Available online at www.sciencedirect.com
ScienceDirect
Procedia Computer Science 52 (2015) 278 – 285
Procedia Computer Science

6th International Conference on Ambient Systems, Networks and Technologies (ANT 2015)

Set partition and trace based verification of Web service composition

Gopal N. Rai[a,b], G. R. Gangadharan[a,*]

[a]IDRBT, Masab Tank, Hyderabad-500057, India
[b]University of Hyderabad, Gachibowli, Hyderabad-500046, India

Abstract

Designing and running Web services compositions are error-prone as it is difficult to determine the behavior of web services during execution and their conformance to functional requirements. Interaction among composite Web services may cause concurrency related issues. In this paper, we present a formal model for reasoning and verifying Web services composition at design level. We partition the candidate services being considered for composition into several subsets on the basis of their service invocation order. We arrange these subsets to form a Web services set partition graph and transform to a set of interacting traces. Then, we propose a novel methodology for service interaction verification that uses service description (from WSDL file) to extract the necessary information and facilitates the process of modeling, analyzing, and reasoning the composite services. As a part of verification technique, we use two levels of modeling. This includes abstract modeling that further leads to detailed modeling if required, thereby reducing the computation time and modeling complexity.

Keywords: Web service composition; Web service interaction; formal verification; modeling

1. Introduction

Service composition is perceived as the federation of a service with other remote services, specifying the participating services, the invocation sequence of services and the methods for handling exceptions[1]. Designing and running Web services compositions are error-prone because a single service may have several dependencies with other services to perform their tasks correctly and the developers may not know the identity of those services that would fulfill the request. Analogous to other distributed systems based on asynchronous communication, it is difficult to anticipate how Web service compositions behave during execution and whether they conform to the functional requirements[2]. Interaction among composite Web services through messages opens the space for concurrency related issues[3]. These concurrency bugs are difficult to be found out by testing, since they tend to be non-reproducible or are not covered by test cases. Furthermore, the asynchronous nature of communications complicates the scenario and it becomes very difficult to reason about and debug. Existing approaches including model checking, Petri net, π calculus, genetic

* Corresponding author. Tel.: +91-40-2329-4184; fax: +91-40-2353-5157.
E-mail address: geeyaar@gmail.com

algorithms, Temporal Logic of Actions (TLA), Alternating-time Temporal Logic based reasoning, and case-based reasoning achieve the aspect of interaction verification by modeling, planning, and verifying the Web service composition. However, these approaches have their disadvantages like involvement of intermediate languages, inability to capture recursive composition, and higher space and time complexity.

In this paper, we present a formal model for reasoning and verifying dynamic Web services composition at design level. Our aim is to provide a methodology that takes a set of candidate Web services, their respective WSDL files, and their interaction specifications as input and provides the output whether an interaction specification is satisfied or not. Furthermore, if the interaction specification is not satisfied, then the counterexample is produced. The salient contributions of our paper are as follows:

- A novel concept of Web services set partitioning technique that makes a set of candidate Web services being considered for a composition scenario into a number of subsets based on service invocation possibility and thereby generating a Web services set partition (WSSP) graph.
- Formal definitions of trace related terminologies for modeling and reasoning interactions of a Web service composition
- A novel methodology for service interaction verification using WSSP and trace concepts. As a part of verification technique, we use abstract modeling in the beginning that further leads to detailed modeling if required. This technique reduces the computation time and modeling complexity.

The rest of the paper is organized as follows: Section 2 discusses various approaches for verification of Web services composition. In section 3, we propose Web services set partitioning concept. Section 4 describes trace related definitions related to Web service composition. Our proposed service interaction verification methodology is presented in Section 5. Section 6 illustrates the evaluation of Web service interaction verification using an example, followed by concluding remarks in Section 7.

2. Related work

Foster et al.[4] present a model-based verification approach to verify Web service composition. The approach supports verification of the specification models to confirm the expected results for both the designer and the implementer. The specifications of the design are modeled in UML in the form of message sequence charts and mechanically compiled into the finite state process notation to concisely describe and reason about the concurrent programs.

