

A Petri Net Siphon Based Solution to Protocol-level Service Composition Mismatches

Pengcheng Xiong¹, Mengchu Zhou² and Calton Pu¹

¹College of Computing, Georgia Institute of Technology

²Department of ECE, New Jersey Institute of Technology

xiong@gatech.edu, zhou@njit.edu, calton@cc.gatech.edu

Abstract

Protocol-level mismatch is one of the most important problems in service composition. The commonly used reachability exploration method focuses on verifying deadlock-freeness. When this property is violated, the states and traces in the reachability graph only give clues to re-design the composition. The process must then repeat itself until no deadlock is found. In this paper, multiple web service interaction is modeled with a Petri net called Composition net (C-net). The protocol-level mismatch problem is transformed into the deadlock structure problem of a C-net. If mismatches are found, a solution based on Petri net siphons is proposed. The proposed method is shown to achieve higher efficiency for resolving protocol-level mismatching issues than traditional ones do.

1. Introduction

In web service composition, when multiple web services are developed by different groups or vendors, they often fail to invoke each other because of mismatches. Service composition mismatches can be divided into interface and protocol-level ones.

The former include message signature mismatches and message split/merge mismatches [1]. Services can be composed if the provided interfaces with port types, operations, and message types match the required interfaces of the other web services. There is significant research result towards service interface level mismatches such as schema matching-based method [1], information retrieval techniques [2] and clustering-based approach [3].

Even if service interface level matches perfectly, there may be protocol-level mismatches causing problems such as unspecified reception and deadlock. Unspecified reception can be automatically solved by generating adaptors [4]. Deadlock mainly comes from message ordering mismatches and non-local choice mismatches [5-6]. Figure 1 gives an example for message ordering mismatches. The customer service first sends order message, waits for delivery and then sends payment message; online shop service waits for the order and payment then delivers the product. Even though the interfaces match syntactically, the interaction between them leads to a deadlock since the customer expects delivery first, while the shop expects payment first.

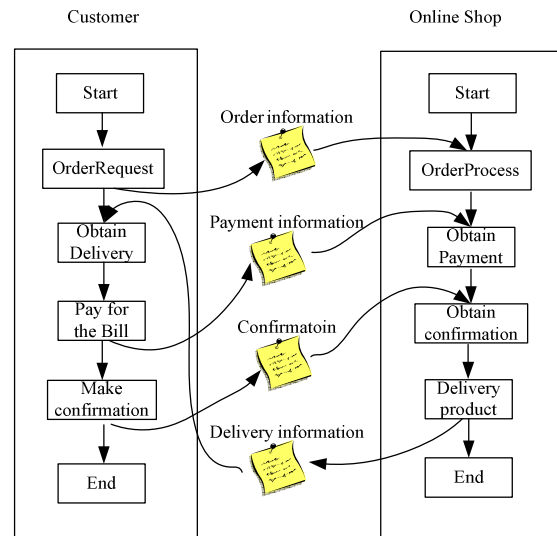


Figure 1. Illustration for message ordering mismatches

For the protocol-level mismatching, previously proposed methods [1, 9-11] mainly take the following steps: (1) check the protocol-level mismatching using a state-space based method; and (2) replace the mismatching services and repeat testing the state-space until there is no mismatching. In other words, they offer quite limited help in resolving protocol mismatches. Generally, finding the mismatching services involves intense interactions with developers. When protocols become complicated, it is very hard for developers to find the best solutions. For example, it is non-trivial [1] to find the best solution for the message ordering mismatches outlined in Figure 1.

The main contribution of the paper is a Petri net-based approach to find protocol-level mismatches and then generate optimized solutions to fix the mismatch problems. Our approach is based on an observation of a special class of Petri net objects called siphons (see Section 3 for details). A siphon is a subset of Petri net nodes with a property analogous to program safety properties. The number of tokens in a siphon will never increase and an empty siphon will always remain empty. Our observation is that protocol-level mismatch happens if and only if there is an empty siphon (see Theorems 1 and 2) in Petri net models created by BPEL composition.

