I.J. Wireless and Microwave Technologies **2012, 6, 66-73**
Published Online December 2012 in MECS (http://www.mecs-press.net)
DOI: 10.5815/ijwmt.2012.06.10

Available online at http://www.mecs-press.net/ijwmt

An Epistemic Model Checking Approach to Web Service Compositions

Xiangyu Luo, Kun Wang, Fengchai Wang

[a] *College of Computer Science & Technology Huaqiao University Xiamen 361021, China*
[b] *School of Computer Science and Engineering Guilin University of Electronic Technology Guilin 541004, China*

Abstract

Due to the dynamics of Web services, the openness and variability of Internet, and the loosely-coupled developing approach of Web services, the development and execution process of Web service compositions becomes uncertain, which imperils the trustworthy properties. In this paper we abstract Web service compositions as multi-agent systems, propose a formal model BSTS for modeling BPEL, develop and implement two translation algorithms B2S and S2I, to translate BPEL into BSTS and translate BSTS into the input language ISPL of the model checker MCMAS for multi-agent systems, respectively. The proposed method supports not only temporal properties, but also epistemic and cooperation properties, which are supported only in multi-agent systems. We implemented the prototype tool, called MCWS, for the proposed method. We modeled and verified an example of Web service compositions. The experimental results show the validity of MCWS.

Index Terms: model checking; Web services; BPEL; epistemic logic; multi-agent systems; strategy

1. Introduction

Web services are the latest distributed technology and, as we will see, the most suitable technology for realization of SOA. They have become the commonly used technology for interoperability and integration of applications and information systems. BPEL [1] is a new language used for composition and coordination of web services. It provides a rich vocabulary for expressing the behavior of business processes.

However, due to the dynamics of Web services and their coordination, the openness and variability of Internet, it is difficult to ensure the reliability, security and usability of the services. As proposed by the W3C consortium: “A Web service is an abstract notion that must be implemented by a concrete agent. The agent is the concrete piece of software or hardware that sends and receives messages.” Therefore, naturally we abstract Web service compositions as multi-agent system and verify not only the temporal specification but also the epistemic and

* Corresponding author.
E-mail address: [1]shiangyuluo@gmail.com,[2]zy_wk@yahoo.com.cn

cooperation specifications by MCMAS [2]. As far as we know, Alessio Lomuscio and Hongyang Qu show how MCMAS can be used to model check temporal-epistemic logic of web service compositions, but they didn't present more elaborate models of coordination, synchronization and communication, and didn't reason about strategies of web service compositions [3].

The rest of the paper is organized as follows. Section II gives the formal model and considers a transformation translates from BPEL to the formal model. In Section III we introduce the algorithm of BPEL to the formal model. Section IV presents the algorithm of the formal model to ISPL. In Section V we discuss a motivating example. Section VI concludes the paper.

2. Modeling with BSTS

A. BSTS

A BSTS (BPEL State Transition System) is a tuple $M = (S, A, V, R, s_0, F)$.

S : Finite set of states.

A : Finite set of actions, including output action o , input action r and internal action i .

V : Finite set of variables

R : Transition relation $S \times A \times S$

$s_0 \in S$: Initial states

$F \in S$: Final states

A transition $t = (n, s_i, v_B, v_L, v_C, p, a, s_{i+1}) \in R$ changes the state from s_i to s_{i+1} with an action $a \in A$ [4]. v_B denotes the set of variables appearing in the source BPEL program. v_L denotes the link variables in the <flow> activity. v_C denotes the condition expressions containing in <pick>, <while>, <if>, the <transitionConditions> in <source> and the <joinConditions> in <target>. p denotes the port name of <portType name>.

B. Basic Activities

<receive> and asynchronous <invoke> modeled as a single state transition represented as $(n, s_i, BPELVariableName, \varnothing, \varnothing, QName, r, s_{i+1})$.

The state transition t_{reply} of <reply> is as follows: $(n, s_i, BPELVariableName, \varnothing, \varnothing, QName, o, s_{i+1})$.

Synchronous <invoke> is modeled two state transitions [5], while the first one is a sending transition, the second one is a receiving transition. The state transitions $t_{sinvoke}$ are represented as

$$(n, s_i, BPELVariableName, \varnothing, \varnothing, QName, o, s_{i+1})\ (n, s_{i+1}, BPELVariableName, \varnothing, \varnothing, QName, r, s_{i+2}).$$

Fig. 1 shows its state transitions.