Fu et al.[5] present a set of tools and techniques for analyzing the interactions of composite Web services that are specified in BPEL and communicate through asynchronous XML messages. They present a framework where BPEL specifications of Web services are translated to an intermediate representation, followed by the translation of the intermediate representation to a verification language.

Zheng et al.[6] propose Web Service Automaton (WSA) that transforms BPEL into the input language (Promela or SMV) for a model-checker (SPIN or NuSMV). This approach generates test cases using counterexamples to perform conformance tests on BPEL and using WSDL to test Web service operations. Zhu et al.[7] propose an effective approach to describe and compose semantic Web services using UML. However UML cannot be used for verifying the correctness of Web service composition.

Separation of operational and control behaviors of Web services makes better service design and verification. This practice is adopted in[8]. Sheng et al.[9] propose a Web service model for the dependable development of Web services. In this model, service behaviors are separated into operational behavior and control behavior and coordination between them is facilitated or achieved through conversational messages. Elkohly et al.[10] propose extended branching-time temporal logic with temporal modalities to specify commitments in Web services interaction and their fulfillments.

A considerable amount of work based on modeling and analyzing Web services using Petri net has been already done[11,12,13]. Schlingloff et al.[14] combine Petri net and model checking for modeling and verification of a composite service. The authors illustrate how to model Web services with Petri net and study the automated verification according to abstract correctness criterion. Further, they relate the correctness of Web service models to the model checking problem for alternating temporal logics.

In comparison to earlier works, our approach is better in the following aspects: It does not employ any intermediate language. Faulty Web services and isolated Web services can be easily detected. The set based mathematical articulation of Web service composition enhances the process of automation of service interaction verification.

3. Web services set partition (WSSP)

Let $\mathcal{W} = \{w_1, \cdots, w_m\}$ be a finite set of Web services being considered for a composition scenario. Throughout the paper, we consider $\mathcal{W}$ in the same meaning unless stated otherwise.

Definition 3.1 (Service invocation possibility set (SIPS))**.** Let '$\mapsto$' be a symbol to represent service invocation possibility. ($w_i \mapsto S_x = \{w_j, w_k\}$ means that w_i can directly invoke w_j and w_k.) Given a Web service $w_i \in \mathcal{W}$, the service invocation possibility set for the service w_i with respect to $\mathcal{W}$ is a set represented by $SIPS(w_i)$ such that $SIPS(w_i) \in 2^{\mathcal{W}}$ and $w_i \mapsto SIPS(w_i)$.

Definition 3.2 (Service invocation chain igniter)**.** A Web service $w_i \in \mathcal{W}$ is called as a service invocation chain igniter with respect to $\mathcal{W}$ if and only if $SIPS(w_i) \neq \emptyset$ and $\nexists w_j \in \mathcal{W} : w_i \in SIPS(w_j)$.

Definition 3.3 (Isolated Web service)**.** A Web service $w_i \in \mathcal{W}$ is called as an isolated service with respect to $\mathcal{W}$ if and only if $SIPS(w_i) = \emptyset$ and $\nexists w_j \in \mathcal{W} : w_i \in SIPS(w_j)$.