Technically, our approach consists of three steps. First, we adopt Business Process Execution Language for Web Services (BPEL) as the web service composition language. In the first step, the BPEL description of a composite service is translated into a Petri net model. Second, we use a mix-integer programming formulation to detect the maximal empty siphons, which are then used to find protocol-level mismatches. Third, we describe an algorithm to find optimized siphon-based solutions for protocol-level mismatches by adding or holding tokens in siphons to prevent them from becoming empty.

The rest of the paper is organized as follows. Section 2 outlines related work. Section 3 introduces the BPEL and Petri net based modeling approach to define service composition. Section 4 describes the siphon-based algorithms to detect protocol-level mismatches and generate solutions. Section 5 concludes the paper.

2. Related work

The previously proposed methods for detecting and solving protocol-level mismatches mainly follow two steps, i.e., modeling and analysis.

There are a plethora of modeling methods. Foster *et al.* use message sequence charts by extracting the interaction message among services [7]. Fu *et al.* model the interactions of composite web services as conversations [8]. Other models and methods include abstract state machines, finite state machines, process algebra and pi-calculus.

As an appropriate method for modeling and analyzing distributed business processes, Petri nets are also an adequate modeling tool for web service behavior. As shown in [9], Petri nets are able to define and verify usability, compatibility and equivalence of web services. In particular, Petri net semantics for BPEL are proposed. Since BPEL is becoming an industrial standard for modeling Web service-based business processes, a Petri net-based method is directly applicable to real world examples.

We focus on Petri net-based methods and make comparisons among them. Ouyang *et al.* [10] transform BPEL into Petri nets represented in the Petri Nets Markup Language (PNML) by BPEL2PNML and propose WofBPEL to support three types of analyses, e.g., reachability analysis by generating the full state space. Lohmann adopts open workflow nets (oWFNs) [11] for modeling BPEL processes and uses Fiona to automatically analyze the interactional behavior of a given oWFN. Martens [6] proposes a BPEL annotated Petri nets (BPN) and presents a decision algorithm for the controllability of a BPN model based on the communication graph (c-graph). The check of interaction between the composed BPEL processes is transformed into the verification of deadlock-freeness of a BPN. After all parts that yield deadlocks are cut off, the remaining part is proven to be controllable. Nezhad *et al.* generate a mismatch tree to handle deadlock situations [1].

The basis of such tree is similar to the reachability graph in Petri nets. Although the current methods provide useful insights into the problem by adopting Petri net based modeling methods, e.g., PNML, oWFNs, c-graph and BPEL2PN, their analysis is mainly based on a reachable state space and they do not propose an effective solution to resolve the protocol-level mismatch issues.

Compared with their work, the proposed one tries to make the most effort to correct the existing composition. Using the analysis and correction methods based on siphons, it not only provides the candidate solutions for the developers but also offer the optimized solution, which greatly reduces a developer's work. Compared with our previous work which proposes a solution for non-local choice mismatch [6], this paper mainly focuses on providing a solution for message ordering mismatch.

3. Modeling methods

A Petri net is a directed bipartite graph. It consists of two components: a net structure and initial marking. A net (structure) contains two sorts of nodes: places and transitions. There are directed arcs from places to transitions and from transitions to places in a net. Places are graphically represented by circles while transitions by boxes or bars. A place can hold tokens denoted by black dots, or a positive integer representing their count.

Definition 1: A Petri net is a 3-tuple [12-13], $N=(P, T, F)$ where:

- i. $P=\{p_1, p_2, \dots, p_m\}$, $m>0$, is a finite set of places.
- ii. $T=\{t_1, t_2, \dots, t_n\}$, $n>0$, is a finite set of transitions.
- iii. $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.

This section shows how we model web service interaction with Petri nets called Composition net (C-net). Assume that service interface level mismatches do not exist, i.e., the message signature and number of interfaces in both parties match.

We divide the basic structures in BPEL, i.e., *receive*, *reply*, *invoke*, *assign*, *throw*, *terminate*, *wait*, *empty* and *link* into two categories. The first category is internal control logic that includes *assign*, *terminate*, *wait* and *empty*. The second category is external control logic that includes *receive*, *reply*, *invoke*, *throw* and *link*. Basic structures in the first category are not related to the interaction between different web services and we model them as internal status places and transitions. Basic structures in the second one are related to the interaction between different web services and we model them as transitions connected with internal status places and interface places as shown in Fig. 2. Note that, *invoke* is a combination of *reply* and *receive*, and *link* is modeled as an information channel.

