PAPER

# A Correctness Assurance Approach to Automatic Synthesis of Composite Web Services*

**Dajuan FAN**[†a)], **Zhiqiu HUANG**[†], *Nonmembers*, *and* **Lei TANG**[††], *Member*

**SUMMARY** One of the most important problems in web services application is the integration of different existing services into a new composite service. Existing work has the following disadvantages: (i) developers are often required to provide a composite service model first and perform formal verifications to check whether the model is correct. This makes the synthesis process of composite services semi-automatic, complex and inefficient; (ii) there is no assurance that composite services synthesized by using the fully-automatic approaches are correct; (iii) some approaches only handle simple composition problems where existing services are atomic. To address these problems, we propose a correct assurance approach for automatically synthesizing composite services based on finite state machine model. The syntax and semantics of the requirement model specifying composition requirements is also proposed. Given a set of abstract BPEL descriptions of existing services, and a composition requirement, our approach automatically generate the BPEL implementation of the composite service. Compared with existing approaches, the composite service generated by utilizing our proposed approach is guaranteed to be correct and does not require any formal verification. The correctness of our approach is proved. Moreover, the case analysis indicates that our approach is feasible and effective.

***key words:*** *web services, composite service, FSM model, automatic synthesis, composition requirement*

## 1. Introduction

As one of the important technologies for realizing services oriented architecture (SOA), web services has proved effective for the application-integration problem in distributed, dynamic and heterogeneous environments. Due to limited functionalities provided by single web services, they are often composed to form a value-added composite service. As the manual development of the new composite service is regarded as a difficult and error-prone task, the automatic synthesis of a composite service, i.e., how to automatically compose existing services to form a correct composite service, has become an important issue in the field of web services. As the basic failure of a system, deadlocks can occur under many situations, e.g, when a component service and its composite service are waiting to receive some message from the other at the same time, and when a message in a component service do not have the correspondence in its composite service. Thus, deadlocks should be avoided when synthesizing a composite service. In other words, the composite service to be synthesized need guarantee a deadlock-free interaction with existing services. Besides, the synthesized composite service should also satisfy composition requirements the users specify, via interactions with these existing services. Hence, the correctness of a composite service includes two aspects: deadlock-freeness and requirement satisfaction [1], [2].

There have been many efforts to synthesize composite services [3]–[8]. However, all of them require developers to provide a composite service model first. In order to ensure the correctness of the composite service, developers need to perform formal verifications such as model checking to verify whether the provided composite service model satisfies the correctness constraints. If it is incorrect, developers must re-design the model and perform verifications again. The process of design, verification, and re-design makes the synthesis of composite services semi-automatic, often time consuming and costly.

Recently, adaptor-based service composition approaches have received significant attention [9]–[12]. Approaches belonging to this kind aim at automatic synthesis of a composite service called "adaptor" to ensure the deadlock-free interactions with existing services. However, they do not take into account composition requirements. A composite service synthesized by this kind of approaches may not satisfy the given requirements and hence is not ensured to be correct.

In addition, several work has been proposed to support automatic synthesis of composite services which can be adopted exclusively for modeling simple composition problems, where existing services are atomic, see, e.g, [13]–[15]. They do not take behavioral descriptions of web services into account, like abstract BPEL processes. In other work from [16], there is no special solution to deal with deadlock, and hence, in order to ensure the deadlock-freeness aspect, the designers need to specify the deadlocking interactions. However, since the deadlocking interactions (i.e., the interactions which cause deadlocks) can occur under many situations during the interactions among a composite service and its environment, i.e., existing services to be composed, they are usually hard to be predicted and specified, especially for

Manuscript received March 6, 2013.
Manuscript revised December 30, 2013.

[†]The authors are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, P.R. China.
[††]The author is with the College of Electronics and Information Engineering, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, P.R. China.

*This work in this paper is supported by the National Nature Science Foundation of China (No. 61272083, No. 61100034 and No. 61170043), the National High-Tech Research and Development Plan of China (No. 2009AA010307), and China Postdoctor Research Foundation (20110491411).

a) E-mail: fanbingjie523@126.com (Corresponding author)

DOI: 10.1587/transinf.E97.D.1535

large-scale web services applications.

In this paper, we present a correctness assurance approach for automatically synthesizing a composite service with services exposing interaction protocols, and with composition requirements expressing temporal constraints on the service interactions. Given a set of abstract BPEL processes describing interaction protocols of existing services, and a composition requirement expressing temporal constraints on their interactions, our approach automatically generates the BPEL implementation of the composite service by using the finite state machine (FSM) model as an intermediate formal language. Composition requirements are also formalized using our proposed FSM-based model with a clear syntax and semantics. The basic idea of our automatic synthesis approach is first to obtain an initial model of the composite service that reflects all the interactions between existing services. We then operate on the initial model to derive the deadlock-free model of the composite service which ensures the deadlock-freeness aspect of being a correct composite service, and finally we derive the final model of the composite service by filtering these interactions violating the given composition requirements to satisfy the requirements of correctness.

The composite service automatically synthesized by utilizing our proposed approach is ensured to be correct. Unlike existing approaches, where the process of design, verification, and re-design makes the synthesis of composite services semi-automatic and time consuming, it does not require developers to provide a prior composite service model. The correctness of our proposed approach is proved. In addition, the feasibility of our approach is validated by means of an explanatory example throughout the paper.

The rest of the paper is organized as follows. Section 2 first introduces the formal models for specifying existing services, the composite service and the interactions between a set of existing services and their composite service respectively. Further, it illustrates the problem of synthesizing the correct composite service through an explanatory example which will be used all along the paper. Section 3 proposes a requirement model with a clear syntax and semantics for expressing temporal properties given by composition requirements. Section 4 shows our synthesis process for the correct composite service and validates our approach based on the explanatory example. Section 5 gives a comparison with related work and finally, Sect. 6 ends the paper with some concluding remarks.

## 2. Problem Description

### 2.1 Preliminaries

In the paper, existing services which are composed to form a composite service are also called **component services**. To support automatic synthesis of a correct composite service for a set of component services and a given composition requirement, the formal modeling and analysis methods should be utilized. We choose FSM model as the adopted formalism in this paper. The reason for using FSM model is that this formalism is intuitive and simple, and fairly easy to understand for users. Besides, FSM model is sufficient for characterizing the interactions aspect of services which we focus on in this paper.

**Definition 1 (FSM)**. A **FSM** model is a quintuple $M = (S, L, T, I, F)$,where $S$ is a set of states, $L$ is a set of transition labels, $T \subseteq S \times L \times S$ is a set of transitions, $I \in S$ is the initial state and $F \subseteq S$ is a set of final states. The transition $t = (s, \mu, s')$ is denoted as $t = s \xrightarrow{\mu} s'$.

The labels in the FSM model can be an action which represents a component service sends or receives a message, labeled message or labeled message expression which represents message exchange between component services.

**Definition 2 (Trace).** Let $M = (S, L, T, I, F)$ be a FSM model, a trace of $M$ is a $\mu \in L^*$ defined in such a way that $\mu = \mu_1\mu_2 \ldots \mu_n \wedge s_0 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_n} s_n$ where $s_0, \ldots, s_n \in S, s_0 = I, s_n = F, \mu_1, \ldots, \mu_n \in L$. The set of traces of $M$ is denoted as Trace($M$).