Algorithm 1 Web Services Set Partition (WSSP)

```
    Input: $\mathcal{W} = \{w_1, w_2, w_3, \cdots, w_m\}$
    Output: $\mathcal{S} = \{S_1, S_2, S_3, \cdots, S_n\}$
 1: let $\mathcal{S} = \{S_1 \leftarrow \emptyset, \cdots, S_n \leftarrow \emptyset\}$, $n = 2^{|\mathcal{W}|} - 1$ be a set
 2: for all $w_i \in \mathcal{W}$ do
 3:     if $SIPS(w_i) \neq \emptyset$ and $\nexists w_j \in \mathcal{W} : w_i \in SIPS(w_j)$ then
 4:         $S_1 \leftarrow w_i$
 5:     end if
 6: end for
 7: for all $w_i \in S_1$ do
 8:     $Temp \leftarrow \mathcal{W}$
 9:     $Temp \leftarrow Temp \backslash S_1$
10:     let $\mathcal{P} = \{P_1\}$ be a set
11:     $n \leftarrow 1$
12:     while ($P_n \neq \emptyset$) do
13:         create a set $P_{n+1} \leftarrow \emptyset$
14:         $\mathcal{P} \leftarrow P_{n+1}$
15:         for all $w_i \in P_n$ do
16:             for all $w_j \in Temp$ do
17:                 if $w_i \mapsto w_j$ then
18:                     $P_{n+1} \leftarrow w_j$
19:                 end if
20:             end for
21:         end for
22:         $Temp \leftarrow Temp \backslash P_n$
23:         $S_n \leftarrow S_n \cup P_n$
24:         $n \leftarrow n + 1$
25:     end while
26: end for
27: for all $S_i \in \mathcal{S}$ do
28:     if $S_i = \emptyset$ then
29:         $\mathcal{S} \leftarrow \mathcal{S} \backslash S_i$
30:     end if
31: end for
```

To analyze the set $\mathcal{W}$, we partition it into n number of subsets $\mathcal{S} = \{S_1, \cdots, S_n\}$ using algorithm 1, where $n < 2^{|\mathcal{W}|}$ and $\mathcal{S} \subset 2^{\mathcal{W}}$ such that following properties hold

- **P1**. $\forall S_i \in \mathcal{S} : S_i \neq \emptyset$ (No partition set is empty.)
- **P2**. $\forall w_i \in S_1 : \{(SIPS(w_i) \neq \emptyset) \wedge (\nexists w_j \in \mathcal{W} : w_j \mapsto w_i)\}$ (Partition set S_1 is the set of igniter Web services.)
- **P3**. $|S_1| = 1 \rightarrow \left(\forall S_i, S_j \in \mathcal{S} : S_i \neq S_j \rightarrow S_i \cap S_j = \emptyset\right)$. (Partition sets with respect to a single igniter are pairwise disjoint.)
- **P4**. $S_n \subseteq \bigcup_{\forall w_i \in S_{n-1}} SIPS(w_i)$, where $n > 1$. (Each successor partition set (except than the first set S_1) $S_n \in \mathcal{S}$ is subset or equal to union of service invocation possibility sets for each element of the predecessor partition set.)

- **P5**. $|\bigcup_{i=1}^{n} S_i| < |\mathcal{W}| \Leftrightarrow \exists w_i \in \mathcal{W} : w_i \notin S_j, \text{where} 1 \leq j \leq n$. (Non exhaustive partition of $\mathcal{W}$ implies existence of isolated services in $\mathcal{W}$. A service that does not participate in partition is isolated service.)

3.1. Web services set partition graph (WSSP graph)

A WSSP graph represents all the subsets and service invocation possibilities in a partitioned Web services set resulting from the WSSP algorithm. In WSSP graphs, each $S_i \in \mathcal{S}$ behave as multi-element nodes and we interpret the service invocation possibility into directed edges from a Web service to another Web service. Two types of directed edges are being used for this graph: dashed arrow and solid arrow. A dashed arrow infers that a particular service that is on the arrow-head side is available on this chain but cannot directly be invoked by an arrow-tail side service. A solid arrow infers that the service on the arrow-head side can be invoked directly by the service on the arrow-tail side. Fig.1 is an example of WSSP graph. In Fig.1, two order subscripts are used to name the Web services (for ease of representation). WSSP graph can be perceived in two different manners as per requirement: consolidated view and distinguished view.

Consolidated view. A consolidated view is an overall view with all igniters being placed in the first set S_1. Fig.1 depicts a consolidated view for a Web service composition scenario. All the services that can be invoked by any service in the subset S_1 will be in the second subset with their respective directed edges and so on. A service could be repeated in several subsets. It is also possible that a service could invoke a service from its own set. Even though a service is invoking a service from its own set, an arrow must not be there within the subset.

