A Petri Net Approach to Mediation-Aided Composition of Web Services

Yanhua Du, Xitong Li, and PengCheng Xiong

***Abstract*—Recently, mediation-aided composition has been widely adopted when dealing with incompatibilities of services. However, existing approaches suffer from state space explosion in compatibility verification and cannot automatically generate the BPEL code. This paper presents a Petri net approach to mediation-aided composition of Web services. First, services are modeled as open WorkFlow Nets (*oWFNs*) and are composed using mediation transitions (*MTs*). Second, the modular reachability graph (*MRG*) of composition is automatically constructed and the compatibility is analyzed, so that the problem of state space explosion is significantly alleviated. Furthermore, an Event-Condition-Action (*ECA*) rule-based technique is developed to automatically generate the BPEL code of the composition, which can significantly save the time and labor of designers. Finally, the prototype system has been developed.**

***Note to Practitioners*—Web services are an emerging area for business process automation. This work presents a novel Petri net approach to mediation-aided composition of Web services. The proposed approach can greatly alleviate state space explosion to automatically verify the composition of partially incompatible services, and significantly save the time and labor of designers to obtain BPEL code. It consists of three phases: modeling composition of Web services, automatic verification of composition, and automatic generation of BPEL code. The prototype system has been developed based on the open source software PIPE and validated in a real-life case study. It is ready to be applied in industrial Web service composition for business automation.**

***Index Terms*—Compatibility verification, mediation transition, mediation-aided composition, modular reachability graph, prototype system.**

I. Introduction

Service-Oriented Architecture (SOA) is becoming one of the main computing paradigms for designing complex business applications [1], [2]. Usually, a business application is not realized by a single Web service but a set of them. Composition, in which multiple independent Web services are assembled to accomplish a more complex task, is one of the key motivations to embrace Web service technology [2]–[6].

According to whether or not the participating Web services (abbreviated to services in the rest of this paper) in the composition are exactly compatible, service composition can be divided into direct composition and mediation-aided one.

Various direct composition methods have been proposed, including planning based [1], logical inference driven [2], Petri net based [3], [4], automata based [5], quality-of-service (QoS) optimizing based [6], etc. These methods [1]–[6] assume both data formats and sequences of the messages are consistent. However, services are not always exactly compatible in real-life composition situations. Usually, two (or more) services providing complementary functionality could be linked together in principle, but cannot be directly composed because of partially compatible interfaces or interaction patterns.

Mediation-aided composition [7]–[15] is attracting more attention, which mainly uses a set of mediators/adaptors to glue two or more partially compatible services. Compatibility verification is a crucial task of mediation-aided composition which is used to check whether there exist mediators to glue two partially compatible services [7]–[15]. Another important task of mediation-aided composition is to automatically generate the (abstract) BPEL code of composition, since BPEL has become the industrial standard for modeling service composition. This can significantly save the time and labor of designers with the fast changing need.

Existing work [7]–[15] has not fully investigated the issue of mediation-aided composition, because they suffer from state space explosion in compatibility verification and cannot automatically generate the (abstract) BPEL code. In this paper, a Petri net approach to mediation-aided composition of services is presented. First, services are modeled using open WorkFlow Nets (*oWFNs*) [16], and are composed by adding mediation transitions (*MTs*). Second, the composition compatibility is verified by automatically constructing and analyzing the modular reachability graph (*MRG*) [17] of composition. Finally, if the composition is verified to be valid, the BPEL code of the composition is automatically generated in an Event-Condition-Action (*ECA*) rule-based way [18].

Compared with the existing work [7]–[15], the contributions of this paper are as follows.

1) *oWFNs* of services are composed by using three basic kinds of *MTs* to address the problem of their partially compatible interfaces or interaction patterns.
2) By automatically constructing and analyzing the *MRG* of composition, our approach can significantly alleviate state space explosion without unfolding to ordinary state space.
3) Once the composition is verified to be valid, its BPEL code is automatically generated in the format of *ECA* rule, which can significantly save the time and labor of designers with the fast changing need.
4) The prototype system based on the open source software Platform Independent Petri net Editor (PIPE) has been developed.

Note that our approach is an offline one. Once the composition is launched, no runtime reconfigurations are possible, e.g., services cannot be replaced or reconfigured during execution, because our approach assumes that the message mappings among services to be composed are specified by designers and should be accurate and faultless.

The rest of this paper is organized as follows. Section II presents the composition of *oWFNs* by adding *MTs*. Section III presents how to automatically construct the *MRG* and analyze its compatibility. Section IV discusses automatically generation BPEL code from composition. Section V presents a prototype system. Section VI discusses related work and Section VII concludes this paper.

II. Modeling Mediation-Aided Composition of Services

In this section, first, the formal concept of open WorkFlow Net (*oWFN*) is introduced. Second, mediation transitions (*MTs*) between *oWFNs* are presented. Finally, the composition procedure of *oWFNs* based on *MTs* is presented.

Manuscript received August 08, 2010; revised July 22, 2011; accepted January 26, 2012. Date of publication March 06, 2012; date of current version April 03, 2012. This paper was recommended for publication by Associate Editor A. Colombo and Editor Y. Narahari fupon evaluation of the reviewers' comments. This work was supported in part by the National Natural Science Foundation of China under Grant 61004109.

Y. Du is with the School of Mechanical Engineering, University of Science and Technology Beijing, Beijing 100083, China (e-mail: duyanhua@ustb.edu.cn).