**Definition 3 (Deadlock State and Deadlock Trace).** Let $M = (S, L, T, I, F)$ be a FSM model, a state $s \in S$ is a deadlock state for $M$ if and only if $s \notin F \wedge \neg\exists a \in L, s' \in S : (s, a, s') \in T$. If there exists a deadlock state in $M$, we denote it as **Deadlock**($M$)= **True**. Otherwise, we denote it as **Deadlock**($M$)= **False**. The set of deadlock states in $M$ is denoted as $S_{dead}(M)$. A $\mu \in L^*$ is a deadlock trace of $M$ if and only if $\mu = \mu_1\mu_2 \ldots \mu_n \wedge s_0 \xrightarrow{\mu_1} \ldots \xrightarrow{\mu_n} s_n$ where $s_0, \ldots, s_n \in S, s_0 = I, s_n = S_{dead}(M), \mu_1, \ldots, \mu_n \in L$.

### 2.2 Component Service Model

As mentioned in Sect. 1, a correct composite service to be synthesized needs to have appropriate interactions (i.e., deadlock-free and requirement-satisfying interactions) with component services. Abstract BPEL processes are often used to describe interaction protocols (also called behaviors) of component services. Due to the lack of precise semantics, abstract BPEL processes need to be translated into the corresponding FSM-based models, which are called SM models in this paper.

**Definition 4 (Component Service Model, SM for short)**. A component service can be described by a special FSM model **SM** $= (S, A, T, I, F)$,where $S$ is a set of service states; $I \in S$ is the initial state; $F \subseteq S$ is the set of final states; $A \subseteq \{!, ?\} \times \sum_M$ is a set of service actions; $T \subseteq S \times A \times S$ is a set of transitions.

A message $m \in \sum_M$ corresponds to an operation in WSDL specifications of component services, where the symbol $\sum_M$ represents the set of messages. The symbols "!$m$" and "?$m$" denote the send action and receive action of message $m$ respectively. We define complementarity as $\overline{?m} = !m$ and $\overline{!m} = ?m$.

Abstract BPEL processes, which abstract from the service internal activities, only include basic communication activities (<*invoke*> activity, <*reply*> activity and <*receive*> activity) and structured activities (<*flow*> activ-

ity, <*sequence*> activity, <*switch*> activity, and so on). According to the mapping relations between these activities in abstract BPEL processes and the elements of FSM models in [17], the SM model representation of a component service can be easily derived from its abstract BPEL specification.

***Example 1*** *Our explanatory example consists in providing a composite service for file download and query, say the FileTQ service, which satisfies the given composition requirement, by composing three independently designed, existing component services:* ***DLfile*** *service,* ***RfileInfo*** *service and* ***FileServer*** *service.****DLfile*** *service is the client-side service requesting for file download.****RfileInfo*** *service is the client-side service querying for file information.* ***FileServer*** *service is the server-side service providing download and information query for a given file. Hereafter we present a simplification of the abstract BPEL processes for the three component services. Note that, in order to express the message exchanges focused in the paper, we simply use service names instead of partnerLinks and portTypes.*

```
<process name="RfileInfo">
  <sequence>
    <while>
      <invoke operation ="R_FN" of FileServer inputVariable="filename"
      outputVariable="fileinfo"/>
    </while>
    <invoke operation ="R_DC" of FileServer Variable="disconnect"/>
  </sequence>
</process>

<process name="DLfile">
  <switch>
    <while>
      <sequence>
        <invoke operation ="LI" of FileServer Variable="login"/>
        <invoke operation ="LO" of FileServer Variable="logout"/>
      </sequence>
    </while>
    <while>
      <sequence>
        <invoke operation="DL" of FileServer Variable =" download"/>
        <invoke operation="D_FN" of FileServer Variable =" filename"/>
        <switch>
          <invoke operation ="TX" of FileServer inputVariable="textformat"
          outputVariable ="text" />
          <invoke operation ="VD" of FileServer inputVariable="videoformat"
          outputVariable ="video" />
        </switch>
      </sequence>
    </while>
    <invoke operation ="D_DC" of FileServer Variable="disconnect"/>
  </switch>
</process>

<process name="FileServer">
  <while>
    <pick>
      <onMessage operation ="LI" from DLfile Variable="login">
        <receive operation ="LO" from DLfile Variable="logout"/>
      </onMessage>
      <onMessage operation="DL" from DLfile Variable ="download">
        <sequence>
          <receive operation="D_FN" from DLfile Variable =" filename"/>
          <switch>
            <sequence>
              <receive operation ="TX" from DLfile Variable="textformat"/>
              <reply operation ="TX" of DLfile Variable ="text" />
            </sequence>
            <sequence>
              <receive operation ="VD" from DLfile Variable="videoformat" />
              <reply operation ="VD" of DLfile Variable="video" />
            </sequence>
          </switch>
        <sequence>
      </onMessage>
      <onMessage operation ="R_FN" from RfileInfo Variable="filename">
        <reply operation ="R_FN" of RfileInfo Variable="fileinfo"/>
      </onMessage>
      <onMessage operation ="D_DC" from DLfile Variable="disconnect">
        <terminate>
      </onMessage>
      <onMessage operation ="R_DC" from RfileInfo Variable="disconnect">
        <terminate>
      </onMessage>
    </pick>
  </while>
</process>
```

*The abstract BPEL descriptions of the three component services are translated into corresponding SM models depicted in Fig. 1. According to SM models of the three component services, they are described as follows:*

*-****DLfile*** *service first sends download request (!download), and then sends the filename of the requested file (!filename) and download format (!textformat or !videoformat) to* ***FileServer*** *service, finally it receives the downloaded file from* ***FileServer*** *service (?text or ?video). When starting or ending a download process, it can login or logout (!login or !logout), or disconnect itself with* ***FileServer*** *service (!disconnect).*

*-****RfileInfo*** *service first sends the filename (!filename) of the queried file and then receives the file information (?fileinfo).When starting or ending a query process, it can disconnect itself with* ***FileServer*** *service (!disconnect).*

*-****FileServer*** *service first receives a request for providing file information (?filename) from* ***RfileInfo*** *service or a request for providing file download (?download) from* ***DLfile*** *service. In the first case, it sends the corresponding file information (!fileinfo) to* ***DLfile*** *service. In the second case, it continues to receive the filename (?filename) and download format of the file (?textformat or ?videoformat) to be*

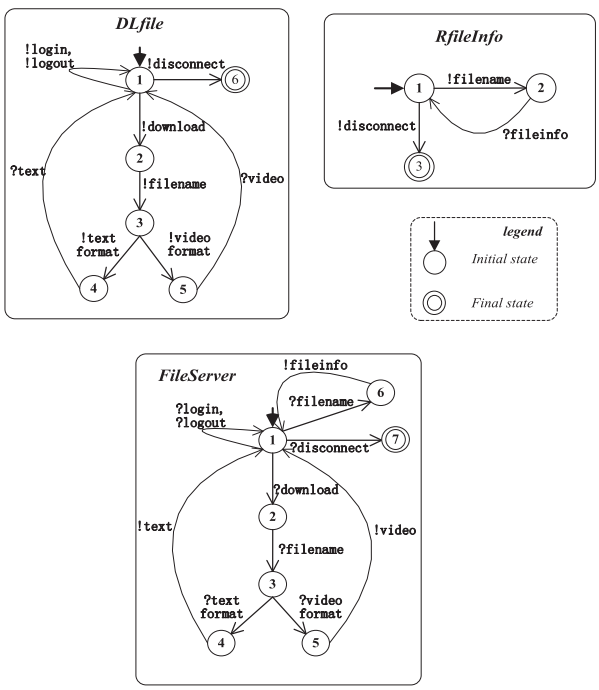


**Fig. 1** SM models for the DLfile serice, RfileInfo service and FileServer service.

*downloaded, and then sends the file in the corresponding format(!text or !video). When starting or ending a download process with* ***DLfile*** *service, it can receive the login or logout request (?login or ?logout), or the disconnect request (?disconnect). When starting or ending a query process with* ***FileServer*** *service, it can receive the disconnect request (?disconnect).*

*The composition requirement gives temporal properties as follows: "file download in video format is performed between* ***FileServer*** *service and* ***DLfile*** *Service under the login mode. Moreover, file download in text format is performed between* ***FileServer*** *service and* ***DLfile*** *Service under the logout mode".*

Indeed, component services which are developed by different third-parties, may have different operation definitions i.e., there may not be a direct syntactical correspondence between messages in different component services. Since in this work we are focusing on automatically synthesizing a correct composite service at the level of the service interactions, we consider this problem out of the scope of this paper. Hereafter, we will assume that messages between component services already match since users already provide correspondences between messages of different component services in advance.