Distinguished view. A distinguished view is an extracted view of a consolidated view. For this, we perceive the whole scenario from the point of view of a specific igniter from the first set. Fig. 2 is a typical depiction of a distinguished view with w_{1j} as an igniter. A distinguished view always forms a tree with an igniter as a root node and a consolidated view may or may not form a tree.

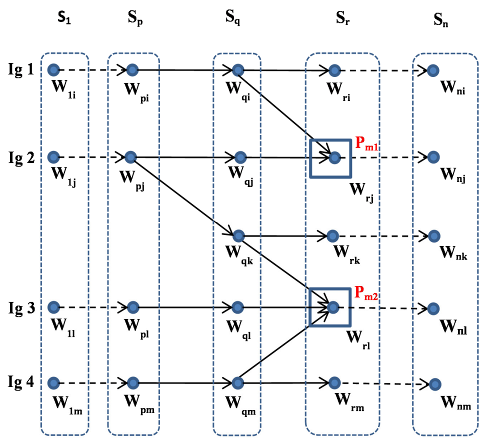

Fig. 1: Consolidated view of a Web service composition scenario

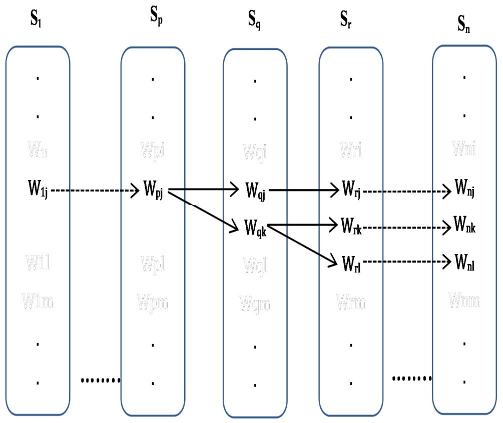

Fig. 2: Distinguished View (w_{1j} as Composition Chain Igniter)

4. Trace inclusion and merging

A trace is a unidirectional tree $\langle V, E\rangle$ where vertices represent Web services, edges represent service invocation possibilities, and arrow head shows the direction of workflow provided that each vertex is connected with exactly two edges (one is input and another is output) except first (root) and last (leaf) vertices. In other words, a trace is a linear Web services composition workflow path.

Trace inclusion. Trace inclusion refers to the containment of a trace by another trace(s). There are two classes of trace inclusion as follows:
Total trace inclusion is a condition where the root of a trace (for example w_i) is present in a trace that is generated from another root (for example w_j). In this way, w_j contains all the traces generated by w_i. By computing total trace

inclusion, we avoid redundant traces in searching for or performing some actions over traces.
Given two or more traces generated by different igniters, a *partial trace inclusion* occurs if a trace-fraction is found common in all traces. Trace fraction from w_{rl} to w_{nl} is common in all trees generated by igniters Ig2, Ig3, and Ig4 (Fig.1). By computing partial trace inclusion, the best-time complexity of a verification process could be improved. The occurrence of a trace inclusion is possible only in a consolidated view because of the existence of more than one igniter.

Trace merging. Trace merging is a condition where two or more traces originated from different igniters conjunct at a point. Trace merging can be classified into two classes: total merging and partial merging. For an igniter, a *total merging* situation occurs when all the traces originated from an igniter get merged at a point that lies on the trace generated from another igniter. In Fig.1, for an igniter Ig3, P_{m2} is a total merging point as the only trace originated from Ig3 is merged here.
For an igniter, a *partial merging* situation occurs if some traces get merged but not all. In Fig.1, for Ig1 and Ig2, P_{m1} is a merging point where each individual traces from both igniters conjunct. These are not the only traces from Ig1 and Ig2. Besides these traces, there are some other traces which do not get merged. Merging points are very critical points and play an important role in our proposed service interaction verification methodology.

5. Service interaction verification and deadlock avoidance

The process of service interaction verification includes modeling of the system, writing the specification, and applying the verification technique. In our approach, we follow a model-based approach for verifying a system. In a model-based approach, the system is represented by a model M for an appropriate logic. The specification is represented by a formula ϕ and the verification method consists of computing whether a model M satisfies ϕ ($M \models \phi$).