X. Li is with the MIT Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA 02142 USA (e-mail: xitongli@mit.edu).

P. Xiong is with the College of Computing, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: xiong@gatech.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TASE.2012.2188511

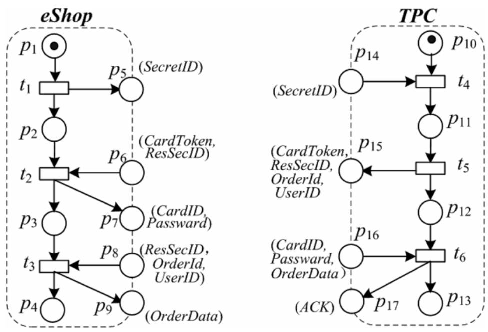

Fig. 1. Two *oWFNs* of eShop and TPC services.

A. Open Workflow Net (oWFN)

Existing service composition specification languages such as Business Process Execution Language (BPEL), Web Service Choreography Interface (WSCI), and Web Service Choreography Description Language (WS-CDL) all provide mechanisms to compose services by specifying message sent or received by interfaces [9]. Among these various languages, BPEL has become dominant because it has been proposed by OASIS as an industry standard [7]–[10] and is supported by major software companies such as IBM, Oracle, and SAP. In this paper, BPEL is assumed as the language for describing the internal logic of services and the final composition of them. By doing so, our theoretical approach becomes practical and can be used to address real-world services in practice.

To formally analyze the composition compatibility of BPEL services, we first model them based on *oWFN*. As a special class of Petri nets [7]–[10], [19]. *oWFN* [16] is generalized from the classical WorkFlow Net (*WFN*) [7]–[10] by introducing the interfaces for exchanging messages.

Definition 1. (Open WorkFlow Net, oWFN): An *oWFN* is a $6\text{-tuple} = (P, T, F, IP, OP, F')$, where:

1) $\{P, T, F\}$ is a *WFN*;
2) *IP* is the set of input message places, and $\forall\, x \in IP, {}^\bullet x = \varnothing$;
3) *OP* is the set of output message places, and $\forall\, x \in OP, x^\bullet = \varnothing$, $IP \cap \mathrm{OP} = \varnothing$;
4) $F' \subset (T \times OP) \bigcup (IP \times T)$ is the set of interface arcs.

Assume there are two BPEL services to be composed: eShop service and a Third-Party Checkout service (TPC), which is the excerpt and adaptation of a real business scenario [20]. When buyers finish shopping and want to check out, eShop and TPC services need to be composed to fulfill the requirement of the online shopping and checkout business. To save the space of this paper, the detailed BPEL code of two services is omitted.

When buyers check out: 1) eShop service is initiated, and it invokes TPC service by sending message *SecretID* . 2) eShop service receives messages including data *CardToken* and *ResSecID* from TPC service, and sends synchronously data *CardID* and *Passward* to TPC service as a message. 3) eShop service receives message composed of *ResSecID*, *OrderId* and *UserID* from TPC service, and replies asynchronously message *OrderData*.

On the other hand: 1) TPC service is initiated by receiving message *SecretID*. 2) TPC service invokes eShop service by sending message including data *CardToken* and *ResSecID*, then starts an asynchronous activity to receive message composed of *CardID*, *Passward* and *OrderData* from eShop service. 3) TPC replies the confirm message *Ack*, and displays the results.

According to the above descriptions, two services are models as *oWFNs* in Fig. 1.

B. Mediation Transition

Based on the Web Service Description Language (WSDL) specifications of messages exchanged of *oWFNs*, the message mappings between two them can be set.

Definition 2. (Message Mapping, MM): A message mapping *MM* between two *oWFNs* is expressed in the form of $<source, target>$, where *source* is the messages, or their parts/elements that need to be sent by an *oWFN*, and *target* is the messages, or their parts/elements that need to be receive by another one.

Source and *target* are expressed in the form of *Service.Message* or *Service.Message.Part*. In this paper, it is assumed that the *MMs* among services to be composed are specified by designers and the *MMs* should be accurate and faultless. The automatic generation of *MMs* is beyond the scope of this paper.

For superfluous message pattern, the *MM* between two *oWFNs* is $<source, \varnothing>$. The *MM* of missing message pattern is $<\varnothing, target>$. These patterns cannot affect the verification result of service composition and do not appear explicitly in the composition models. Superfluous messages can be discarded by designers, because they do not lead to deadlocks of the *oWFNs*. On the other hand, if the designers cannot provide the missing messages, then the composition is usually considered to be incompatible and do not need further verification.

Definition 3. (Message Place Mapping, MPM): A message place mapping *MPM* is transformed from a *MM* $<source, target>$ and expressed in the form of $<ps_s, ps_t>$, where ps_s is the set of output message places corresponding to messages (or their parts/elements) in *source* and ps_t is the set of input message places corresponding to messages (or their parts/elements) in *target*.

Not all of input or output message places of *oWFNs* need to appear in a *MPM*. For those redundant output messages that no *oWFN* accepts, their corresponding places will not appear in a *MPM*. For an *oWFN* with choice branches, if one path is not picked up by its partner *oWFN*, the corresponding input message place will also not appear.

Based on *MPMs*, we can derive mediation transitions to connect *oWFNs*, which serve as information channel by specifying the transferring relation of messages.

Definition 4. (Mediation Transition, MT): A mediation transition *MT* is a transition which has at least one input places and at least one output places in *oWFNs*.

In this paper, three basic kinds of *MTs* are defined as follows.

1) Forward Mediation Transition (*FMT*): A *FMT* stores the incoming message and forwards it to the receiver when needed. For a *MPM* $<ps_s, ps_t>$, where ps_s and ps_t all have only one message place, a *FMT* is used to connect the output message place in ps_s and the input message place in ps_t.
2) Merge Mediation Transition (*MMT*): A *MMT* collects the multiple source messages/parts/elements and then combines them into one single target message. For a *MPM* $<ps_s, ps_t>$, where ps_s has more than one message places and ps_t has only one, a *MMT* is used to connect the output message places in ps_s and the input message place in ps_t.
3) Split Mediation Transition (*SMT*): A *SMT* replicates the source message (or its part/element) into multiple copies. For a *MPM* $<ps_s, ps_t>$ where ps_s has only one message place and ps_t has more than ones, a *SMT* is used to connect the output message place in ps_s and the input message places of in ps_t.

