WSAT: A Tool for Formal Analysis of Web Services

Xiang Fu, Tevfik Bultan, and Jianwen Su

Department of Computer Science
University of California
Santa Barbara, CA 93106, USA
{fuxiang,bultan,su}@cs.ucsb.edu

1 Introduction

This paper presents Web Service Analysis Tool (WSAT), a tool for analyzing and verifying composite web service designs, with the state of the art model checking techniques. Web services are *loosely* coupled distributed systems communicating via XML messages. Communication among web services is asynchronous, and it is supported by messaging platforms such as JMS which provide FIFO queues to store incoming messages. Data transmission among web services is standardized via XML, and the specification of web service itself (invocation interface and behavior signature) relies on a stack of XML based standards (e.g. WSDL, BPEL4WS, WSCI and etc.). The characteristics of web services, however, raise several challenges in the application of model checking: **(1)** Numerous competing web service standards, most of which lack formal semantics, complicate the formal specification of web service composition. **(2)** Asynchronous messaging makes most interesting verification problems undecidable, even when XML message contents are abstracted away [3]. **(3)** XML data and expressive XPath based manipulation are not supported by current model checkers.

WSAT, as shown in Fig. 1, tackles these challenges as follows: **(1) An Intermediate Representation:** We use automata with XPath guards (called GFSA) as an intermediate representation for web services. A translator from BPEL4WS to GFSA is developed, and support for other languages can be added without changing the analysis and the verification modules of the tool. **(2) Synchronizability and Realizability Analyses:** We define a set of sufficient *synchronizability* conditions to restrict control flows of a composite web service. When the analysis succeeds, LTL verification can be performed using the synchronous communication semantics instead of asynchronous communication semantics. We also define a set of sufficient *realizability* conditions that are used to synthesize a set of GFSA (called peers) which communicate with asynchronous messages from a single GFSA (called a conversation protocol) which specifies the set of desired global behaviors. The behaviors of the synthesized peers are the same as the behaviors of the conversation protocol if the conversation protocol is realizable [3]. **(3) Handling of XML Data Manipulation:** We developed and implemented algorithms for translating XPath expressions to Promela code [5], and we use model checker SPIN [7] as the back-end of WSAT to check LTL properties.

2 Guarded Finite State Automata

A composite web service can be specified in either bottom-up or top-down fashion. Formally, for a composite web service, its bottom-up specification (called a *web service*

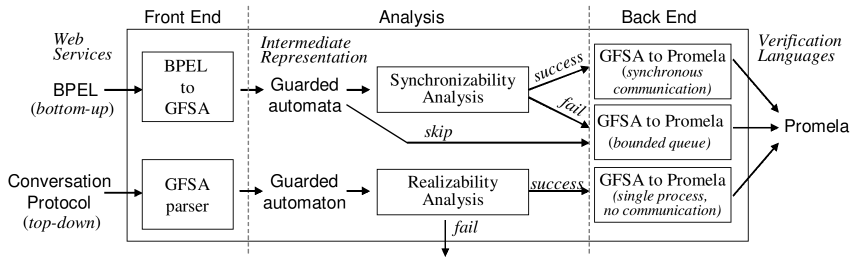


Fig. 1. WSAT architecture

composition) [4] is described as a tuple $\langle (P, M), \mathcal{A}_1, \ldots, \mathcal{A}_n \rangle$, and its top-down specification (called a *conversation protocol*) [2,3,6] is specified as a tuple $\langle (P, M), \mathcal{A} \rangle$. Here (P, M) is the composition schema where P is a set of peer prototypes, and M is a set of message classes. Guarded Finite State Automata (GFSA) $\mathcal{A}_1, \ldots, \mathcal{A}_n$ are the peer implementations (supposing $|P| = n$), and $\mathcal{A}$ specifies the desired set of global behaviors. Below, we present a fragment of the Stock Analysis Service (SAS) conversation protocol studied in [5]:

```
Conversation Protocol{
 Composition Schema{
  PeerList{Inv,SB,RD},
  MSL Type List{
   Register[
    orderID[xsd:int],
    reqList[stockID[xsd:int]{1,10}],
    payment [
     account[xsd:int] |
     creditCard[xsd:int]
    ]], ...
  },
  Message List{
   register{Inv->SB:Register},
   bill{SB->Inv:Bill},...
  }}
```

```
GFSA{
 States{s1,s2,...,s12},
 InitialState {s1},
 FinalStates{s3},
 TransitionRelation{
  t14{s8 -> s12 : bill,
   Guard{
    $request//stockID =
     $register//stockID[position()=last()]
    =>
    $bill//orderID := $register//orderID
   }
  }, ...
 }//end of TransitionRelation
}//end of GFSA
}//end of Conversation Protocol
```