### 2.3 Composite Service Model and Interaction Model

A composite service interacts with a set of component services to achieve a specific business function. We assume that a composite service interacts with component services via synchronous communication. This is not a limitation because we can always simulate an asynchronous service composition [18] by a synchronous one with the introduction of a buffer component service.

In order to describe interactions between a composite service and a set of component services, we need to relate actions of a composite service to concrete component services by using a labeling function.

**Definition 5 (Labeling Function $f_i$).** Let SM = $(S, A, T, I, F)$ be the model of a component service, $f_i$ is a labeling function, $i \in \{1, \cdots, N\}$, the following formulas holds: $f_i(A) = \{a(i) \mid a \in A\}$, $f_i(\overline{A}) = \{\overline{a}(i) \mid a \in A\}$.

The labeled actions $?m(i)$ and $!m(i)$ represent that the composite service receives the message $m$ from the $i$th component service and sends the message $m$ to the $i$th component service respectively, $i \in \{1, \cdots, N\}$.

**Definition 6 (Composite Service Model, CM for short).** Let $\mathrm{SM}_i = (S_i, A_i, T_i, I_i, F_i)$ be the models of $N$ component services, $i \in \{1, \cdots, N\}$, where $N$ stands for the number of component services to be composed. Their composite service can be described by a special FSM model **CM** $= (S, A, T, I, F)$,where:

$-S \subseteq S_1 \times \cdots \times S_N, I = (I_1, \cdots, I_N), F \subseteq F_1 \times \cdots \times F_N$;

$-A \subseteq \bigcup_{i=1}^{N} f_i(\overline{A}_i), T \subseteq S \times A \times S$.

The states in the CM model correspond to combinations of states of $N$ SM models. For example, the composite service is in its state $(s_1, \cdots, s_N)$ if and only if each component service $i$ is in its state $s_i$. The formula $A \subseteq \bigcup_{i=1}^{N} f_i(\overline{A}_i)$ means that every action in CM model has the corresponding complement action in $\mathrm{SM}_i$ model. Specially, for each action $?m(i)$ in the CM model, there exists the send action $!m$ in the $\mathrm{SM}_i$ model which represents the $i$th component service sends the message $m$, and for each action $!m(j)$ in the CM model, there exists the receive action $?m$ in the $\mathrm{SM}_j$ model which represents the $j$th component service receives the message $m$, $i, j \in \{1, \cdots, N\}$.

A correct composite service needs to have deadlock-free and requirement-satisfying interactions with component services. So in the following we will give the interaction model between a composite service and a set of component services.

**Definition 7 (Interaction Model between a Composite service and Component Services).** Let $\mathrm{SM}_i = (S_i, A_i, T_i, I_i, F_i)$ be models of $N$ component services, $i \in \{1, \cdots, N\}$ and CM=$(S_C, L_C, T_C, I_C, F_C)$ be the model of a composite service, the interaction model between a composite service and $N$ component services is the $\mathrm{SM}_1|\cdots|\mathrm{SM}_N$ CM=$(S, LM, T, I, F)$, where:

$-S \subseteq S_1 \times \cdots \times S_N, I = (I_1, \cdots, I_N), F \subseteq F_1 \times \cdots \times F_N$;

$-LM \subseteq \sum_M \times \{1, \cdots, N\} \times \{1, \cdots, N\}$ is a set of labeled messages, $\sum_M \subseteq \bigcup_{i=1}^{N} \sum_M^i$, where the symbol $\sum_M^i$ denotes the set of messages of the $i$th component service;

$-(s_1, \cdots, s_N), m(i, j), (s'_1, \cdots, s'_N) \in T \Leftrightarrow \forall (s_1, \cdots, s_N) \in S$, $\exists i, j \in \{1, \cdots, N\}$, $(s_i, !m, s'_i) \in T_i \wedge (s_j, ?m, s'_j) \in T_j \wedge ((s_1, \cdots, s_N), ?m(i), s^*), (s^*, !m(j), (s'_1, \cdots, s'_N)) \in T_A \wedge \forall k \in \{1, \cdots, N\} : (k \neq i, j \Rightarrow s'_k = s_k) \wedge s^* = (s_1, \cdots, s'_i, \cdots, s_j, \cdots, s_N)$.

States in the interaction model correspond to states in the CM model, i.e., the combinations of states of $N$ SM models. The computation of the transitions expresses that, given some state $s$ in the interaction model, there is some transition $t : s \xrightarrow{m(i,j)} s'$ outgoing from this source state if and only if there are two component services, $i$ and $j$, that perform $!m$ and $?m$ from states $s_i$ and $s_j$ in their SM models respectively, while the composite service performs the receive action $?m(i)$ from the source state $s$ and reaches the state $s^*$ in which the composite service continues to perform the send action $?m(j)$ and comes to the target state $s'$, where $i \neq j$.The labeled message $m(i, j)$ in the interaction model denotes that the message m exchange between two component services, $i$ and $j$, via their composite service.

## 3. Requirement Model

Our aim is to automatically synthesize a correct composite service that interacts with component services seamlessly and satisfies the given composition requirements. In this paper, we focus on the important kind of composition requirements which express the desired temporal properties with regard to the services interaction. So in this section, we will introduce a formal model that can specify temporal properties given by composition requirements.

In terms of model checking, temporal properties are

usually expressed in temporal logic formulae, e.g., Linear-Time Temporal Logic (LTL) by making use of temporal operators. However, as discussed in [19], expressing temporal properties in those logics-based languages is a difficult task due to the inherent complexity of these languages. Apart from the advantages of FSM model discussed in Sect. 2.1, we present a FSM-based requirement model (RM model for short) for the reason of unified formal modeling. Besides, It is well-known that all Linear temporal logic formulae can be automatically translated into equivalent Buchi automata [20], [21], which are FSMs on infinite words [22]. In particular, the translation from the temporal operators such as "G", "F", "U", to FSMs is discussed in [20]. Therefore, the FSM-based requirement model proposed in our approach, is sufficient to specify temporal constraints expressed by temporal operators such as "G", "F", "U".

In our approach, temporal properties given by composition requirements are represented as the set of traces of the corresponding FSM-based requirement model, where traces describe the permitted temporal constraints on message exchanges between component services. As introduced in Definition 7, the labeled message $m(i, j)$ describes the message $m$ exchange between the $i$th component service and the $j$th component service via their composite service. In order to facilitate expressions of some complex composition requirements, we introduce the concept of labeled message expression.

Labeled message expressions are generated according to the following syntax:

$LM_{Ex} ::= m(i, j)|exc(m(i, j))|\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$ $|exc\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$

The symbol $Ex(LM)$ is used to denote a set of labeled messages expressions generated by a set $LM$ of labeled messages according to the aforementioned syntax.