There are *sequence*, *flow*, *pick*, *switch* and *while* structured activities in a BPEL process. Based on basic structures, the *sequence*, *pick* and *switch* structures can be covered. The semantics of *while* structure are similar to while-loop in programming languages like *Java*. Here we approximate the number of loops in a finite *while* structured activity and transform the activity to a *sequence* activity by expanding cycles [14]. We can transform the processes that are executed in parallel in the *flow* structure into the same processes that are invoked simultaneously in the *invoke* structure while maintaining the business logic. For example, we can divide the processes that are executed in parallel into separate BPEL processes while maintaining the business logic as shown in Fig. 2.

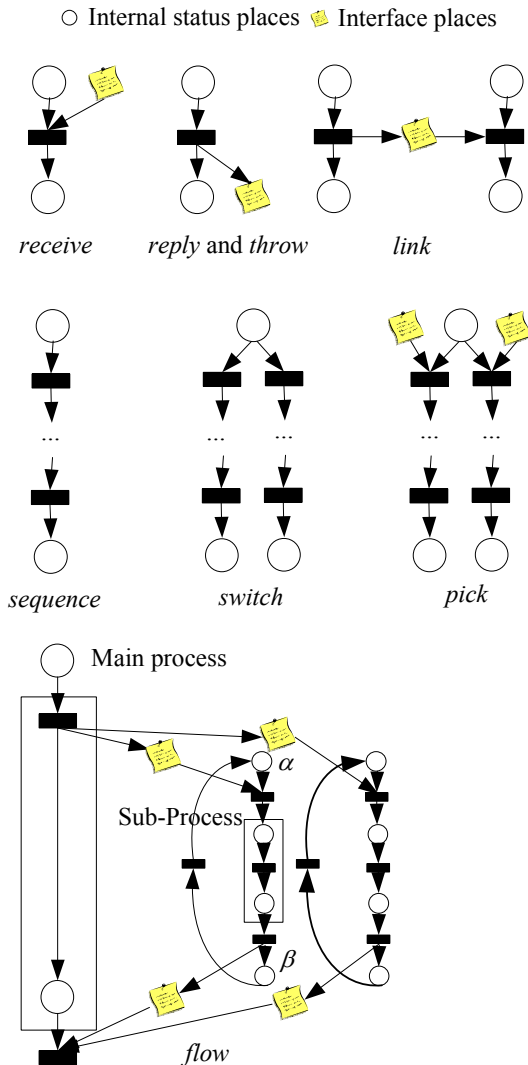


Figure 2. Transforming BPEL into Petri nets

For example, we model the action of sending order information of Customer service in Fig. 3(a). Here we

model the order information message as p_{11} and the customer service status before and after sending the message as p_1 and p_2 , respectively. We also model the action of receiving order information of Online Shop service in Fig. 3(b). We model the order information message as the same p_{11} and the online shop service status before and after receiving the message as p_6 and p_7 , respectively. The places like p_{11} are interface ones and the others like p_{1-2} and p_{6-7} are internal status ones. The whole C-net for Fig. 1 is illustrated in Fig. 6(a). We also model the start and end status as two special places and add a new transition t to connect them. For example, we model the start and end status for customer service as p_1 and p_5 , respectively. We use t_5 to connect them. An n -member C-net denoted as $N = \odot_{i=1}^n N_i$ is defined recursively by composing each C-net N_i .

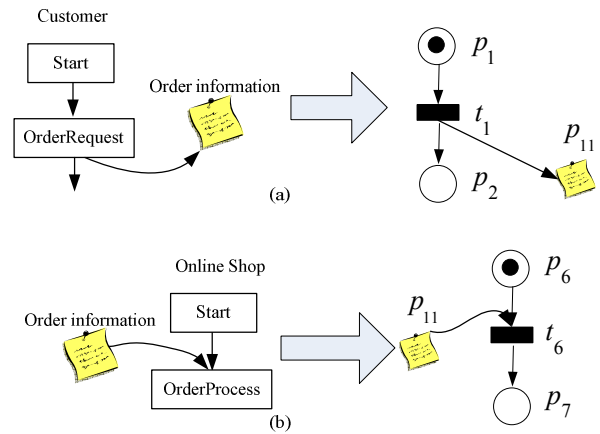


Figure 3. Modeling the case in Fig. 1 as C-net

Note that the interface places do not have tokens initially because no message is created. They can have tokens if and only if some transition wants to send a message through the information channel while they can lose a token if and only if some transition wants to receive a message through the information channel. A token in them models the situation when the required message is ready. We assume that the maximum number of tokens that an interface place can hold is one. Otherwise a BPEL process is not correct.