Definition 5. (Composition of Open Workflow Nets by MTs, CoWFN): Suppose $oWFN_i (i = 1, 2, \ldots, n)$ models n services, a tuple $CoWFN = (oWFN_1, \ldots, oWFN_n, IT, FI)$ is called as *Composition of open WorkFlow Nets by MTs* if and only if:

1) *IT* is the set of *MTs* between *oWFNs*;
2) *FI* is the set of arcs between the IP_i/OP_i in $oWFN_i (i = 1, 2, \ldots, n)$ and *MTs*.

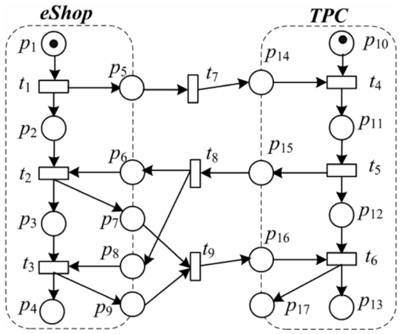

Fig. 2. The *CoWFN* of eShop and TPC services.

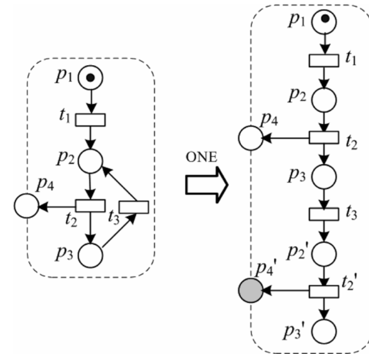

Fig. 4. Transformation of iterative structure in *oWFNs*.

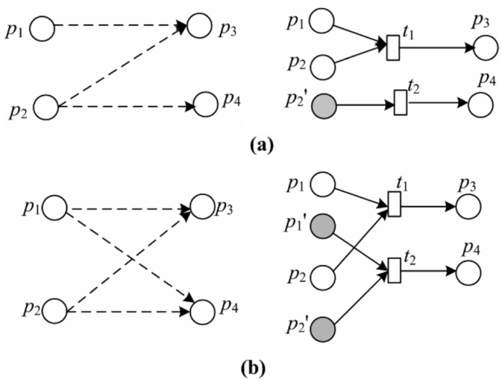

Fig. 3. Two examples of complex conditions.

Reconsidering the previous scenario, by analyzing the interfaces of eShop and TPC services in Fig. 1, the *MMs* are obtained as follows:

$<eShop.SecretID, TPC.SecretID>$,

$<TPC(CardToken, ResSecID, OrderID, UserID)$,

$eShop.(CardToken, ResSecID){+}eShop.(ResSecID,$

$OrderID, UserID)>$,

$<eShop.(CardID, Passward){+}eShop.OrderData,$

$TPC.(CardID, Passward, OrderData)>$

Based on the above *MMs*, we obtain three *MPMs*: $<p_5, p_{14}>$, $<p_{15}, p_6{+}p_8>$, and $<p_7{+}p_9, p_{16}>$. Then, *FMT* t_7, *SMT* t_8 and *MMT* t_9 are constructed, as shown in Fig. 2.

For the complex conditions that an output message place appears in more than one *MPM*, we cannot directly use the above basic *MTs*. The output message places for these *MPMs* need to be duplicated. This is do not affect analyzing of composition compatibility, because the corresponding tokens remains in the duplicated output message places are allowed for the compatibility of *oWFNs* (see Definition 9 in Section III).

By adding duplicated places, we assure that each output message place only exists in only one *MPM* so that complex conditions can be achieved by combining basic *MTs*.

For the example of the left of Fig. 3(a), the message of p_1 and a part of message p_2 are composed as the message p_3, and the rest of the part of message p_2 is used by the message of p_4. We get the *MPMs* $<p_1 + p_2, p_3>$ and $<p'_2, p_4>$. Here, a copy p'_2 of p_2 is inserted, and a *MMT* t_1 and *FMT* t_2 are constructed, as shown in the right of Fig. 3(a). Another example in the left of Fig. 3(b), its left *MPM* is divided into $\{<p_1{+}p_2, p_3>, <p'_1{+}p'_2, p_4>\}$ by duplicating p_1 and p_2. Subsequently, two *MMTs* t_1 and t_2 are constructed, as shown in the right of Fig. 3(b).

C. Mediation-Aided Composition of oWFNs

The procedure of mediation-aided composing *oWFNs* into a *CoWFN* is shown in the following Algorithm 1.

Different from mediators/adaptors in [7]–[15], *MTs* in our paper are high level and abstractive transitions, which hide unnecessary structural details irrelevant to the verification of composition. As conceptual mediators, *MTs* will be detailed implemented in the phase of automatically generating BPEL code of the composition.

Note that four kinds of basic control structures, namely, sequential, parallel, selective and iterative structures, have been defined in the Workflow Reference Model [7]–[10]. The *iterative structure* occurs when some transitions are executed iteratively. If *oWFNs* do not contain iterative structures, we can directly use Algorithm 1. Otherwise, we approximate the number of loops in a finite iterative structure and transform it to a sequence of transition by expanding cycles [23]. The input/output message places linked by transitions in an iterative control structure should also be duplicated into several "copied" places in the transformed model. As depicted in Fig. 4, a copy p'_4 of p_4 is inserted because the iterative structure is executed only one time.