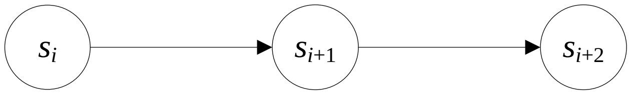

Fig1. State transitions of synchronous<invoke>

<assign> state transition t_{assign} is represented as $(n, s_i, \text{message-two} = \text{message-one}, \varnothing, \varnothing, \varnothing, i, s_{i+1})$.

<empty> is used to model a "no-op" activity of the process. <exit> activity can be used to immediately end the business process instance and ends in a special state s_{end}.

C. Structured Activities

The state transitions $t_{sequence}$ of <sequence> activity are represented as $(\cdots, s_i, \cdots, s_{i+1}) \cdots (\cdots, s_{i+k-1}, \cdots, s_{i+k})$.

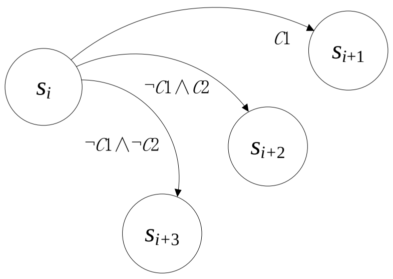

Fig2. State transitions of <if>

The state transitions t_{if} are represented as

$$(n, s_i, \varnothing, \varnothing, C1, \varnothing, i, s_{i+1})$$
$$(n, s_i, \varnothing, \varnothing, \neg C1 \wedge C2, \varnothing, i, s_{i+2})$$
$$(n, s_i, \varnothing, \varnothing, \neg C1 \wedge \neg C2, \varnothing, i, s_{i+3}).$$

As shown in Fig. 2.The dealt of <pick> activity is similar to <if> activity.

<while> state transitions t_{while} are represented as $(n, s_i, \varnothing, \varnothing, C, \varnothing, i, t_i)$ … $(\cdots, t_n, \cdots, s_i)$ $(n, s_i, \varnothing, \varnothing, \neg C, \varnothing, i, s_{i+1})$.

As shown in Fig. 3.

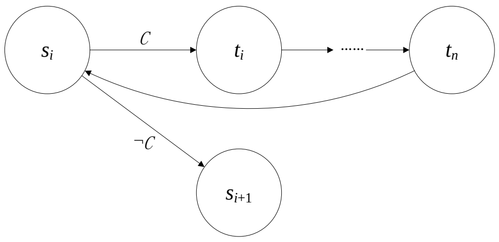

Fig3. State transitions of <while>

In this paper, <flow> is taken as a main state transition system, and the activities that are paralleled are taken as the sub state transition system [6]. The state of the main state transition system is changed from s_i to s_{i+1} by the paralleled action in the main state transition system. The paralleled action in the main state transition system controls each sub state transition system. Through this way each sub activity can be asynchronously performed at the same time. When all the sub activities finished, the state of the main state transition system is changed from s_{i+1} to s_{i+2}.

The state transitions of <target> t_{target} is represented as

$$(n, s_i, \varnothing, v_{L_1} \vee \cdots \vee v_{L_n}, joinCondition, \varnothing, i, s_{i+1})\ (\cdots, s_{i+1}, \cdots, s_{i+2}).$$

The state transitions t_{source} are represented as $(\cdots, s_i, \cdots, s_{i+1})\ (n, s_{i+1}, \varnothing, (l_1, \cdots, l_n), (tc_1, \cdots, tc_n), \varnothing, i, s_{i+2})$.

3. The Algorithm of BPEL to BSTS

A. *Algorithm 1. The B2S Algorithm*

Input: BPEL

Output: BSTS

1) Read BPEL program.
2) If the activity is <receive>, according to part II, produce $t_{receive}$, skip to 1).
3) If the activity is <reply>, according to part II, produce t_{reply}, skip to 1).
4) If the activity is synchronous <invoke>, according to part II, produce $t_{sinvoke}$, skip to 1).
5) If the activity is asynchronous <invoke>, according to part II, produce $t_{ainvoke}$, skip to 1).
6) If the activity is <assign>, according to part II, produce t_{assign}, skip to 1).
7) If the activity is <sequence>, according to part II, produce $t_{sequence}$, skip to 1).
8) If the activity is <pick> or <if>, invoke algorithm 5.
9) If the activity is <while>, invoke algorithm 4, skip to 1).
10) If the activity is <flow>, invoke algorithm 2, skip to 1)
11) End

However, according to DPE's definition, we can't identify two links whose values are false. One is calculated as false by transition condition, another is false because of the DPE [7]. In order to distinguish them we draw into a new symbol #, marking the link is false because of DPE. If the <joinCondition> of an activity is false or #, then this activity's all the links are defined as # and this activity is skipped. When it is true, this activity is performed.