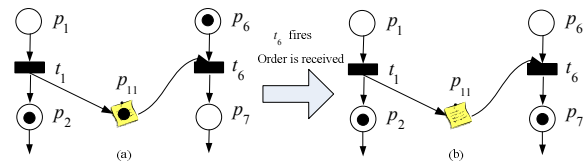


Figure 4. Modeling the interaction of web services

The firing of transitions in a C-net simulates the interaction of web services. For example, order information is

ready in Fig. 4(a). After t_6 fires, the order information is received by online shop service in Fig. 4(b).

The distribution of tokens over the places of a net is called a marking that corresponds to a state of the modeled system. The initial token distribution is hence called the initial marking. Let \mathbb{Z}^+ denote the set of non-negative integers and \mathbb{N} the set of positive integers. Then a marking M of a Petri net N is a mapping from P to \mathbb{Z}^+ . $M(p)$ denotes the number of tokens in place p . An initial marking is denoted by M_0 . The sum of tokens in all places in S is denoted by $M(S) = \sum_{p \in S} M(p)$.

A transition $t \in T$ is enabled under M , if and only if $\forall p \in \bullet t : M(p) > 0$ holds, denoted as $M \llbracket t \rrbracket$. For example, in Fig. 5(a), since $\bullet t_1 = \{p_1\}$ and $M(p_1) = 1 > 0$ holds, t_1 is enabled. If $M \llbracket t \rrbracket$ holds, t may fire, resulting in a new marking M' , denoted as $M \llbracket t \rrbracket M'$, with $M'(p) = M(p) - 1$ if $\forall p \in \bullet t \setminus \bullet t'$; $M'(p) = M(p) + 1$ if $\forall p \in \bullet t' \setminus \bullet t$; and otherwise $M'(p) = M(p)$. For example, after t_1 fires, we have a new marking M' where $M'(p_2) = 1$ and $M'(p_1) = 0$.

M' is reachable from M iff there exists a firing sequence $\sigma = t_1 t_2 \dots t_n$, such that $M \llbracket t_1 \rrbracket M_1 \llbracket t_2 \rrbracket \dots M_{n-1} \llbracket t_n \rrbracket M'$ holds. The set of markings reachable from M_0 in N is denoted as $R(N, M_0)$. For example, if we denote the marking in Figs. 5(a-d) as M_{0-3} , then $M_0 \llbracket t_1 \rrbracket M_1 \llbracket t_2 \rrbracket M_3$, $M_0 \llbracket t_2 \rrbracket M_2 \llbracket t_1 \rrbracket M_3$, and $R(N, M_0) = \{M_{1-3}\}$. The reachability set $R(N, M_0)$ of a net (N, M_0) can be expressed by a reachability graph. A reachability graph is a directed graph whose nodes are markings in $R(N, M_0)$ and arcs are labeled by the transitions of N .

Given a marked net (N, M_0) and $N = (P, T, F)$, a transition $t \in T$ is live under M_0 iff $\forall M \in R(N, M_0)$, $\exists M' \in R(N, M)$, $\exists M' \llbracket t \rrbracket$ holds. (N, M_0) is live iff $\forall t \in T : t$ is live under M_0 . For example, t_1 is not live under M_1 because $M_3 \in R(N, M_1)$ but $\forall M' \in R(N, M_3)$, $M' \llbracket t_1 \rrbracket$ cannot hold. A place subset $S \subseteq P$ is marked by M iff at least one place in S is marked.