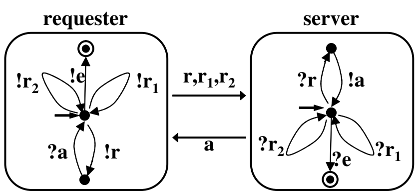
As shown above, each message class has a type defined using MSL [1], a compact theoretical model of XML Schema. WSAT supports a fragment of MSL, where complex types can be constructed using sequence '`,`' (e.g. the `Register`) and choice '`|`' (e.g. the `payment`) operators. An MSL type can also have multiple occurrences (e.g. `payment` can have 1 to 10 `stockID` children), however, maximum occurrence must be bounded.

A GFSA is a tuple (M, T, s, F, Δ). M is the message class set in the composition schema. T, s, F are the set of states, initial state, and the set of final states, respectively. Δ is the transition relation. Each transition $\tau \in \Delta$ is of the form $\tau = (s, (c, g), t)$, where $s, t \in T$ are the source and the destination states of τ, $c \in M$ is a message class and g is the *guard* of the transition. Guards are written using XPath expressions. WSAT supports a subset of XPath which consists of the following operators: child axis (`/`), descendant axis (`//`), self-reference (`.`), parent-reference (`..`), basic type test (`b()`), node name test (t), wildcard (`*`), function calls `position()` and `last()`, and predicates (`[]`). Arithmetic and boolean constraints can be used as predicates in WSAT.

As shown in the SAS protocol, a guard consists of a guard condition and a set of assignments which specify the contents of the message that is being sent. For example the guard of transition `t14` specifies that: if the `stockID` attribute in the `request` message is the last `stockID` in the `register` message, then send out a `bill` message whose `orderID` attribute matches the `orderID` of `register`. The powerful XPath language allows guards to express very rich semantics. In [4] we showed that static BPEL4WS web services can be translated into GFSA representation without loss of data semantics.

3 Synchronizability and Realizability Analyses

Consider the simple client-server web service composition given in the following figure. A requester and a server interact with each other via three request messages and one acknowledgment message. Recall that each peer is equipped with a queue to store incoming messages. It is not hard to infer that the composition has an infinite number of configurations, because the requester can send arbitrary number of r_1 and r_2 messages (which are stored in the queue of the server) before any acknowledgment is sent back. However, another interesting observation is that the global behaviors, characterized by the sequence of messages, is a regular language $(r_1 \mid r_2 \mid ra)^*e$, which is the set of behaviors generated by the composition under synchronous semantics (i.e., the Cartesian product of the two automata). Since in our model [3], LTL properties are defined over the global behaviors, it is decidable to check LTL properties for such services.



We say a web service composition is *synchronizable* if it generates the same set of global behaviors for both synchronous and asynchronous semantics. In [4] we present three sufficient synchronizability conditions to identify synchronizable web service compositions. For each synchronizable web service composition WSAT will generate a Promela specification with synchronous (rendezvous) communication (by limiting the Promela channel size to 0), and then call SPIN to verify LTL properties on the synchronous specification. The verified LTL properties are guaranteed to be satisfied by the original asynchronous web service composition. For top-down specified conversation protocols, we developed a similar analysis called *realizability analysis* [3], which is further extended to consider message contents in [6].