B. *Algorithm 2. The Algorithm of < flow> to BSTS*

Input: < flow>

Output: BSTS of flow activity

1) Define set A which is storing all the links of flow.
2) If the flow activity is not ended, deal with the next activity.
3) Scan the flow activity and mark the whole activities those are paralleled.
4) If the paralleled activities exist, invoke the algorithm3, turn back to 2), or transfer to 5).

5) If the paralleled activity does not exist, sequentially deal with all the target links in activity; if the target link is true and the join condition is true, add a state transition of the target link and turn to 6); if the join condition is false or #, and suppressJoinFailure=yes skip this activity and evaluate all the source link in this activity to # and turn back to 2).

6) Deal with this activity, invoke algorithm 1 and turn to 7)

7) Deal with the source link, if the transition condition is true, add a state transition of the source link is true, if the transition condition is false, add a state transition of the source link is false and turn back to 2).

8) End.

C. Alogrithm 3. The Algorithm of Sub Activities to Sub BSTS

Input: all the paralleled activities in <flow>

Output: the main state transition system and the sub state transition systems

1) Add the main state transition.

2) Search paralleled activity, if exist, skip to 3), if not, skip to 4).

3) If there are some paralleled activities, handle all the paralleled activities at present, invoke algorithm 1 to generate the sub state transition system and skip to 2).

4) After all the paralleled activities are done, add a main state transition to the main state transition system.

5) End.

D. Alogrithm 4. The Algorithm of <while> to BSTS

Input: <while>

Output: BSTS of <while>

1) When come across the keyword while, two sign InL and OutL are introduced.

2) Calculate the condition expression of while activity and generate a state transition and use the sign InL to mark the entrance.

3) If the condition is true, perform the activities in the circle, invoke algorithm1, if the condition is false, skip to OutL, transfer to 5).

4) If met with the end sign of the while activity and use the sign Out to mark the exit and turn back to 2).

5) Dap the circle and generate a state transition.

6) End.

E. Alogrithm 5. The Algorithm of <if> to BSTS

Input: <if>

Output: BSTS of if activity

1) Produce the state transition of condition expression.

2) If the condition is true, transfer to algorithm 1.

3) If the condition is false and the else if condition is true, generate a state transition of condition expression, invoke algorithm 1.

4) Calculate all the branches of <if>.

5) End.

4. The Algotithm of BSTS to ISPL

The process is seen as an agent, while the <partnerLinks> as the other agent. When the agents interact with each other, it needs to insert an environment agent to model channels in the systems. We take <portType> as environment for the agents.

A. *Algorithm 6. The B2I Algorithm*

Input: BSTS

Output: ISPL

1) Read the next state transition, if it is non-empty, then skip to 2), if it is null, then skip to 14).

2) If the action in the state transition is a sending action, then skip to 3). If the action in the state transition is a receiving action, then skip to 6). If the action in the state transition is a inner action, then skip to 9). Break.

3) Add states s_i , s_{i+1} and variables v_B, v_L, v_C to the list of state in the agent process. Add action a to the list of action. Add s_i :{ a } to the list of protocol. Add state=s_{i+1} and v_B if state=s_i and Action=a and v_L and v_C to the list of evolution.

4) Add states p_i , p_{i+1} and variable v_B to the list of state in the agent n . Add action $none$ to the list of action. Add p_i :{ $none$ } to the list of protocol. Add state= p_{i+1} and v_B if state= p_i and Process.Action=a and Environment.Action= p to the list of evolution

5) Add state p and state sleep to the list of state of the agent environment. Add action p to the list of action. Add sleep:{ p } into the list of protocol. Add state= p if Action= p into the list of evolution, and skip to 1).

6) Add states s_i , s_{i+1} and variable v_B, v_L, v_C to the list of state in the agent process. Add action $none$ to the list of action. Add s_i :{ $none$ } into the list of protocol. Add state=s_{i+1} and v_B if state= s_i and n.Action= a and Environment.Action= p to the list of evolution.

7) Add states p_i , p_{i+1} and variable v_B to the list of state in the agent n . Add action a to the list of action. Add p_i :{ a } to the list of protocol. Add state= p_{i+1} if state= p_i and action=a to the list of evolution.

8) Add state p and state sleep to the list of state of the agent environment. Add action p to the list of action. Add sleep :{ p } into the list of protocol. Add state= p if Action= p into the list of evolution, and skip to 1).