Modeling the system. Given a set $\mathcal{W}$ of candidate Web services being considered for composition, the Web Services Set Partition algorithm in section 3 provides the partitioned subsets of Web services set $\mathcal{W}$ on the basis of the service invocation order of the candidate Web services. The WSSP graph satisfying disjointness and orthogonality results in a WSSP graph representing an abstract model of the system. The abstract model works as a base model against which the interaction specification has to be verified. This abstract model captures an abstract view of the system. In the abstract model, the activities of Web services are not considered.
To model subtleties regarding the system, we are extracting two types of information from a given WSDL file: activities within Web services and communication pattern. The abstract model becomes more detailed after specifying these sets of information. **Interaction specification**. Interaction specifications are the properties about interaction among Web services written formally that need to be verified against the intended model. We formalize the interaction specifications using temporal logic. We select Linear Temporal Logic (LTL)[15] and Computational Tree Logic (CTL)[15] to write specifications and interpret their meaning over the model in the process of verification. We write the interaction specifications in terms of the activities of Web services.

Verification technique. Our approach differ from classical model-based scheme at modeling phase and at verification phase. Specification writing phase is same as in classical scheme. The verification process begins with the analysis of specification formulae. Consistency among specifications are checked as explained in[15]. Algorithm 2 presents our service interaction verification methodology.

In LTL model checking, before applying a verification technique, LTL properties must be negated. Here we do not negate the property and do not perform any kind of product between property specification and model specification because we check all possible traces explicitly.
We consider the interaction specification formulae written in LTL/CTL from the set of specifications Φ and extract the activities of Web services. After the extraction of activities, we map the activities to their respective owner Web services and inflate Web services with its activities. Use of the concept of inflation and deflation in verification technique reduces the chances of state explosion. However, in worst case, if the activities from all the candidate Web services are involved in a single interaction specification formula, the deflation is not useful. We deflate the services if the services are not in use. After inflation, we explore all the possible communication patterns among the involved Web services. To explore the possibilities, we consider one of the following four communication types: one-way, solicit-response, request-response, and notification. The set of constraints based on the model reduces the number of possible communication patterns.

Algorithm 2 Service Interaction Verification

```
Input: $\mathcal{W}$, $\mathcal{S}$, WSDL files, abstract model $\mathbb{M}$, specification formulae set $\Phi = \{\phi_1, \cdots, \phi_p\}$
Output: yes or no with counter trace

 1: $ACT(w_i)$ : Set of all activities of $w_i$
 2: $ATOM(\phi_i)$ : Set of all atoms in $\phi_i$
 3: for all $\phi_i \in \Phi$ do
 4:     let $W_{\phi_i} \leftarrow \emptyset$ be a set of Web services
 5:     for all $w_j \in \mathcal{W}$ do
 6:         if $ACT(w_j) \cap ATOM(\phi_i) \neq \emptyset$ then
 7:             $W_{\phi_i} \leftarrow w_j$
 8:         end if
 9:     end for
10:     $\mathcal{T}_{(w_j)}$ : Set of traces generated by igniter $w_i$
11:     $D(w_i)$ : Set of all services in distinguished
              view of igniter $w_i$
12:     $S_1 \in \mathcal{S}$ : Set of igniters
13:     for all $w_j \in S_1$ do
14:         if $W_{\phi_i} \subseteq D(w_j)$ then
15:             if any trace-constraint exists then
16:                 apply over $\mathcal{T}_{(w_j)}$
17:             end if
18:             $FLAG = FALSE$
19:             for all $\mathcal{T}_i \in \mathcal{T}_{(w_i)}$ do
20:                 if $\mathcal{T}_i \not\models \phi_i$ then
21:                     $\mathcal{T}_i$ is a counter example
22:                     $FLAG \leftarrow TRUE$
23:                 end if
24:             end for
25:             if FLAG=FALSE then
26:                 $d(w_i) \models \phi_i$
27:             end if
28:         else $\phi_i$ is not relevant
29:         end if
30:     end for
31:     deflate the services
32: end for
```

We match each trace generated by with the intended interaction specification formulae. If there is a match, then the property (interaction specification) is satisfied within the trace, otherwise violation exists. A trace which does not follow the property specification is considered as a counter example. With the help of a counter example, the designer corrects the model. We use trace concepts to optimize the computation process and search space. This technique behaves in a correct manner for any number of involved services.