The semantics of labeled message expressions are given by the following semantic equivalence formulas, where the binary relation $\cong_{Ex}$ represents two labeled message expressions are semantically equal:

(1) $m_1(i_1, j_1) \cong_{Ex} m(i, j) \Leftrightarrow m = m_1 \wedge i = i_1 \wedge j = j_1$;

(2) $exc(m_1(i_1, j_1)) \cong_{Ex} m(i, j) \Leftrightarrow m \neq m_1 \vee i \neq i_1 \vee j \neq j_1$;

(3) $\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\} \cong_{Ex} m(i, j) \Leftrightarrow \exists l \in \{1, \cdots, k\} : m(i, j) \cong_{Ex} m_l(i_l, j_l)$;

(4) $exc\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\} \cong_{Ex} m(i, j) \Leftrightarrow \nexists l \in \{1, \cdots, k\} : m(i, j) \cong_{Ex} m_l(i_l, j_l)$;

According to the above semantic equivalence formulas, the labeled message expression $\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$ represents any labeled message in the set $\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$, the labeled message expression $exc(m_1(i_1, j_1))$ represents any labeled message in $LM$ except the label message $m_1(i_1, j_1)$, and the labeled message expression $exc\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$ represents any labeled message in $LM$ except the label messages $m_1(i_1, j_1), \cdots, m_k(i_k, j_k)$.

**Definition 8 (Requirement Model, RM for short)**. A composition requirement expressing temporal properties can be described by a special FSM model

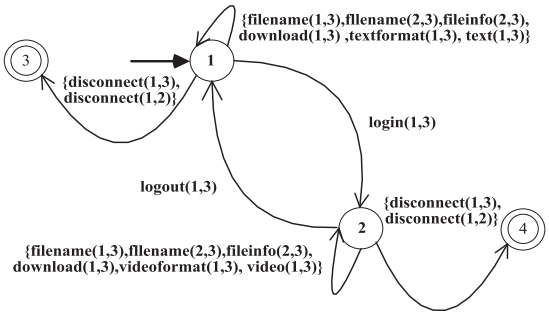


**Fig. 2** RM model describing the composition requirement for the composite service FileTQ.

**RM**=$(S, LM, LE, T, I, F)$,where:

-$S$ is a set of abstract states;

-$LM \subseteq \sum_M \times \{1, \cdots, N\} \times \{1, \cdots, N\}$ is a set of labeled messages, $\sum_M \subseteq \bigcup_{i=1}^{N} \sum_M^i$, where the symbol $\sum_M^i$ denotes the set of messages of the $i$th component service;

-$LE \subseteq Ex(LM)$ is a set of labeled message expressions;

-$T \in S \times LE \times S$ is a set of transitions;

-$I \in S$ is the initial abstract state;

-$F \subseteq S$ is a set of final abstract states;

Each of the abstract state in a RM model corresponds to one or more actual states in the corresponding interaction model. RM model is always in abstract forms and hence the labels of loop transitions in a RM model may be in the form of $\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$ or $exc\{m_1(i_1, j_1), \cdots, m_k(i_k, j_k)\}$, which indicates that these labeled messages happen without temporal constraints. Therefore, users often do not need to have a prior understanding of all SM models details, when giving the RM model.

**Example 2**. *The RM model describing the composition requirement in example 1 is shown in Fig. 2. Numbers 1,2,3 in the labeled messages depicted in Fig. 2 correspond to the* ***DLfile*** *service,* ***RfileInfo*** *service and* ***FileServer*** *service respectively. According to the RM model in Fig. 2, the labeled messages videoformat(1,3) and video(1,3) happen after the labeled message login(1,3), which means that "file download in video format is performed between FileServer service and DLfile Service under the login mode". The labeled messages textformat(1,3), text(1,3) happen before the labeled message login(1,3) (or happen after the labeled message logout(1,3)), which means that "file download in text format is performed between FileServer service and DLfile Service under the logout model". So the RM model depicted in Fig. 2 can express the composition requirement in example 1.*

## 4. The Automatic Synthesis Approach

### 4.1 The Problem of Synthesizing Composite Services

Based on the discussion in the previous sections, we formally define the automatic synthesis problem of a correct composite service as follows: given $N$ SM models specify-

ing interaction protocols of component services and the RM model expressing temporal constraints on the services interactions given by composition requirements, the CM model of a composite service is automatically constructed, satisfying the following formulas:

$$\text{Deadlock}(\text{SM}_1 |\cdots |\text{SM}_N|\text{CM}_\text{F}) = \text{False}. \quad (4.1)$$

$$(\text{SM}_1|\ldots|\text{SM}_N|\text{CM}_\text{F}) \models \text{RM}. \quad (4.2)$$

where the formula C $\models$ P represents that the model C satisfies the requirement P.

Formula (4.1) indicates that the synthesized composite service interacts with component services seamlessly, i.e., the interactions between component services and their composite service are deadlock-free, and Formula (4.2) states that the synthesized composite service obeys the temporal constraints given by composition requirements via the interactions with component services. Therefore, the composite service implemented according to the constructed CM model satisfying the two formulas is guaranteed to be a correct composite service.

The basic idea of our correctness assurance approach for automatically synthesizing a composite service is to obtain firstly an initial model of the composite service that reflects all possible interactions between component services. However, the interactions between component services via their composite service which is generated according to the initial model of it may contain the interactions leading to deadlocks. So then, by operating on the obtained initial model, the deadlock analysis is performed and subsequently the deadlock-free model of the composite service is derived, which ensures the deadlock-freeness aspect of being a correct composite service. To ensure the requirement satisfaction aspect of correctness, by operating on the deadlock-free model, the final model of the composite service is produced, which satisfies the given composition requirement.

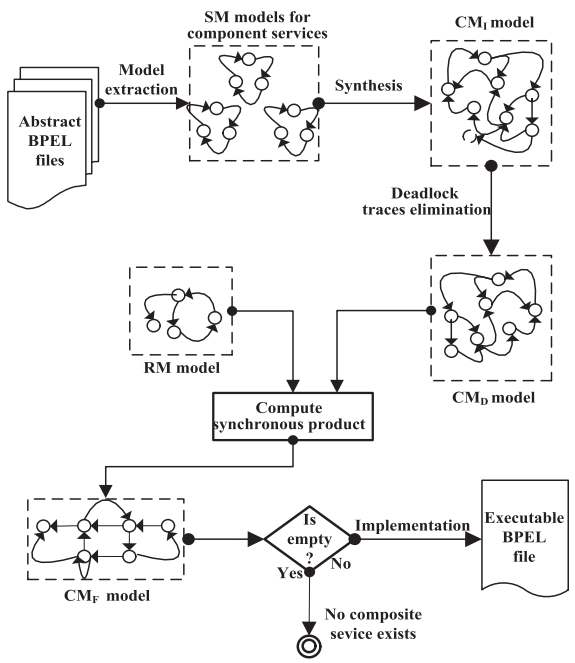


**Fig. 3** The automatic synthesis procedure for correct composite services.

Therefore, our correctness assurance approach to automatic synthesis of composite services follows these steps described in Fig. 3. Firstly, the initial model of a composite service denoted as $\text{CM}_\text{I}$ is generated. Secondly, the deadlock-free model denoted as $\text{CM}_\text{D}$ is gained by eliminating deadlock traces of the $\text{CM}_\text{I}$ model to ensure deadlock-free interactions between component services and their composite service. Then, the final model of the composite service called $\text{CM}_\text{F}$ model is derived from the $\text{CM}_\text{D}$ model by filtering the interactions which violate the given composition requirement expressed by the RM model. Finally, the concrete BPEL implementation of the composite service, which can be executed on standard business processes execution engines, is automatically generated according to the $\text{CM}_\text{F}$ model.

In the following subsections we will describe these steps in our synthesis approach separately.

### 4.2 Synthesis of Composite Service Models

For a set of component services, the $\text{CM}_\text{I}$ model of their composite service reflects all the possible interactions between component services. Once a component service sends a message and another component service is ready to receive the message, the composite service specified by the $\text{CM}_\text{I}$ model will first receive the message, and then forward it to another component service, strictly following its former message reception. Therefore, when eliminating interactions which lead to deadlocks and filtering interactions which do not obey the given composition requirement, it is only required to operate on the $\text{CM}_\text{I}$ model of composite service.