Algorithm 1: Mediation-aided compose *oWFNs*

Input: *oWFNs*

Output: *CoWFN*

Step 1: According to WSDL specifications of exchanged messages among *oWFNs*, the *MMs* are set.

Step 2: For each *MM* $<source, target>$, a *MPM* are obtained by getting the place name of messages in $source/target$, denoted by $<ps_s, ps_t>$.

Step 3: If each output message place of all *oWFNs* exist in only one *MPM*, then go to Step 5.

Step 4: Assume an output message place *op* appears in more than one *MPM*s, e.g., Mp_a, $Mp_{a+1}, \ldots$, and Mp_b, op is duplicated into $op'_a, op'_{a+1}, \ldots, op'_{b-1}$, and set ${}^\bullet op'_x = {}^\bullet\ op\ (a{\le}x{\le}b\text{-}1)$. Then, ${}^\bullet op'_x(a{\le}x{\le}b\text{-}1)$ is used to replace op in $Mp_x(a{\le}x{\le}b\text{-}1)$.

Step 5: According to the *MPMs*, construct the *MTs* among *oWFNs*.

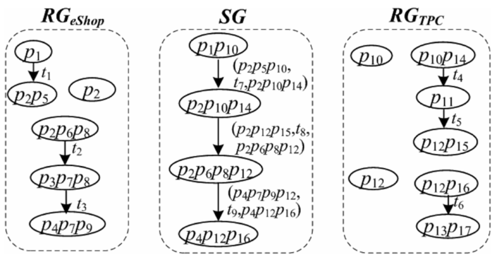

Fig. 5. The *MRG* of *CoWFN* in Fig. 2.

III. AUTOMATIC VERIFICATION OF COMPATIBILITY

Firstl, the concept of modular reachability graph (*MRG*) and its constructing procedure are presented in this section. Then, how to verify the compatibility of services by analyzing the *MRG* is discussed.

A. Modular Reachability Graph

First, a *CoWFN* is divided into a set of fragments (*oWFNMs*).

Definition 6. (Open Workflow Net With MTs, oWFNM): Given $oWFN = (P, T, F, IP, OP, F')$, a 3-tuple $oWFNM = (oWFN, C, F'')$ is called as *open Workflow Net with MTs* if and only if:

1) C is the set of *MTs*; given $\forall\, c \in C$, ${}^\bullet c = \varnothing$ or $c^\bullet = \varnothing$;
2) $F'' \subset (OP \times C) \bigcup (C \times IP)$ is the set of interface arcs.

The *CoWFN* of eShop and TPC services in Fig. 2 can be divided into two *oWFNMs*.

Intuitively, *MRG* is composed of one local reachability graph for each *oWFNM*, and a synchronization graph which captures their communications [17], [19].

Definition 7. (Modular Reachability Graph, MRG): Let $oWFNM_i$ $(i = 1, 2, \ldots n)$ be decomposed from a *CoWFN*. The modular reachability graph is a $(n + 1)$-tuple $MRG = (RG_1, \ldots, RG_n, SG)$, where:

1) $RG_i = (V_i, E_i, f_i)$, $(i = 1, 2, \ldots, n)$, is local reachability graph of $oWFNM_i$ in which: (a) (V_i, E_i) is a directed graph, where $V_i = R(M_{0i})$, and $E_i = \{(M_x, M_y) | M_x, M_y \in R(M_{0i}), \exists t_k \in T_i\text{: } M_x[t_k{>}M_y\}$; (b) f_i: $E_i \rightarrow T_i$ is a mapping from E_i to T_i in $oWFNM_i$, $f_i(M_x, M_y) = t_k$ iff $M_x[t_k{>}M_y$.
2) $SG = (V, E, f)$ is the synchronization graph among *oWFNMs*, in which: (a) (V, E) is a directed graph, $V \subseteq R(M_{01}) \times \cdots \times R(M_{0n})$; $E = \{(Mv_x, Mv_y) | Mv_x, Mv_y \in V, \exists t_f \in IT\text{: } Mv_x[t_f{>}Mv_y\}$; (b) f $:\rightarrow EIT$ is a mapping from E to the set of IT, $f(Mv_x, Mv_y) = t_j$ iff $Mv_x[t_j > Mv_y$.

A semiautomatically method of constructing *MRG* has been presented in [19], which needs the designers to manually decompose the composition into fragments. In this paper, we move one step forward to propose a full-automatically constructing and analyzing procedure from a *CoWFN*, which is shown in the following Algorithm 2.

First of all, we explain the operation of *marking projection* that will be used in the automatically constructing procedure.

Definition 8. (Marking Projection): The marking projection of $M_s = (p_x p_{x+1} \ldots p_y)$ in SG on RG_i is to remove the places from M that do not exist in the $oWFNM_i$, and is denoted with $\Gamma_i\ (M_s)$.

For the *CoWFN* in Fig. 2, we construct the $MRG = (RG_{eShop}, RG_{TPC}, SG)$ as shown in Fig. 5.

In order to prove of correctness of the Algorithm 2, we give the following Theorem 1.

Algorithm 2: Construct *MRG* from *CoWFN*

Input: *CoWFN*

Output: *MRG*

Step 1: For each MT t, we duplicate a set of copies, i.e., $Ct = \{t_1, t_2, \ldots, t_k\}$, in which k is the number of *oWFNs* of t connecting.

Step 2: For $i = 1$ to n:

For $\forall\, p_x \in IP_i$ of $oWFN_i$, if $p_x \in^\bullet t$, set p_x to be the input place of one transition t_s of Ct. If $p_x \in t^\bullet$, set p_x to be the output place of one transition t_s of Ct. Then, delete t_s from Ct.