Theorem 5.1. *A WSSP graph $\mathcal{G}$ for the set $\mathcal{W}$ of Web services, satisfying the properties P1-P5 cannot lead to a communication deadlock.*

Proof. Assume, to the contrary, that $\mathcal{G} \models (P1 - P5)$ and deadlock is possible in $\mathcal{G}$.
Let invocation orders $w_i \mapsto w_j,\ w_j \mapsto w_k,$ and $w_k \mapsto w_i$ forms a deadlock cycle in $\mathcal{G}$. Since we have considered that $w_i \mapsto w_j$ and $w_k \mapsto w_i$, w_i works as a non-igniter service. Let w_{ig} be an igniter such that w_i lies on a trace generated by w_{ig}. This implies that $w_j,\ w_k,$ and w_l fall on same trace as we have assumed $w_i \mapsto w_j,\ w_j \mapsto w_k,$ and $w_k \mapsto w_i$. Let $S_1, \cdots, S_n$ be partition sets with respect to igniter w_{ig}. Let $w_i \in S_i,\ SIPS(S_i) = S_j,\ SIPS(S_j) = S_k,$ and $SIPS(S_k) = S_l$. This implies that $w_i \in S_l$. But, we have already assumed that $w_i \in S_i$. This implies $S_i \cap S_l \neq 0$. This result contradicts the property P3. This contradiction establishes the proof. □

6. Evaluating our service interaction verification approach

We implemented a *travel agency* scenario using Eclipse IDE for Java, XAMPP Apache Tomcat server, and MySql database. Experiments were performed under Windows 8 64-bit operating system with Intel(R) Core(TM) i5 2.6 GHz processor and 8GB RAM.

We present a detailed underlying description how our travel agency scenario implementation and verification works. A customer, who wants to plan her travel, invokes the travel agency Web service at first. Then the travel agency Web service invokes three Web services: car rental (CR), hotel booking (HB), and flight booking (FB) to book the respective services.

Modeling. As discussed previously, we have four Web services $\mathcal{W}$ = TA, CR, HB, and FB. By applying algorithm 1 we have $\mathcal{S} = \{S_1, S_2\}$ where $S_1 = \{TA\}$ and $S_2 = \{CR, HB, FB\}$. The WSSP graph for the set $\mathcal{S}$ represents a distinguished view with TA as an igniter service. As this graph satisfies disjointness and orthogonality, no deadlock is possible.

TA conveys two types of messages. The purpose of first type of messages ($Car\_Avail?$, $Hotel\_Avail?$, and $Flight\_Avail?$) is to get information regarding availability. Recipient Web services (CR, HB and FB) reply Car_Yes, $Hotel\_Yes$, and $Flight\_Yes$ respectively if availability is there. Otherwise Car_No, $Hotel\_No$, and $Flight\_No$ results in response to the availability inquiry. After getting an affirmative answer regarding availability, TA requests for booking by sending the second type of messages (Car_Bk, $Hotel\_Bk$, and $Flight\_Bk$). On receiving a booking request message from TA, the recipient services reply as Car_Bkd, $Hotel\_Bkd$, and $Flight\_Bkd$ if booking is done. Otherwise, the recipient services reply $\neg Car\_Bkd$, $\neg Hotel\_Bkd$, and $\neg Flight\_Bkd$ respectively. There may be several reasons for failure of booking including technical errors.