**Algorithm 1** The $\text{CM}_\text{I}$ model synthesis algorithm
**Input:** $N$ SM models of component services.
**Output:** the $\text{CM}_\text{I}$ model.
**(1)** Set the action set $A = \emptyset$, the transition set $T = \emptyset$, the initial state $I = (I_1, \ldots, I_N)$, the state set $S$={$I$};
**(2)** Call function *CreateCS* ($I$);
**Function** *CreateCS*($s$[]){
visited[$s$]=True;
**For** $i$=1 to $N$
**For** $j$ =1 to $N$
**If** $(v_i == \text{s}[i] \wedge v_j == s[j] \wedge (v_i, !m, v_i') \in T_i \wedge (v_j, ?m, v_j') \in T_j)$ **Then{**
$A = A \cup \{?m(i), !m(j)\}$;
**For** $k$=1 to $N$
**If** $k = i$ **Then** $s'[k] = v_i'$;$s''[k] = v_i'$;}
**ElseIf** $k = j$ **Then** {
$s'[k] = s[k]$;$s''[k] = v_j'$;}
**Else** {$s'[k] = s[k]$;$s''[k] = s[k]$;}
$T = T \cup \{(s, ?m(i), s'), (s', !m(j), s'')\}$;
$S = S \cup \{s', s''\}$;
**If** ($s'' \notin Final \wedge$visited[$s''$]==False) **Then**
*CreateCS*($s''$);
}

}

Initially, Algorithm 1 creates the $CM_I$ model with one state $I$ (corresponding to the combination of the respective initial states of each component service) and no transitions. Then it calls the recursive function *CreateCS* with the current state $I$ of $CM_I$ model as the parameter value. The function *CreateCS* finds pairs of complementary transitions $(v_i, !m, v_i') \in T_i$ and $(v_j, ?m, v_j') \in T_j$ among $N$ component services from their respective current states $s[i]$ and $s[j]$, and adds the corresponding labeled actions $?m(i)$ and $!m(j)$ to the set $A$, one receive action followed by the other send action. Then, the two successors $s'$ and $s''$ of the state $s$ that the $CM_I$ model successively reaches after performing the labeled action $?m(i)$ and the labeled action $!m(j)$ from the state $s$ are computed and added to the set $S$. The set $T$ is also updated by adding the two new transitions $t = (s, ?m(i), s')$ and $t' = (s', !m(j), s'')$. Then the function *CreateCS* recursively calls itself with the current state $s''$of $CM_I$ model. The procedure lasts until the state $s''$ reaches the final states (corresponding to the combination of the respective final states of each component service). Supposing that the maximum number of transitions among $N$ SM models is $|T|$, we can conclude that the maximum number of transitions of $CM_I$ model is $N|T|$ according to the $CM_I$ model generation process in Algorithm 1, and hence the time complexity of Algorithm 1 is deduced to be $O(N^2|T|^2)$ in the worst case.

The $CM_I$ model generated by Algorithm 1 may not be deadlock-free. Deadlock traces of a $CM_I$ model represent the interactions leading to deadlocks between a composite service and component services. In order to ensure the deadlock-free interactions, we should restrict the set of all possible interactions to deadlock-free interactions by eliminating deadlock traces of the $CM_I$ model and gain the deadlock-free model denoted as $CM_D$.

The $CM_D$ model may include interactions which violate the given composition requirement, so we will introduce the synchronous product of RM model and CM model to filter these interactions and gain the final model denoted as $CM_F$. Therefore, the composite service implemented according to the $CM_F$ model is guaranteed to be correct because it can not only interacts with component services seamlessly but also satisfies the given composition requirement by interactions with them.

According to Definition 7, the two successive labeled actions $?m(i)$ and $!m(j)$ in CM models mean that a composite service receives the message $m$ from the $i$th component service and sends it to the $j$th component service. The labeled message $m(i, j)$ represents the message exchange between component service $i$ and $j$ via the composite service. Hence, the two successive labeled actions $?m(i)$ and $!m(j)$ correspond to these labeled message expressions which are semantically equivalent to the labeled message $m(i, j)$. Based on the above discussion, we will give the trace equivalence relation between CM model and RM model, and their synchronous product in the following.

**Definition 9 (Trace Equivalence Relation between RM Model and CM Model).** Two traces $\mu_C = \mu_1^C \mu_2^C \ldots \mu_n^C$ and $\mu_R = \mu_1^R \mu_2^R \ldots \mu_l^R$ are the traces of CM model and RM model respectively, we say that the two traces are equivalent, denoted as $\mu_C \approx \mu_R$, if and only if $n=2l$ and $\forall k \in \{1, 2, \ldots, l\} : \mu_{2k-1}^A = ?m(i) \wedge \mu_{2k}^A = !m(j) \wedge m(i, j) \cong_{Ex} \mu_k^R$.

**Definition 10 (The Synchronous Product of RM model and CM Model).** Let $CM=(S_C, L_C, T_C, I_C, F_C)$ be a CM model and $RM=(S_R, LM_R, LE_R, T_R, I_R, F_R)$ be a RM model, their synchronous product is $SP(CM,RM)=(S, A, T, I, F)$, where:

$-S \subseteq S_C \times S_R, I = (I_C, I_R), F \subseteq F_C \times F_R,$

$-((s, r), ?m(i), (s', r)), ((s', r), !m(j), (s'', r')) \in T \Leftrightarrow$

$\exists (s, ?m(i), s'), (s', !m(j), s'') \in T_C : \exists (r, l, r') \in T_R$

$: m(i, j) \cong_{Ex} l.$

According to Definition 9-10, SP(CM,RM) only includes these traces of the CM model which have equivalent traces in the RM model. Since traces of the RM model give the temporal constraints on message exchanges between component services, SP(CM,RM) filters the interactions which do not obey these temporal constraints specified by the RM model.

Based on the above discussion in this section, we summarize the steps of our entire synthesis approach used to automatically build the final model $CM_F$ of the correct composite service by performing the following Algorithm 2.

**Algorithm 2** The $CM_F$ model synthesis algorithm

**Input:** $N$ SM models and an requirement model RM.

**Output:** the $CM_F$ model.

**(1)** derive $CM_I$ model according to Algorithm 1;

**(2) If** ((Deadlock($CM_I$)==True)

**Then** Eliminate all the deadlock traces of $CM_I$ model and derive $CM_D$ model;

**Else** $CM_D = CM_I$;

**(3) If** ($CM_D$ is empty) **Then** exit;

**(4)**

**(4.1)** $CM_F$=SP($CM_D$,RM);

**(4.2) If** ($CM_F$ is empty) **Then** {output "No correct composite service exists"; exit;}

**(5)** output $CM_F$;

Firstly, the $CM_I$ model is generated according to Algorithm 1. Then, if the generated $CM_I$ model contains deadlock states (i.e., the formula Deadlock($CM_I$)==True holds) according to Definition 3, all the deadlock traces of the $CM_I$ model are eliminated to derive the $CM_D$ model. The deadlock state containment check and the deadlock traces elimination for $CM_I$ model in Step 2 are realized by means of a classical depth-first visit. In particular, the procedure for realizing Step 2 begins with the initial state of the $CM_I$ model and set it as the current state. If a deadlock state has been found, the deadlock state and all its incoming transitions are removed. Otherwise, the procedure is recursively performed on the non-visited successors of the current state. The procedure ends when all the states of the $CM_I$ model have been visited. According to the above procedure, if the generated $CM_I$ model contains deadlock states, all the deadlock traces of the generated $CM_I$ model can be removed in order to automatically derive the $CM_D$ model.