Step 3: For each MT t, rename its copies $t_1, t_2, \ldots,$ and t_k to t.

Step 4: Construct each initial node M_{0i} in the $RG_i(1{\le}i{\le}n)$; then construct the initial node $(M_{01} \times \cdots \times M_{0n})$ for SG.

Step 5: For $i = 1$ to n:

Parallel compute and draw the nodes for each RG_i until no any enabled transitions in $oWFNM_i$.

Step 6: Get the set of enabled *MTs*, denoted as *ST*, and for each $t \in ST$:

Assume t is enabled by $M_i \times \cdots \times M_j$ that $M_i, \ldots, M_j$ are the markings of $RG_i, \ldots, RG_j$, the node $(M_i \times \cdots \times M_j)$ and arc $<(M_i \times \cdots \times M_j)[t{>}(M'_i \times \cdots \times M'_j)>$ are constructed in *SG*. Then, we construct the node of $\Gamma_i(M'_i \times \cdots \times M'_j)$ in the RG_i $(1{\le}i{\le}n)$, respectively.

Step 7: If there is not any possible enabled transition in *oWFNMs*, the Algorithm ends. Otherwise, go to Step 5.

Theorem 1: Assume $CoWFN = (oWFN_1, \ldots, oWFN_n, IT, FI)$ and its fragments denoted as $oWFNM_i(1{\le}i{\le}n)$, the *MRG* of *CoWFN* is isomorphic with its traditional reachability graph.

Its proof is very similar with [17], and is omitted because of the limit of paper space.

B. Analyzing Based on MRG

Definition 9. (Composition Compatibility of oWFNs): The *CoWFN* of $oWFN_i(i = 1, 2, \ldots, n)$ are regarded as compatible, iff its *MRG* satisfies the following cases:

1) For $\forall\, M \in RG_i(1{\le}i{\le}n)$, there exists a firing sequence σ and M_σ, satisfying $M[\sigma{>}M_\sigma$ and $M_\sigma{\ge}M_e$ (terminal marking).
2) Assume $M \in RG_i(1{\le}i{\le}n)$ and $M \ge M_e$, if $\exists p \in P$ satisfying $M(p){>}M_e(p)$, then $p \in IP_1 \cup OP_2 \ldots IP_n \cup OP_n$.

Before presenting the automatically analyzing procedure of compatibility by *MRG*, as shown in the following Algorithm 3, we explain the operation of *cross-product of marking sets* that will be used in it.

Definition 10. (Cross-Product of Marking Sets): Given two marking sets: $S_1 = \{M_{i1}, \ldots, M_{ix}\}$ in RG_i and $S_2 = \{M_{j1}, \ldots, M_{jy}\}$ in RG_j, The cross-product of them is to combine all possible the markings from S_1 and S_2, and is denoted with $\prod(S_1, S_2)$.

For the example of *MRG* in Fig. 5, we conclude that eShop and TPC services are compatible by Algorithm 3.

TABLE I
PERFORMANCE COMPARISON WITH TRADITIONAL APPROACH

Specification of cases	Traditional approach	Our approach
Two *oWFNs* with 12 places, 8 transitions and 1 *MT*.	15 nodes, 29 arcs	11 nodes, 12 arcs
Three *oWFNs* with 20 places,13 transitions and 2 *MTs*.	81 nodes, 231 arcs	22 nodes, 17 arcs
Four *oWFNs* with 29 places, 18 transitions and 3 *MTs*.	289 nodes, 2101 arcs	34 nodes, 25 arcs
Four *oWFNs* with 31 places, 20 transitions and 3 *MTs*.	387 nodes, 2300 arcs	35 nodes, 27 arcs
Five *oWFNs* with 40 places, 23 transitions and 4 *MTs*.	1497 nodes, 12281 arcs	49 nodes, 38 arcs

Algorithm 3: Analyze the compatibility by *MRG*

Input: *MRG*

Output: Results of compatibility

Step 1: For each $\forall\ M_s \in SG$, gets its $\Gamma_i(M_s)$ on $oWFNM_i(1 \le i \le n)$.

Step 2: For $i = 1$ to n:

Take all possible projection markings on $oWFNM_i$, and denote it as the set MS_i.

Step 3: Construct the cross-product set denoted as *CMS* by $\prod(MS_1, MS_2, \ldots, MS_n)$, and remove the product markings from *CMS* which are labeled on *SG* arcs.

Step 4: For each $M_{\rm cs} \in CMS$, gets its projection marking $\Gamma_i(M_{\rm cs})$ on $oWFNM_i(1 \le i \le n)$ denoted as the set TMS_i.

Step 5: If $\forall\ M_t \in TMS_i(1 \le i \le n)$ satisfies the following cases, the composition is compatible:

1) $M_t \in TMS_i$ is the logical state M_e of $oWFN_i$; and

2) If $\exists p \in P$ satisfying $M_t(p) > Me(p)$, then $p \in IP_1 \cup OP_2 \ldots IP_n \cup OP_n$.

Compared with traditional approach [7]–[10] that analyze the ordinary state space, our approach based on *MRG* can effectively mitigate state explosion without unfolding to the ordinary state space. In the worst case where there exist no transition in *oWFNs*, the *SG* in *MRG* will be identical with the ordinary state graph [7]–[10], i.e., having the same number of nodes and arcs. In the best case where there is no *MT* at all, the *SG* contains no nodes and arcs, and each RG_i of $oWFNM_i$ is identical to the ordinary state graph. In a real application, there are often some *MTs* and thus we can expect it can always outperform the traditional approach.

In order to quantitatively analyzing performance of our approach, we compare it with the ordinary state graph approach by several test cases in Table I. As we can see, our approach dramatically decreases the complexity of state space. Especially, with the increasing number of services increasing and the augmenting complexity of models, our approach is more efficient than the traditional approach.