Specification. Let $\rightarrow$ be the logical implication. For writing specifications, we use the activities of Web services as propositions. For this example, we consider following two interaction specification properties as LTL formulae

- $\phi_1 = G((Hotel\_Yes \wedge Hotel\_Bk) \rightarrow F(Hotel\_Bkd))$. This property states that *if hotel is available and TA request to book the hotel then eventually hotel booking must be done*.
- $\phi_2 = G((Hotel\_Bkd) \rightarrow F(Flight\_Bkd))$. This property states that *if hotel is booked then eventually flight also must be booked* (without flight booking there is no meaning of hotel booking).

For the above mentioned specifications, we do not have any inconsistencies.

Verification. At first, we consider the first LTL specification: $\phi_1 = G((Hotel\_Yes \wedge Hotel\_Bk) \rightarrow F(Hotel\_Bkd))$ to verify against the model. By extraction, we find that the activities $Hotel\_Yes$, $Hotel\_Bk$, and $Hotel\_Bkd$ constitute the specification ϕ_1. These activities belong to HB, TA, and HB respectively. Therefore, we inflate these services in the abstract model. Since we do not have any activities from car rental service and flight booking service in specification formulas, we do not inflate these services with activities.
We establish all communication patterns (shown in Fig.3) between TA and HB as the activities belong to these two services. We find three traces between TA and HB. We verify each trace one by one and find out that the property ϕ_1 gets violated in the following trace: $TA \rightarrow Hotel\_Avail? \rightarrow Hotel\_Yes \rightarrow Hotel\_BK \rightarrow \neg Hotel\_Bkd \rightarrow HB$.

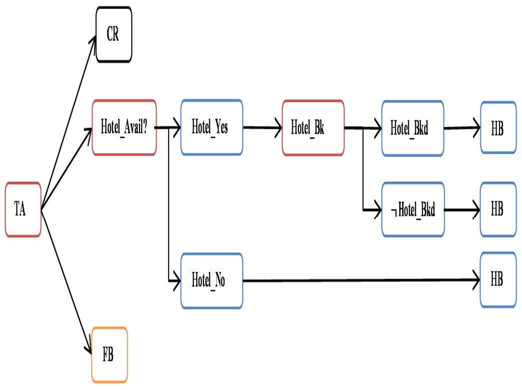

Fig. 3: All possible communication patterns between TA and HB

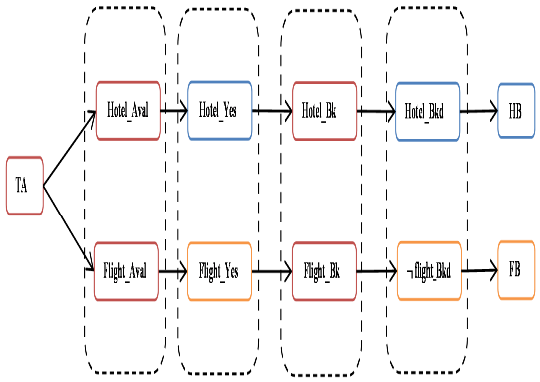

Fig. 4: Communication pattern where ϕ_2 gets violated

Now, we consider the second LTL specification: $\phi_2 = G((Hotel\_Bkd) \rightarrow F(Flight\_Bkd))$. By analyzing the said specification formula, we extract two activities as follows: $Hotel\_Bkd$ and $Flight\_Bkd$. These activities belong to HB and FB. Activities involved in this specification belong to parallel traces. However, both traces are being generated by the same igniter. One activity belongs to the trace from TA to FB and the other activity belongs to the trace from TA to

HB. Two traces alternative to each other cannot run at the same time while two parallel traces can run simultaneously. For example, there are three traces from TA to HB. All are alternative to each other. Traces from TA to FB are parallel to traces from TA to HB. We consider one-one relevant traces from TA to FB, TA to HB at a time and we find that property ϕ_2 is not satisfied. Fig.4 shows a communication pattern where specification ϕ_2 gets violated.