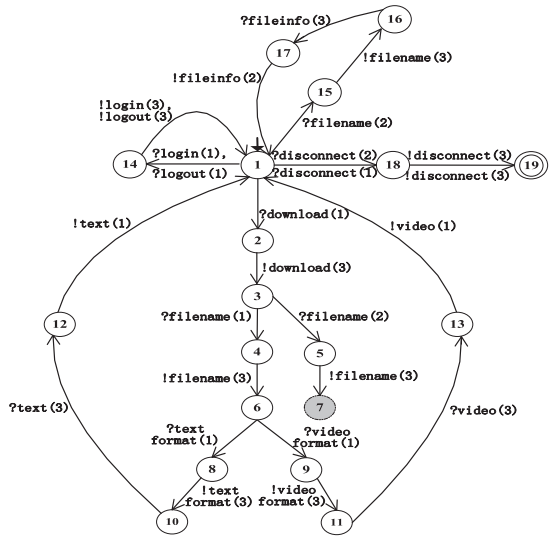


**Fig. 4** $CM_I$ model for the composite service FileTQ.

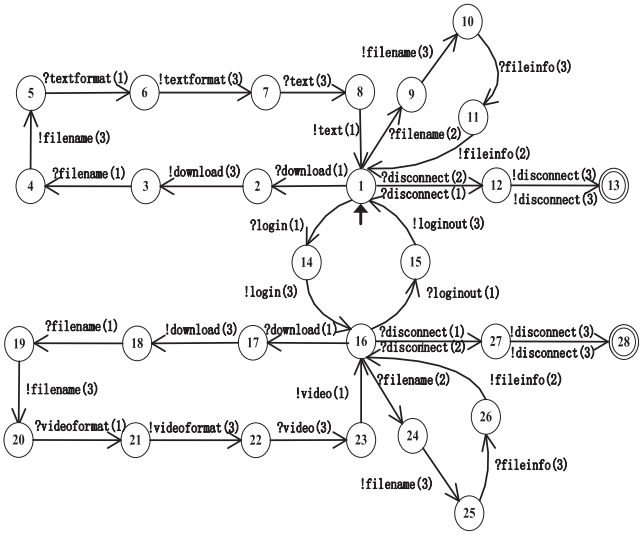


**Fig. 5** $CM_F$ model for the composite service FileTQ.

In the next step, the empty determination of $CM_D$ model is done. If it is not empty, the synchronous product of $CM_D$ model and RM model, i.e., SP($CM_D$,RM) is computed. The computation of SP($CM_D$,RM) is formalized through Definition 10. Specially, the procedure for computing SP($CM_D$,RM) first builds the set of states of SP($CM_D$,RM) by performing the Cartesian product of the set of states of $CM_D$ model and RM model. Then, the set of transitions of SP($CM_D$,RM) is generated according to the definition of transitions in Definition 10. Finally, the algorithm discards the states that can not be reached from the initial state (see the definition of the initial state in Definition 10). According to the above procedure which comes directly from Definition 10, SP($CM_D$,RM) is computed and thus the $CM_F$ model is derived. If the derived $CM_F$ model is empty, it means that there exists no correct composite service for these$N$ component services. Otherwise, the $CM_F$ model is output as the final model of the correct composite service. The time complexity of deadlock elimination is $O(N|T|+M)$ where $M$ is the number of states of $CM_I$ model, and the time complexity of synchronous product computation is $O(N|T|)$ according to Definition 10. So we can conclude the time complexity of Algorithm 2 is $O(N^2|T|^2)$ due to the fact that $M$ is typically fewer than $N^2|T|^2$.

**Example 3.** *Fig. 4 gives the $CM_I$ model for the composite service FileTQ generated by Algorithm* 1. *The trace* $s_1 \xrightarrow{?download(1)} s_2 \xrightarrow{!download(3)} s_3 \xrightarrow{?filename(2)} s_5 \xrightarrow{?filename(3)} s_7$ *in the $CM_I$ model is a deadlock trace. This trace reflects the interaction leading to deadlock between the three component services and their composite service and should be eliminated. So the two transitions* $s_3 \xrightarrow{?filename(2)} s_5$ *and* $s_5 \xrightarrow{?filename(3)} s_7$ *in the $CM_I$ model are eliminated and $CM_D$ model is obtained (the figure describing the $CM_D$ model is omitted due to space limitations).*

*According to the RM model depicted in Fig. 2, the synchronous product of RM model and $CM_D$ Model is computed and the final model $CM_F$ for composite service FileTQ is derived, as shown in Fig. 5. Note that, numbers 1,2,3 in the labeled actions correspond to the **DLfile** service, **RfileInfo** service and **FileServer** service depicted in Fig. 4 and Fig. 5 respectively. For all the traces of the $CM_F$ model, the transition sequence* $s_{20} \xrightarrow{?videofomat(1)} s_{21} \xrightarrow{!videoformat(3)} s_{22} \xrightarrow{?video(3)} s_{23} \xrightarrow{?text(1)} s_1$ *appears after the transition sequence* $s_1 \xrightarrow{?login(1)} s_{14} \xrightarrow{!login(3)} s_{16}$, *which means that the composite service satisfies the temporal constraint "file download in video format is performed between **FileServer** service and **DLfile** Service under the login mode", and the transition sequence* $s_5 \xrightarrow{?textfomat(1)} s_6 \xrightarrow{!textformat(3)} s_7 \xrightarrow{?text(3)} s_8 \xrightarrow{?text(1)} s_1$ *appears before the transition sequence* $s_1 \xrightarrow{?login(1)} ws_{14} \xrightarrow{!login(3)} s_{16}$ *or after the transition sequence* $s_{16} \xrightarrow{?loginout(1)} s_{15} \xrightarrow{!loginout(3)} s_1$, *which means that the composite service satisfies the temporal constraint "file download in text format is performed between **FileServer** service and **DLfile** Service under the logout model". Therefore, the composite service FileTQ implemented according to the $CM_F$ model in Fig. 5 is correct because it can not only interact with the three component services without deadlock but also satisfy temporal properties specified by the composition requirement via interactions with these component services.*

### 4.3 Implementation of Composite Services

The main part of the transformation is concerned with the encoding of the state and transition structure of the $CM_F$ model. Additionally, BPEL implementations include the following aspects: WSDL interfaces partnership definition,

basic activities (e.g., <*receive*>, <*reply*>, <*invoke*> activities), or structured activities (e.g., <*sequence*>, <*flow*>, <*switch*>, <*pick*> activities). The transform process is specifically as follows:

(1) A partner link is created for each component service, and variables are created for each received/emitted message.

(2) The <*Process*> activity and </*Process*> activity are generated, which logically mark the initiation of the BPEL process, as well as their termination, respectively.

(3) The transition in the form of $(u, ?m(i), u')$ is encoded as a <*receive*> activity or a <*onMessage*> event, and the transition in the form of $(v, !m(j), v')$ is encoded as a <*invoke*> activity. Specifically, if the number of transitions outgoing from the state $s$ is greater than or equal to 2 and they are all receive transitions, then each receive transition is encoded as a <*onMessage*> event, otherwise each of them is encoded as a <*receive*> activity.

(4) According to the structural position where a transition is in the $CM_F$ model, its basic activity is placed in the corresponding structured activity such as <*sequence*>, <*flow*>, <*switch*>, <*pick*> activities and all the activities including basic activities and structured activities are placed between <*Process*> activity and </*Process*> activity.