IV. AUTOMATIC GENERATION OF BPEL CODE

After verifying the composition compatibility, in this section, how to automatically generate the BPEL code of composition to support its execution is discussed.

ECA rule offers flexible, adaptive and human-readable advantages to realizing processes in SOA environments [18], so that the BPEL code in the format of *ECA* rule will significantly save the designers time and labor. First, *ECA* rule is used to translate the *MTs* into BPEL code blocks as follows.

1) *FMT*: Preconditions of *FMT t* is the event of $\text{completion}(p)$, which p is the input message place of t and corresponds to activity $<invoke>$ or $<reply>$. Activity $<onMessage>$ should consume these events and receives activities, referring to the corresponding *partner link*, *port type* and *operation*. Then, activities $<assign>$ and $<copy>$ perform the transferring and transforming of messages between services. For example, t_7 in Fig. 2, its BPEL code is following.
2) *MMT*: This case is similar with a *FMT*, except that we orderly use more than one $<onMessage>$ to specify the occurrences for the events corresponding to preconditions.
3) *SMT*: This case is similar with a *FMT*, except that we orderly use more than one pair of $<assign>$ and $<copy>$ to perform the transferring and transforming of messages.

```
<!– completion(p5)–>
<onMessage partnerLink =
“...” portType=“...” operation = “...”...>
<assign>
<copy>
<from variable = “...”/> <to variable = “...”/>
</copy>
</assign>
<!—invoke port of p14–>
<invoke partnerLink = “...” portType=“...” operation =
“...”...>
</invoke>
</onMessage>
```

Then, the rest of a *CoWFN* is treated like a state machine and translated into event handlers, which mimic the pre and postconditions of each activity by sending messages to them.

Algorithm 4: Transform to BPEL code

Input: *CoWFN*

Output: The BPEL code of *CoWFN*

Step 1: Services are defined as the partner links by activity $<partnerLink>$. Then, the variables for composition are also defined by activity $<variables>$ according to the messages of the WSDL description of services.

Step 2: According to the kinds of *MTs*, we construct the BPEL code for them in the format of *ECA* rule. Then, the code of all *MTs* are embraced by $<eventHandlers>$ and $</eventHandlers>$.

Step 3: For the some copied output places in the *CoWFN*, complete events are defined according to needs by activity $<invoke>$, which do not carry any data.

Step 4: If there exist a $MPM<\varnothing, target>$ in the procedure of composition, we supplement the corresponding messages (parts or elements) between the pair of $<assign>$ and $<copy>$ to perform the transferring or transforming of messages.

Step 5: For each service, we use activity $<while>$ to check whether or not the *ECA* rules (the code of *MTs*) have been triggered. If

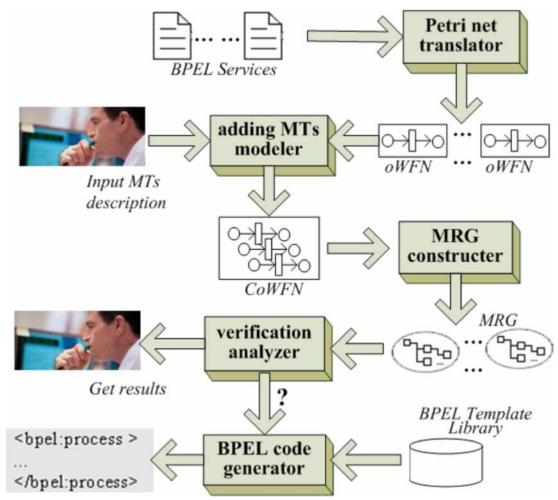

Fig. 6. The prototype system- *MCSS*.

true, to perform the interface operations by activities $<receive>$, $<invoke>$ or $<reply>$.

Step 6: The BPEL code blocks of each service is embraced by $<process>$ and $</process>$, in order to make services execute concurrently.

Step 7: To specify the *CoWFN* receiving the initial message that starts it, the initial activity $<receive>$ is added with *createInstance* attribute. Then, the BPEL code of *CoWFN* is obtained by putting all the above code blocks into $<process>$ and $</process>$.

Note that the generated BPEL code lack specific implementation details which cannot be automatically derived from a *CoWFN* by the above Algorithm 4, so the code needs further refinement.

V. Prototype System

In this section, a prototype system (Mediation-aided Composition System of Services, *MCSS*) implementing the proposed approach is presented.

As shown in Fig. 6, *MCSS* has five components: *Petri net translator*, *adding MTs modeler*, *MRG constructer*, *verification analyzer*, and *BPEL code generator*. *Petri net translator* adopts the free software *BPEL2oWFN* [21] and the other four modules are developed based on the open-source software PIPE [22]. The original PIPE uses the Model-Controller-View architecture pattern to implement several Petri net analysis plug-in modules. Based on the existing architecture of PIPE, the modules in *MCSS* (*adding MTs modeler*, *MRG constructer*, *verification analyzer*, and *BPEL code generator*) can be developed quickly.

Petri net translator is responsible for translating BPEL services into *oWFNs* in the file format of Petri Net Markup Language (PNML). We adopt the existing free software tool *BPEL2oWFN*. The output PNML files of *oWFNs* are input into *adding MTs modeler*. Then, *adding MTs modeler* composes the *oWFNs* into a *CoWFN*. It merges the PNML files of *oWFNs* as the whole file of a *CoWFN* by adding XML sections of *MTs*. The designers can manually add *MTs* based on the WSDL descriptions of *oWFNs*.