4 Handling XML Data Manipulation

WSAT translates each GFSA into a Promela process. The central issue of the translation algorithm is how to handle XML data and XPath expressions. Each MSL type declaration is translated into a record type (“`typedef`”) in Promela, and types with multiple occurrences are translated into Promela arrays. For example, the `stockID` in the SAS protocol is mapped into a Promela array of size 10. Based on the type mapping, XPath expressions can be translated into Promela code. When MSL types with multiple occurrences are involved, the translation is essentially a nested-loop. For example, consider the following boolean XPath expression: `$reg1//stockID = $reg2//stockID`, where `$reg1` and `$reg2` are two XML variables of type `Register` that is defined in the SAS protocol. Note each side of the equality is a location path which returns a set of `stockID` values. According to XPath standard, the expression evaluates to `true` if we can find one value from each side to satisfy the equality. Hence the expression captures the query: "Is there any `stockID` value which appears in both `$reg1` and `$reg2`?". The corresponding Promela translation is a two-layer nested loop which searches the two arrays (that correspond to the `stockID` array of `$reg1` and `$reg2` respectively), to find a pair of array elements that satisfy the equality. When function calls such as `position()` and `last()` are involved (e.g. the transition guard of `t14` in the SAS protocol), the translation is more complex. The main idea is to substitute the appearance of a function call with an integer variable, and properly update its value so that when the function is called the integer variable contains the right value. More details are available in [5].

We applied WSAT to a range of examples, including six conversation protocols converted from the IBM Conversation Support Project [8], five BPEL4WS services from BPEL4WS standard and Collaxa.com, and the SAS from [5]. Synchronizability and realizability analysis are applied to each example, and except two conversation protocols, all examples pass these checks. This implies that the sufficient conditions in our synchronizability and realizability analysis are not restrictive and they are able to capture most practical applications. For each example, we generated the corresponding Promela specification using WSAT, and we checked LTL properties of the form "$\mathbf{G}(p \rightarrow \mathbf{F}q)$" using SPIN. Our experience with these examples suggests that while exhaustive search of the state space may be very costly for verifying correct properties, SPIN's performance at discovering false LTL properties is satisfactory. For instance, we identified a very delicate design error (a misuse of XPath `position()` function in a transition guard) in the SAS example [5] using SPIN.

WSAT can be extended in the future, by supporting other web service specification languages at the front end, and targeting different verification tools at the back-end. We are especially interested in extending WSAT with symbolic verification techniques in order to handle large state spaces generated by XML data.

Acknowledgments. Authors are supported by NSF Career award CCR-9984822, NSF grant CCR-0341365, IIS-0101134, and IIS-9817432.

References

1. A. Brown, M. Fuchs, J. Robie, and P. Wadler. MSL a model for W3C XML Schema. In *Proc. of 10th Int. World Wide Web Conference (WWW)*, pages 191–200, 2001.
2. T. Bultan, X. Fu, R. Hull, and J. Su. Conversation specification: A new approach to design and analysis of e-service composition. In *Proc. of 12th Int. World Wide Web Conference (WWW)*, pages 403–410, May 2003.
3. X. Fu, T. Bultan, and J. Su. Conversation protocols: A formalism for specification and verification of reactive electronic services. In *Proc. of 8th Int. Conf. on Implementation and Application of Automata (CIAA 2003)*, volume 2759 of *LNCS*, pages 188–200, 2003.
4. X. Fu, T. Bultan, and J. Su. Analysis of interacting BPEL Web Services. To appear in the *Proc. of 13th Int. World Wide Web Conf. (WWW)*, 2004.
5. X. Fu, T. Bultan, and J. Su. Model checking XML manipulating software. To appear in the *Proc. of 2004 IEEE Int. Symp. on Software Testing and Analysis (ISSTA)*, 2004.
6. X. Fu, T. Bultan, and J. Su. Realizability of conversation protocols with message contents. To appear in the *Proc. of 2004 IEEE Int. Conf. on Web Services (ICWS)*, 2004.
7. G. J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*. Addison-Wesley, Boston, Massachusetts, 2003.
8. IBM. Conversation support project. http://www.research.ibm.com/convsupport/.