According to the above procedure, the $CM_F$ model for the synthesized composite service FileTQ in Fig. 5, leads to the following BPEL process:

```
<process name=" FileTQ">
  <while>
    <pick>
      <onMessage operation="DL" from DLfile Variable =" download">
        <invoke operation ="DL" of FileServer Variable="download"/>
        <receive operation="D_FN" from DLfile Variable =" filename"/>
        <invoke operation="D_FN" of FileServer Variable =" filename"/>
        <receive operation ="TX" from DLfile Variable="textformat"/>
        <invoke operation ="TX"of FileServer Variable="textformat"/>
        <receive operation ="TX" from DLfile Variable="text"/>
        <invoke operation ="TX"of FileServer Variable="text"/>
      </onMessage>
      <onMessage operation ="R_FN" from RfileInfo Variable="filename">
        <invoke operation ="R_FN" of FileServer Variable="filename"/>
        <receive operation="D_FN" from RfileInfo Variable =" fileinfo"/>
        <invoke operation="D_FN" of FileServer Variable =" fileinfo"/>
      </onMessage>
      <onMessage operation ="D_DC" from DLfile Variable="disconnect">
        <invoke operation ="D_DC" of FileServer Variable="disconnect"/>
        <terminate>
      </onMessage>
      <onMessage operation ="R_DC" from RfileInfo Variable="disconnect">
        <invoke operation ="R_DC" of FileServer Variable="disconnect"/>
        <terminate>
      </onMessage>
      <onMessage operation ="LI" from DLfile Variable="login">
        <invoke operation ="LI" of FileServer Variable="login"/>
        <assgin><copy> <from> "True" </from> <to variable="login "/>
    </copy>
        </assign>
        <while login==True>
          <pick>
            <onMessage operation="DL" from DLfile Variable =" download">
              <invoke operation ="DL" of FileServer Variable="download"/>
              <receive operation="D_FN" from DLfile Variable =" filename"/>
              <invoke operation="D_FN" of FileServer Variable =" filename"/>
              <receive operation ="VD" from DLfile Variable="videoformat"/>
              <invoke operation ="VD"of FileServer Variable="videoformat"/>
              <receive operation ="VD" from DLfile Variable="video"/>
              <invoke operation ="VD"of FileServer Variable="video"/>
            </onMessage>
            <onMessage operation ="R_FN" from RfileInfo Variable="filename">
              <invoke operation ="R_FN" of FileServer Variable="filename"/>
              <receive operation="D_FN" from RfileInfo Variable =" fileinfo"/>
              <invoke operation="D_FN" of FileServer Variable =" fileinfo"/>
            </onMessage>
            <onMessage operation ="D_DC" from DLfile Variable="disconnect">
              <invoke operation ="D_DC" of FileServer Variable="disconnect"/>
              <terminate>
            </onMessage>
            <onMessage operation ="R_DC" from RfileInfo Variable="disconnect">
              <invoke operation ="R_DC" of FileServer Variable="disconnect"/>
              <terminate>
            </onMessage>
            <onMessage operation ="LO" from DLfile Variable="loginout">
              <invoke operation ="LO" of FileServer Variable="loginout"/>
              <assgin><copy> <from> "False" </from> <to variable="login
    "/></copy>
              </assign>
            </onMessage>
          </pick>
        </while>
      </onMessage>
    </pick>
  </while>
</process>
```

## 4.4 Correctness of Our Approach

In order to prove the correctness of our proposed synthesis approach, we need to prove the aforementioned Formulas (4.1) and (4.2) hold.

**Theorem 1** The $CM_F$ model synthesized by our proposed approach satisfies the formula $\text{Deadlock}(SM_1|\cdots|SM_N|CM_F) = \text{False}$.

**Proof)** By contradiction, let us assume that the interaction model $SM_1|\cdots|SM_N|CM_F$ is not deadlock-free and hence it has a deadlock state $s'$. It means that there exists a deadlock trace $\mu$ in the interaction model such that $\mu = \mu_1\mu_2\ldots\mu_n \wedge \mu_n = m(i,j) \wedge s \xrightarrow{m(i,j)} s'$. According to Definition 7, in the $CM_F$ model, there must exist a deadlock trace $\mu^A$ such that $\mu^A = \mu^A_1\mu^A_2\ldots\mu^A_{2n} \wedge \mu^A_{2n-1} = ?m(i) \wedge \mu^A_{2n} = !m(j) \wedge s \xrightarrow{?m(i)} s^* \xrightarrow{!m(j)} s'$.

According to Algorithm 2, the $CM_D$ model is deadlock-free because the deadlock traces of it are eliminated by using Step 2 in Algorithm 2 and we can easily conclude that $CM_F$ model is also deadlock-free due to the computation formula $CM_F$=SP($CM_D$,RM) in Step 4.1 in Algorithm 2 .

This contradicts the hypothesis that the $CM_F$ model has a deadlock trace $\mu^A$, and hence the proof is given.

**Theorem 2** The $CM_F$ model synthesized by our proposed approach satisfies the formula $(SM_1|\ldots|SM_N|CM_F) \models RM$.

**Proof)** Also by contradiction, we suppose that $(SM_1|\ldots|SM_N|CM_F) \models RM$ does not hold, so there must exist a trace $\mu^C$ in the interaction model $SM_1|\cdots|SM_N|CM_F$ such that $\mu^C = m_1(i_1,j_1),\ldots m_r(i_r,j_r) \wedge s_0 \xrightarrow{m_1(i_1,j_1)} s_1 \ldots, s_{r-1} \xrightarrow{m_r(i_r,j_r)} s_r$, which has no equivalent trace in RM model.

According to Definition 7, there must exist a deadlock trace $\mu^A$ in $CM_F$ model such that $\mu^A = ?m_1(i_1), !m_1(j_1), \ldots, ?m_r(i_r), !m_r(j_r) \wedge s_0 \xrightarrow{?m_1(i_1)} s^*_0 \xrightarrow{!m_1(j_1)} s_1, \ldots\ldots, s_{r-1} \xrightarrow{?m_r(i_r)} s^*_{r-1} \xrightarrow{!m_r(j_r)} s_r$. Since $CM_F$ model is produced by the formula $CM_F$=SP($CM_D$,RM) specified by

Step 4.1 of Algorithm 2, we can conclude that there exists a trace $\mu^{RM}$ in RM model which is trace equivalent to the trace $\mu^A$, i.e., the formula $\mu^A \approx \mu^{RM}$ holds.

Since two successive labeled actions $?m(i)$ and $!m(j)$ in $CM_F$ model correspond to the labeled message $m(i, j)$ in the interaction model $SM_1 |\cdots |SM_N |CM_F$ according to Definition 7, we can conclude the trace $\mu^C$ is trace equivalent to the trace $\mu^A$. Due to the above conclusion $\mu^A \approx \mu^{RM}$, we can conclude that the trace $\mu^C$ has the equivalent trace $\mu^{RM}$ in RM model.

This contradicts the hypothesis that the trace $\mu^C$ has no the equivalent trace in RM model and, hence, the proof is given.

To sum up, the correctness of our synthesis approach is proved according to Theorem 1 and Theorem 2. This means that the composite service automatically synthesizing by our proposed approach is guaranteed to satisfy the correctness constraints of being a correct composite service, i.e., deadlock-freeness and requirement satisfaction.

## 5. Related Work

A large number of research work has been developed for synthesis of composite services to address service composition issue by using formal methods such as process algebras (see, e.g, [3], [4]), automata (see, e.g, [5], [6]) or Petri Net (see, e.g, [7], [8]). For example, in [3], the authors present a BP-calculus, a $\pi$-like calculus and define specific equivalence relations. Based on the theoretical basis, BPEL implementation of a composite service is verified through formal specification expressed in this calculus to ensure its correctness. In [5], the authors discuss synchronizability and realizability analysis to check whether it is possible to automatically verify correctness of composite services in asynchronous messages based on guarded automata model. In [8], the authors present a class of Petri nets called workflow nets. For this class of Petri nets, a verification tool named Wolfan has been developed. This tool can verify the properties of composite services described by BPEL processes.