Informally, a siphon of a Petri net is defined as a set S of places such that existence of any edge from a transition t to a place of S implies that there is an edge from some place of S to t . A post (pre) set of a place p is the set of output (input) transitions of p , denoted by p^\bullet and $\bullet p$ respectively. Given $Q \subseteq P$, $\bullet Q = \cup_{p \in Q} \bullet p$ and $Q^\bullet = \cup_{p \in Q} p^\bullet$. For example, we have $S = \{p_{1-3}\}$, $\bullet S = \{t_1\}$ and $S^\bullet = \{t_{1-2}\}$ as shown in Fig. 5. Since $\bullet S \subseteq S^\bullet$, S is a siphon. A siphon has a property: the number of tokens in it will never increase and an empty siphon will always remain empty and all its output transitions are dead. For example, there are two tokens in S initially in Fig. 5(a). But as the Petri nets

evolve in Figs. 5(b-d), the number of tokens in S never increases.

Definition 2: A nonempty place set $S \subseteq P$ is called a siphon iff $\bullet S \subseteq S^\bullet$.

A siphon is minimal iff there does not exist a siphon $S' \subset S$.

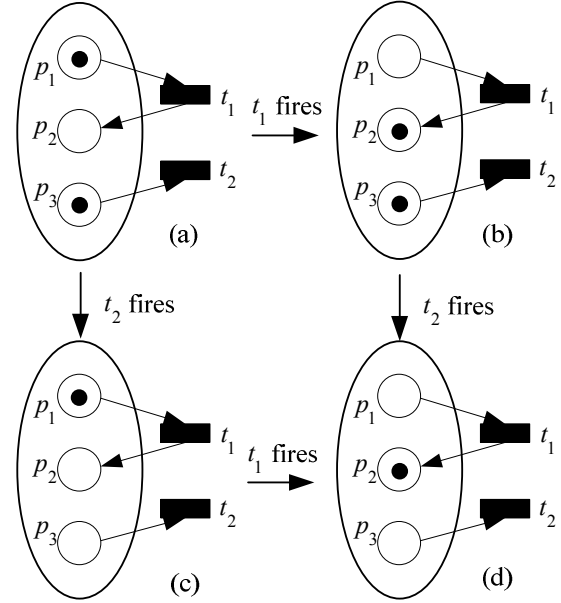


Figure 5. Illustration for Petri net siphon

4. Protocol-level mismatch detection and resolution

As stated in Section 1, deadlock in protocol-level mainly comes from message ordering mismatches and non-local choice mismatches. In this section, we will propose a detection as well as solution method for protocol-level mismatch.

4.1 Detecting protocol-level mismatch

As stated in the previous section, the start and end status is modeled as two special places and there is also a transition t connecting them. Thus if the token can arrive at the end place, then the initial marking is reachable, i.e., there should be no protocol-level mismatch. For example, in Fig. 6(a), if the token in p_1 finally arrives at p_5 , the initial marking is reachable by firing t_5 .

Definition 4: A C-net $N = \odot_{i=1}^n N_i$ matches at protocol-level iff the initial marking is reachable for each reachable marking.

We can also prove that if there is no dead transition when the message exchanges between protocols, the inter-

action among web services can proceed until the exchange ends up in final states. There is a dead transition in C-net if and only if there is an empty siphon.

Theorem 1: Suppose a C-net N with an initial marking M_0 . Let $M \in R(N, M_0)$ and let $t \in T$ be a dead transition at M . Then \exists a siphon S , $\exists M(S)=0$.

Theorem 2: A C-net $N = \odot_{i=1}^n N_i$ matches at protocol-level iff $\forall M \in R(N, M_0)$, \forall (minimal) Siphon S , $M(S) \neq 0$.

The detailed proofs for Theorems 1 and 2 are in [6]. They are omitted here due to space constraints. For instance, there are 4 minimum siphons in Fig. 6(a), i.e., $S_1=\{p_{1-5}\}$, $S_2=\{p_{6-10}\}$, $S_3=\{p_1, p_{3-5}, p_{7-9}, p_{11}, p_{14}\}$, $S_4=\{p_3, p_{8-9}, p_{12}, p_{14}\}$, $S_5=\{p_{3-4}, p_9, p_{13-14}\}$. $M_0(S_{1-3})=1$ and $M_0(S_{4-5})=0$. Since there are initial empty siphons, i.e., S_4 and S_5 , there exists protocol-level mismatching. This is true because after t_1 and t_6 fire, there is a deadlock as shown in Fig. 6(b).