9) If the action is the sub paralleled activity of the flow, turn to 10).If the activity is assign activity or internal activity, turn to 13).

10) Add states s_i , s_{i+1} to the list of state in the agent process. Add action a to the list of action. Add s_i :{ a } to the list of protocol. Add state= s_{i+1} if state= s_i and Action= a to the list of evolution.

11) Generate the sub state transition system of all the paralleled activities. Add Process.Action= a to all the lists of evolutions of the sub state transition systems.

12) Add states s_i , s_{i+1} to the list of state in the agent process. Add action $none$ to the list of action. Add s_i :{ $none$ } into the list of protocol. Add state= s_{i+2} if state= s_{i+1} and sub agent.Action=end to the list of evolution. Turn to 1).

13) Add states s_i , s_{i+1} and variables v_B, v_L, v_C to the list of state in the agent process. Add action a to the list of action. Add s_i :{ a } to the list of protocol. Add state= s_{i+1} and v_B if state= s_i and Action= a and v_L and v_C to the list of evolution. Turn to 1).

14) End.

5. Model Checking The Loan Approval Service

We used MCWS to verify the Loan Approval Service [1]. The service involves five agents: Customer, Assessor, Approver, LoanApprovalProcess and Environment. In the following are parts of ISPL:

```
Agent loanApprovalProcess
state:{sleep, rec_req_ Less10000, rec_r_low, rec_r_h,...};
Action={s_req_Less10000,ass_r_low_app_yes, …};
Protocol:
state=sleep:{none};
state=rec_req_Less10000:{s_req_Less10000};
state=rec_req_Less10000 if (state=sleep) and (customer.Action= s_req_Less10000) and
(Environment.Action= cus_to_loanAppProcess);
```

We had tested the following specifications:

CTL formulae: AF amountGreater10000 –> AF recapproval_no. It stands for eventually in all paths, if the loaning amount is more than ten thousand dollars, it would fail. The formulae is FALSE, because the loaning amount is more than ten thousand dollars, the approver may approve the request.

Epistemic property: AF (recapproval_yes -> ((amountGreater10000 -> K (Customer, K (Approver, approver_risk_low))) or (amountLess10000 -> K (Customer, K (Assessor,assessor_risk_low))))). It stands for, if the loan request is approved, then the customer knows the fact that when the loaning amount is more than ten thousand dollars, the risk of the request is low, or when the loaning amount is less than ten thousand dollars, the risk of the request is low. This property is TRUE.

Strategy formulae: <g1> G (<g1> F recapproval). It stand for finally, whether the request is success or failure, under the cooperation of multiple agents, there is a strategy for the customer receives the reply of the LoanApprovalProcess agent. The result is TRUE. To verify the strategy formulae visually reflects the property of Web services compositions.

6. Conclusion

In this paper we show that MCWS can automatically verify temporal, epistemic properties and cooperation which are important to the multi-agent of Web service compositions. The proposed method takes into account such DPE and the link. As part of our future work we intend to verify the security aspect.

Acknowledgment

This work is supported by the National Natural Science Foundation of China (No.60763004), the Fundamental Research Funds for the Central Universities (No. JB-GJ1001), and the Scientific Research Foundation of Huaqiao University (No.11BS108)

References

[1] OASIS Standard. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html. 2007.

[2] Franco Raimondi. “Model checking multi-agent systems”. Phd, University College London, 2006. pp. 112-127

Alessio Lomuscio, Hongyang Qu, Monika Solanki. “Towards verifying contract regulated service composition”. ICWS’08. Beijing. 2008. pp.254-261.

[3] Alessio Lomuscio, Hongyang Qu, Marek sergot, Monika Solanki. “Verifying temporal and epistemic properties of web service composition”. Lecture Notes in Computer Science, Vol. 4, Springer, 2007. pp. 456-461.

[4] Raman Kazhamiakin. “Formal analysis of web service composition”. Phd, University of Trento. 2007. pp. 48-71

[5] Andreas Wombacher, Peter Fankhauser, Erich Neuhold. “Transforming BPEL into annotated deterministic finite state automata for service discovery”. ICWS’04. California. 2004, pp. 316-323.

[6] S.Nakajima. “Model-Checking Behavioral Specification of BPEL Applications”. Electronic Notes in Theoretical Computer Science. 2006. pp. 89-105.

[7] Franck van Breugel and Mariya Koshkina. “Dead-path-elimination in BPEL4WS”. ACSD’05. France. 2005. pp. 192-201.