7. Concluding remarks

In this paper we worked towards a formal verification methodology for Web services composition. We partition the candidate Web services being considered for composition into several subsets on the basis of service invocation order using web service partitioning algorithm. Arranging these subsets in a specific fashion results in a WSSP graph that represents the abstract model of the system. Further, we transform this model into a set of interacting traces that provides a strong formal basis to reason about the anticipated interaction specifications (properties) that a system supposed to be have. We presented a service interaction verification methodology, that uses service description (from a WSDL file) to extract the necessary information and facilitates the process of modeling, analyzing and reasoning composite services. Using a WSSP graph satisfying disjointness and orthogonality, deadlocks are detected and resolved in Web services composition. Misbehaving Web services workflow can also be investigated with the help of our proposed approach. Further, the distributed execution of composition pattern can be verified easily and isolated services could be identified easily.

In our future work, we plan to verify the crucial properties like atomicity and recovery in service interactions. Bisimulation study is useful when we wish to determine whether a service can substitute another service during composition. We plan to extend our service interaction verification methodology with bisimulation and redundancy checking.

References

1. Alonso, G., Casati, F., Kuno, H.A., Machiraju, V.. *Web Services - Concepts, Architectures and Applications*. Data-Centric Systems and Applications. Springer; 2004.
2. Röglinger, M.. Verification of web service compositions: An operationalization of correctness and a requirements framework for service-oriented modeling techniques. *Business & Information Systems Engineering* 2009;**1**(6):429–437.
3. Betin-Can, A., Bultan, T., Fu, X.. Design for verification for asynchronously communicating web services. In: *Proceedings of the 14th international conference on World Wide Web*. ACM; 2005, p. 750–759.
4. Foster, H., Uchitel, S., Magee, J., Kramer, J.. Model-based verification of web service compositions. In: *Proceedings of the 18th IEEE International Conference on Automated Software Engineering*. IEEE; 2003, p. 152–161.
5. Fu, X., Bultan, T., Su, J.. Analysis of interacting bpel web services. In: *Proceedings of the 13th International Conference on World Wide Web*; WWW '04. ACM; 2004, p. 621–630.
6. Zheng, Y., Zhou, J., Krause, P.. A model checking based test case generation framework for web services. In: *4th International Conference on Information Technology (ITNG '07)*. IEEE; 2007, p. 715–722.
7. Zhu, Z., Lan, R., Ma, R., Chen, Y.. Describing and verifying semantic web service composition with MDA. In: *International Conference on E-Business and Information System Security (EBISS '09)*. IEEE; 2009, p. 1–6.
8. Bentahar, J., Yahyaoui, H., Kova, M., Maamar, Z.. Symbolic model checking composite web services using operational and control behaviors. *Expert Systems with Applications* 2013;**40**(2):508–522.
9. Sheng, Q.Z., Maamar, Z., Yao, L., Szabo, C., Bourne, S.. Behavior modeling and automated verification of web services. *Information Sciences* 2014;**258**:416–433.
10. El Kholy, W., Bentahar, J., El Menshawy, M., Qu, H., Dssouli, R.. Modeling and verifying choreographed multi-agent-based web service compositions regulated by commitment protocols. *Expert Systems with Applications* 2014;**41**(16):7478–7494.
11. Hamadi, R., Benatallah, B.. A petri net-based model for web service composition. In: *Proceedings of the 14th Australasian database conference-Volume 17*. Australian Computer Society, Inc.; 2003, p. 191–200.
12. Zhang, J., Chang, C., Chung, J.Y., Kim, S.. Ws-net: a petri-net based specification model for web services. In: *Proceedings of the IEEE International Conference on Web Services*. IEEE; 2004, p. 420–427.
13. Tan, W., Fan, Y., Zhou, M.. A petri net-based method for compatibility analysis and composition of web services in business process execution language. *IEEE Transactions on Automation Science and Engineering* 2009;**6**(1):94–106.
14. Schlingloff, H., Martens, A., Schmidt, K.. Modeling and model checking web services. *Electronic Notes in Theoretical Computer Science* 2005;**126**:3–26.
15. Baier, C., Katoen, J.P.. *Principles of model checking*. MIT press Cambridge; 2008.