do not allow alteration of their internal implementations. In addition, the current version of conversation protocol requires that the participants are fixed, i.e., we cannot dynamically determine the destination of a message, e.g., "check the `url` of the `Req` from Browser, and then send a second request to `Req.url`". We are investigating the trade-off between the top-down and bottom-up approaches to address these challenges.

Automatic verification and validation of composite web services is a new area with interesting challenges — the difficulties arise from both the open system aspect and the hardness of verification problem itself. As we mentioned earlier, to verify the design of U2M example in Figure 1, a model checker with abilities to handle first order constraints is required. We are also looking into the issue of enhancing model checkers with theorem provers to validate a non-trivial composite web service design.

Acknowledgments

Bultan was supported in part by NSF grant CCR-9970976 and NSF Career award CCR-9984822; Fu was partially supported by NSF grant IIS-0101134 and NSF Career award CCR-9984822; Su was also supported in part by NSF grants IIS-0101134 and IIS-9817432.

References

[1] S. Abiteboul, V. Aguilera, S. Ailleret, B. Amann, F. Arambarri, S. Cluet, G. Cobena, G. Corona, G. Ferran, A. Galland, M. Hascoet, C-C. Kanne, B. Koechlin, D. LeNiniven, A. Marian, L. Mignet, G. Moerkotte, B. Nguyen, M. Preda, M-C. Rousset, M. Sebag, J-P. Sirot, P. Veltri, D. Vodislav, F. Watezand, and T. Westmann. A dynamic warehouse for XML data of the Web. *IEEE Data Engineering Bulletin*, 2001.

[2] S. Abiteboul, V. Vianu, B. Fordham, and Y. Yesha. Relational transducers for electronic commerce. In *Proc. ACM Symp. on Principles of Database Systems*, 1998.

[3] Philippe Althern. The scala home page. `http://lamp.epfl.ch/scala/`.

[4] A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, D. McDermott, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the semantic web. In *Proc. Intl. Semantic Web Conf. (ISWC)*, July 2002.

[5] B. Benatallah, B. Medjahed, A. Bouguettaya, A. Elmagarmid, and J. Beard. Self-coordinated and self-traced composite services with dynamic provider selection. Technical report, University of New South Wales, March 2001. (Available at `http://sky.fit.qut.edu.au/~dumas/selfserv.ps.gz`).

[6] Business Process Execution Language for Web Services (Version 1.0). `http://www.ibm.com/developerworks/library/ws-bpel`, 2002.

[7] Business Process Modeling Language (BPML). `http://www.bpmi.org`.

[8] R. Breite, P. Walden, and H. Vanharanta. C-commerce virtuality - will it work in the Internet? In *Proc. of International Conf on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet (SSGRR 2000)*, 2000. (`http://www.ssgrr.it/en/ssgrr2000/proceedings.htm`).

[9] J. R. Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the International Congress on Logic, Methodology, and Philosophy of Science*, pages 1–11. Stanford University Press, 1960.

[10] 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.

[11] T. Bultan and T. Yavuz-Kahveci. Action language verifier. In *Proceedings of the 16th IEEE International Conference on Automated Software Engineering*, pages 382–386, 2001.

[12] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. H. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the 5th Annual IEEE Symposium on Logic in Computer Science*, pages 428–439, January 1990.

[13] C. Bussler, R. Hull, S. McIlraith, M.E. Orlowska, B. Pernici, and J. Yang, editors. *Proceedings of Workshop on Web Services, E-Business, and the Semantic Web (WES).* Springer-Verlag Lecture Notes in Computer Science, number 2512, Toronto, 2002.

[14] F. Casati, S. Sayal, and M. Shan. Developing e-services for composing e-services. In *Proceedings of CAISE 2001*, Interlaken, Switzerland, June 2001.

[15] F. Casati and M.-C. Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–163, 2001.

[16] V. Christophides, R. Hull, G. Karvounarakis, A. Kumar, G. Tong, and M. Xiong. Beyond discrete e-services: Composing session-oriented services in telecommunications. In *Proc. of Workshop on Technologies for E-Services (TES)*, Rome, Italy, September 2001.

[17] G. Cobena, S. Abiteboul, and A. Marian. Detecting changes in xml documents. In *Proc. Int. Conf. on Data Engineering*, 2002.

[18] Electronic Business using eXtensible Markup Language. `http://www.ebxml.org`.

[19] D. Florescu, A. Grünhagen, and D. Kossmann. XL: An XML programming language for web service specification and composition. In *Proc. Int. World Wide Web Conf. (WWW)*, 2002.

[20] X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. Int. Conf. on Implementation and Application of Automata (CIAA)*, 2003.

[21] X. Fu, T. Bultan, and J. Su. Model checking conversation protocols: A top-down approach to specification and verification of web services. *manuscript*, 2003.

[22] S. Gay and M. Hole. Types for correct communication in client-server systems. Technical Report CSD-TR-00-07, Department of Computer Science, Royal Holloway, University of London, December 18 2000.

[23] G. J. Holzmann. The model checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279–295, May 1997.

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

[25] S. A. McIlraith, T. C. Son, and H. Zeng. Semantic web services. In *IEEE Intelligent Systems*, March/April 2001.

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

[27] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. Int. World Wide Web Conf. (WWW)*, 2002.

[28] A. Pnueli. A temporal logic of concurrent programs. *Theoretical Computer Science*, 13(1):45–60, 1981.

[29] Simple Object Access Protocol (SOAP) 1.1. W3C Note 08, May 2000. (`http://www.w3.org/TR/SOAP/`).

[30] Web Service Choreography Interface (WSCI) Version 1.0. `http://www.w3.org/2003/01/wscwg-charter`.