The whole PNML file of *CoWFN* is the input of *MRG constructer*. *MRG constructer* serves as the engine to automatically decompose a *CoWFN* into some *oWFNMs* in the background through manipulating its PNML file. The PNML files of *oWFNMs* are put into the fragment pool and are used to support verifying the compatibility of composition

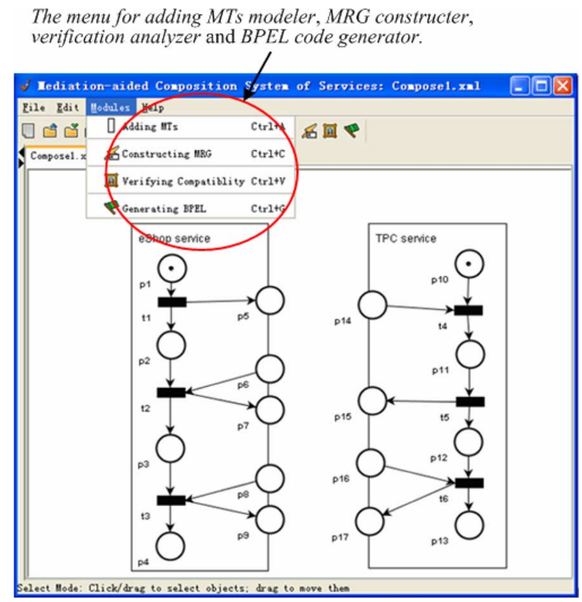

Fig. 7. The main interface of *MCSS*.

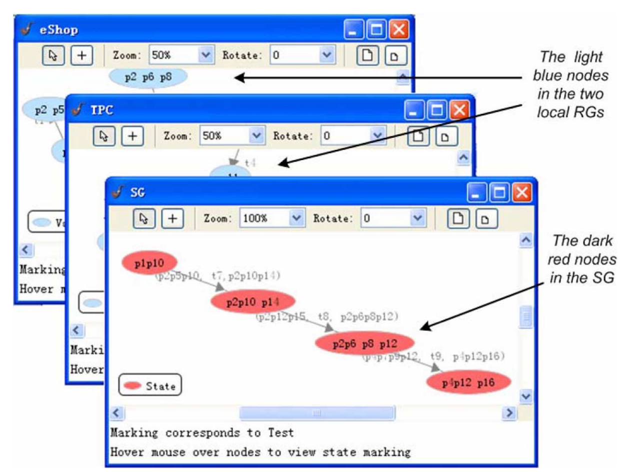

Fig. 8. The screenshot of *MRG constructer*.

based on *MRG*. In order to intuitively illustrate the *MRG*, *MCSS* return the graphic results to the designers.

Based on the above *MRG*, *verification analyzer* determines whether the composition is compatible. If the compatibility is validated, *BPEL code generator* is enabled for the designers to get the BPEL code of composed service. Otherwise, the error messages are returned.

BPEL code generator can automatically generate the BPEL code from *CoWFN* in the format of *ECA* rules. All code are implemented and stored in the BPEL template library. Here,the library includes the predefined BPEL template for three kinds of *MTs*, and message variables for composition according to their message formats.

In order to illustrate the executing procedure of *MCSS*, we adopt the example of the eShop and TPC services described in Section II. As shown in the Fig. 7, the *oWFNs* of eShop and TPC services are output by *Petri net translator*. Based on *adding MTs modeler*, the designers can add *MTs* t_7, t_8 and t_9, and obtain their *CoWFN*.

Next, *MRG constructer* automatically generates the *MRG* in the background and demonstrates the *MRG* in the graphic format as shown in Fig. 8, where the node color of *RGs* for eShop and TPC is light blue, while the node color of *SG* is dark red.

Afterwards, *verification analyzer* returns the result that the composition is compatible and the BPEL code generator is enabled. Finally, the whole BPEL code of the example from the *CoWFN* is automatically generated by the *BPEL code generator*.

Because of the limit of paper length, the screenshots of other components and the whole BPEL process of example are not shown in this paper.

VI. Related Work

There are some related studies [7]–[15] on the issue of compatibility verification and automatic generation of BPEL code in mediation-aided composition.

A. Compatibility Verification

Mooij and Voorhoeve [7] address the automated generation of adapters based on the *open WorkFlow Net* (same with our work in this paper). Then, correctness of adapters is verified by proving the properties (e.g., deadlock-freedom) holding for the composition of services. Wang [8] proposes a visual language for specifying adaptors for services and use Petri nets to check their correctness. Tan *et al.* [9] transform the BPEL services into service workflow nets (a special class of Petri nets) and analyze the compatibility of two services based on mediators. Based on Colored Petri Nets (CPNs), Li *et al.* [10] present a heuristic approach to identify the protocol mismatches of services and select appropriate mediator patterns. Then, if the composition is verified, the BPEL templates of mediator patterns are also developed. Guermouche *et al.* [11] model services are as automata models. Then, the mediator are generated to supply the missing messages which are required to complete the Cartesian product of automates. Simultaneously, the correctness of constructing adapters is verified. Also, based on automata, Nezhad *et al.* [12] present a method for identification of the split/merge class of interface mismatches and a semiautomated matching approach to construct an adaptor for these services. Bachir and Fauvet [13] check whether two services based on Finite State Machines (FSMs) are incompatible, and provides the locations in the service interfaces where these incompatibilities occur. So that mediators can be constructed subsequently. Zhou [14] obtains abstract protocols from service protocols by a set of rules. Then, they construct adaptation matrix, using an adapted depth-first search with back tracking technique, so that the conditions that these two services can be adapted is identified. Canal *et al.* [15] present an approach to automatically generate adaptor based on Labeled Transition Systems (LTSs). Along with adapters is constructed, their correctness is also verified.

The above methods [7]–[15] can address the issue of compatibility verification for a small number of services. However, they all suffer from the problem of state space explosion when verifying the composition of complex services, especially the number of services is large.