However, the problem faced by all these approaches is requiring developers to provide a detailed composite service model in advance. This makes the synthesis process of composite services semi-automatic, complex and inefficient. In our approach, developers need not provide any prior model of a composite service. BPEL implementation of the composite service can be automatically derived from a set of component services exposing interaction protocols, and a given composition requirement expressing temporal constraints on the service interactions.

[11] proposes a methodology for automatic generation of an adaptor to ensure that interactions between two component services via the adaptor is deadlock-free, by using an intermediate workflow language YAWL. In [12], the authors present a runtime adaptation approach to ensure deadlock-freeness of the adapted service composition in their execution process. It relies on an event-based technology which triggers a specific adaptation action when catching some event leading to a deadlock. These Approaches focus on automatic synthesis of a composite service called "adaptor" to ensure the deadlock-free interactions with component services [8], [9]. However, they do not take into account composition requirements and hence the composite service synthesized by this kind of approaches is not ensured to be correct, while our approach goes beyond ensuring deadlock-freeness by also considering the aspect of requirement satisfaction and the composite service synthesized by our approach is guaranteed to be correct.

In [13], the authors use AI planning techniques to address the problem of automatic service discovery and synthesis of composite services in semantic web services. While in our approach we assume that component services are already discovered and thus we do not address the problem of service discovery. Besides, we tackle automatic synthesis of composite services that is more complex than the one considered in [13]. Indeed, [13] does not take into account stateful web service, like our approach does with abstract BPEL processes. This is the case in [14], [15], which support simple service compositions starting from WSDL-like specifications of web services.

In [16], each component service is formally specified as a transition system and the composition requirement can be viewed as a target service, also described itself as a transition system. The approach is to synthesize a composite service realizing the target service. In contrast to our approach, there is one main assumption to deal with deadlocks: in order to automatically synthesize a composite service which also ensures the deadlock-freeness aspect, they need to consider a specification of the deadlocking interaction that causes deadlocks. This is a problem because the developers might not know the deadlocking interactions since they might be unpredictable.

## 6. Conclusions

In this paper, we have presented a correctness assurance approach for automatically synthesizing a composite service with services exposing interaction protocols, and with composition requirements expressing temporal constraints on the service interactions. Our approach is based on the modeling that is able to capture the interaction protocols of services. For what concerns the specification of composition requirements, the approach also provides formal models for specifying temporal constraints on the service interactions. Given a set of abstract BPEL processes describing interaction protocols of existing services and a composition requirement expressing temporal constraints on their interactions, our approach can automatically generate the BPEL implementation of the composite service.

The composite service generated by utilizing our proposed approach is guaranteed to be correct and unlike existing approaches, it does not require developers to provide a prior composite service model and perform formal verifications on it. The correctness of our approach is proved, and moreover, the feasibility of it is validated through a case

analysis throughout the paper.

In our approach, we focused mainly on the level of service interactions. In future work, we plan to extend our work to incorporate it with exchange data flows, i.e., with services exposing interaction protocols and exchanging data, and with composition requirements expressing complex constraints not only on the service interactions but also on the exchanged data.

**References**

[1] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility verification for Web service choreography," Proc. ICWS, 2004, pp.738–741, 2004.
[2] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "LTSA-WS: A tool for model-based verification of web service compositions and choreography," Proc. ICSE, 2006, pp.771–774, 2006.
[3] F. Abouzaid and J. Mullins, "Model-checking Web services orchestrations using BP-calculus," Electronic Notes in Theoretical Computer Science, vol.255, pp.3–21, 2009.
[4] G. Tremblay and J. Chae, "Towards specifying contracts and protocols for web services," Proc. MCETECH, 2005, pp.73–85, 2005.
[5] T. Bultan, J. Su, and X. Fu, "Analyzing conversations of Web services," IEEE Internet Computing, vol.10, no.1, pp.18–25, 2006.
[6] M. Emilia Cambronero, G. Díaz, V. Valero, and E. Martínez, "Validation and verification of Web services choreographies by using timed automata," J. Logic and Algebraic Programming, vol.80, no.1, pp.25–49, 2011.
[7] P. Sun, C.J. Jiang, and M.C. Zhou, "Interactive Web service composition based on Petri net," Trans. Institute of Measurement and Control, vol.33, no.1, pp.116–132, 2011.
[8] H.M.W. Verbeek and W.M.P. van der Aalst, "Analyzing BPEL processes using Petri nets," Proc. PNCWB, 2005, pp.59–78, 2005.
[9] M. Dumas, B. Benatallah, and H.R. Motahari Nezhad, "Web service protocols: Compatibility and adaptation," IEEE Data Eng. Bull, vol.31, no.3, pp.40–44, 2008.
[10] W.M.P. Van der Aalst, A.J. Mooij, and S. Christian, "Service interaction: Patterns, formalization, and analysis," Proc. SFM, 2009, pp.42–88, 2009.
[11] B. Benatallah, F. Casati, and D, Grigori, "Developing adapters for Web services integration," Proc. CAiSE, 2005, pp.415–429, 2005.
[12] Y. Taher, A. Ait-Bachir, and M.C. Fauvet, "Diagnosing incompatibilities in Web service interactions for automatic generation of adapters," Proc. AINA, 2009, pp.652–659, 2009.
[13] S. McIlraith and S. Son, "Adapting golog for composition of semantic Web services," Proc. KR, 2002, pp.482–496, 2002.
[14] D. Skogan, R. Gronmo, and I. Solheim, "Web service compositionin UML," Proc. EDOC, 2004, pp.47–57, 2004.
[15] K. Pu, V. Hristidis, and N. Koudas, "Syntactic rule based approach to Web service composition," Proc. ICDE, 2006, pp.31–40, 2006.
[16] D. Calvanese, G.D. Giacomo, and M. Lenzerini, "Automatic service composition and synthesis: The roman model," IEEE Data Eng. Bull., vol.31, no.3, pp.18–22, 2008.
[17] M. Pistore, P. Traverso, P. Bertoli, and A. Marconi, "Automated synthesis of composite BPEL4WS web services," Proc. ICWS, 2005, pp.293–301, 2005.
[18] R. Kazhamiakin, M. Pistore, and L. Santuari, "Asynchronous timed Web service-aware choreography analysis," Proc. CAiSE, 2009, pp.364–378, 2009.
[19] M. Autili, P. Pelliccione, and P. Inverardi, "Graphical scenarios for specifying temporal properties," J. Automated Software Engineering, vol.14, no.3, pp.293–340, 2007.
[20] P. Gastin and D. Oddoux, "Fast LTL to Büchi automata translation," Proc. CAV, 2001, pp.53–65, 2001.
[21] E.M. Clarke, O. Grumberg, and D.A. Peled, Model Checking, Massachusetts Institute of Technology, The MIT Press, Cambridge, 1999.
[22] B. Alpern and F.B. Schneider, "Verifying temporal properties without temporal logic," ACM Trans. Programming Languages and Systems, vol.11, no.1, pp.147–167, 1989.

**Dajuan Fan** is currently a Ph.D. candidate in the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. Her main research interests include software engineering, formal methods, and service-oriented computing.

**Zhiqiu Huang** is currently a Ph.D. professor, Ph.D. supervisor and the head of the College of Computer Science and Technology at Nanjing University of Aeronautics and Astronautics. He is a senior member of IEEE. His main research interests include software engineering, formal methods, service-oriented computing and knowledge engineering.

**Lei Tang** is currently a Ph.D. candidate in the College of Electronics and Information Engineering, at Nanjing University of Aeronautics and Astronautics. Her main research interests include software engineering and concurrency theory.