B. Automatic Generation of BPEL Code

The abovementioned methods [7]–[9], [11]–[15] only propose conceptual mediators, e.g., the function of mediators such as how to transfer or split messages, and cannot produce the (abstract) BPEL code from composition models to support execution. Li *et al.* [10] present the BPEL templates of several mediator patterns. However, they all do not consider how to automatically generate the whole BPEL code from the composition models.

VII. Conclusion

The mediation-aided composition is attracting more attention when dealing with incompatibilities of services. But existing approaches suffer from the problem of state space explosion and cannot automatically generate the BPEL code. The approach presented in this paper has addressed the weaknesses of prior ones. Specifically, the compatibility of composition is verified by automatically constructing and analyzing its *MRG*, which can alleviate the state space explosion problem. If the composition is verified to be valid, the BPEL code is automatically generated, which can significantly save the time and labor of designers with the fast changing need.

In the future, we plan to extend our approach in two aspects.

1) The present approach needs designers to give *MMs* based on service WSDL specifications. We will explore how to adopt semantic technology to automatically generate *MMs*.
2) In a practical business environment, some service composition often has timed specification (or temporal constraints) [23]. We will extend *MRG* to express the timed state space.

References

[1] M. Carman, L. Serafini, and P. Traverso, "Web service composition as planning," in *Proc. Int. Conf. ICAPS*, 2003, pp. 45–51.
[2] X. Tang, C. Jiang, and M. Z. Zhou, "Automatic Web service composition based on Horn clauses and Petri nets," *Expert Systems With Applications*, vol. 38, no. 10, pp. 13024–13031, 2011.
[3] R. Hamadi and B. Benatallah, "A Petri net-based model for Web service composition," in *Proc. Int. Conf. ADC, 2003*, pp. 191–200.
[4] J. M. Mendes, P. Leitão, F. Restivo, and A. W. Colombo, "Composition of Petri nets models in service-oriented industrial automation," in *Proc. 8th Int. Conf. INDIN*, 2010, pp. 578–583.
[5] H. Foster, S. Uchitel, J. Magee, and J. Kramer, "Compatibility verification for web service choreography," in *Proc. Int'l Conf. ICWS*, 2004, pp. 738–741.
[6] P. Xiong, Y. Fan, and M. Z. Zhou, "Web service configuration under multiple quality-of-service attribute," *IEEE Transactions on Automation Science and Engineering*, vol. 6, no. 2, pp. 311–321, 2009.
[7] A. J. Mooij and M. Voorhoeve, "Proof techniques for adapter generation," in *Proc. 5th Int. Conf. WS-FM* , 2008, pp. 207–223.
[8] K. W. S. Wang, "Interface Adaptation for Conversational Services," Ph.D., Faculty of Information Technology, Queensland University of Technology., Brisbane, Australia, 2008.
[9] W. Tan, Y. S. Fan, and M. C. Zhou, "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.*, vol. 6, no. 1, pp. 94–106, 2009.
[10] X. Li, Y. Fan, S. Madnick, and Q. Z. Sheng, "A pattern-based approach to protocol mediation for web service composition," *Information and Software Technology*, vol. 52, no. 3, pp. 304–323, 2010.
[11] N. Guermouche, O. Perrin, and C. Ringeissen, "A mediator based approach for services composition," in *Proc. 6th Int. Conf. SERA*, 2008, pp. 273–280.
[12] H. R. M. Nezhad, G. Y. Xu, and B. Benatallah, "Protocol-aware matching of web service interfaces for adapter development," in *Proc. Int. Conf. WWW*, 2010, pp. 731–740.
[13] A. A. Bachir and M. C. Fauvet, "Diagnosing and measuring incompatibilities between pairs of services," in *Proc. 20th Int. Conf. DEXA*, 2009, pp. 229–243.
[14] Z. Zhou, S. Bhiri, H. Zhuge, and M. Hauswirth, "Assessing service protocol adaptability based on protocol reduction and graph search," *Concurrency and Computation: Practice and Experience*, vol. 23, no. 9, pp. 880–904, 2011.
[15] C. Canal, P. Poizat, and G. Salaun, "Model-based adaptation of behavioral mismatching components," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 546–563, 2008.
[16] L. Niels, M. Peter, S. Christian, and W. Daniela, "Analyzing interacting WS-BPEL processes using flexible model generation," *Data & Knowledge Engineering*, vol. 64, no. 1, pp. 38–54, 2008.
[17] C. Sqren and P. Laure, "Modular analysis of Petri nets," *The Computer Journal*, vol. 43, no. 3, pp. 224–242, 2000.
[18] J. Y. Jung, J. Park, S. K. Han, and K. Lee, "An ECA-based framework for decentralized coordination of ubiquitous web services," *Information and Software Technology*, vol. 49, no. 5, pp. 1141–1161, 2007.
[19] Y. Du, Y. Fan, and X. Li, "Verifying service composition based on modular reachability graph and generating BPEL codes," *Journal of Software*, vol. 21, no. 8, pp. 1810–1819, 2010.
[20] eBay developers Program, 2006. [Online]. Available: http://developer.ebay.com/DevZone/XML/docs/Web-Help/Checkout_Third_Party_Checkout.html
[21] BPEL2oWNF, 2007. [Online]. Available: http://www.gnu.org/software/bpel2owfn
[22] PIPE, 2005. [Online]. Available: http://www.pipe2. sourceforge.net/
[23] Y. Du, P. Xiong, Y. Fan, and X. Li, "Dynamic checking and solution to temporal violations in concurrent workflow processes," *IEEE Trans. Syst., Man,Cybern.: Part A*, vol. 41, no. 6, pp. 1166–1181, 2011.