Thus the problem of protocol-level mismatching of web service interaction is transformed to the problem of empty minimal siphons in a C-net. We can use the mix-integer programming algorithm to detect the maximal empty siphon [15].

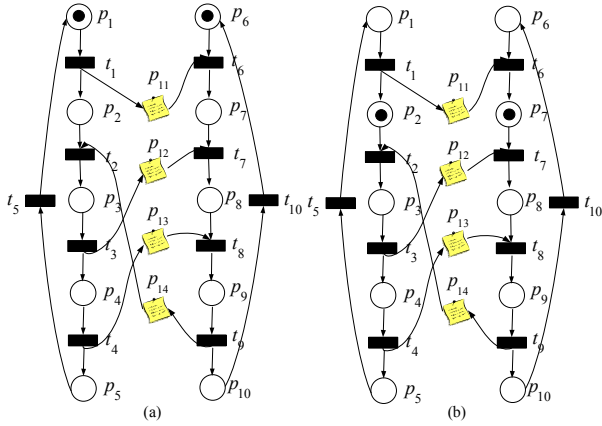


Figure 6. Modeling the interaction of web services

In the aspect of detecting protocol-level mismatch, neither of the reachability analysis based method and mix-integer programming has clear computational advantage over the other because both of them have exponential complexity [16]. However, the former is sensitive to the change of initial markings while the latter is not. This property can be used to reduce the computational complexity for incompatibility detection for different initial states. Note that a new reachability graph is required for each new initial state.

Suppose that we have a C-net $N = \odot_{i=1}^n N_i$ where some minimal siphons can become empty. Our main goal is to introduce into the system a solution to guarantee that no empty minimal siphons are reachable during the evolution of the new C-net, i.e., the new C-net at protocol-level matches.

4.2 Correcting message ordering mismatch

The scenario in Fig. 1 is classified as message ordering mismatch. In this case, there are initial empty siphons. The customer is waiting for the delivery information while the online shop is waiting for the payment information. Thus, developers must be involved to provide additional information at the deadlock point, i.e., to ask the customer to provide the payment information or to ask the online shop to provide the delivery information.

But the developers will probably not make the best decision considering the complicated future message exchanges of two parties after their decision. For example, if they ask the customer to provide the payment information, then the next deadlock happens, i.e., the online shop is waiting for confirmation and the customer is waiting for delivery. But if they ask the online shop to provide the delivery information, then no deadlock happens.

Here we provide an algorithm based on the C-net to help developers choose the best solution. According to Theorems 1-2, the deadlock happens because there are empty siphons in the C-net. According to the property of a siphon, the number of tokens in a siphon will never increase and an empty siphon will always remain empty. Thus, possible solutions can be adding one or more tokens to interface places p_{11-14} to make all of the empty siphons marked. For example, adding one token to p_{11} means that the order information is ready while adding to p_{12} means that the payment information is ready.

Among all the solutions that the developers can have, we should provide the best one. It should contain the smallest amount of information. For example, payment, confirmation and delivery information is three different kinds of information. The best solution is to provide only the delivery information, not all of the three. The best solution should guarantee the message ordering match in the future as well. For example, the best decision is to provide the delivery information that guarantees that no deadlock will ever happen.

We find the best solution by linear programming.

Algorithm:

INPUT: (1) n -member C-net N with minimum siphon set $\Omega = \Omega_N \cup \Omega_Y$. Ω_N denotes the initial non-empty siphon set, and $\Omega_Y = \{S_1, S_2, \dots, S_i\}$ denotes the initial empty siphon set. (2) interface place set $P_E = \{p_1, p_2, \dots, p_j\}$

OUTPUT: A list of messages that should be provided. We denote the list as a $j \times 1$ vector L where $L(j)=1$ if $p_j \in L$;

and 0 otherwise.

BEGIN:

Step 1. / Calculate the contribution matrix of every message to the siphon*/*

Constitute an $i \times j$ matrix A , where $A(i, j)=1$ if $p_j \in S_i$; and 0 otherwise.

Step 2. / Optimization*/*

Compute the following linear programming problem:

Minimize $\mathbf{1}^*L$

s.t. $A*L=\mathbf{1}^T$

Step 3. / Return result*/*

Return L

END

We explain the idea underlying this algorithm as follows:

Firstly, the constraint function $A*L=\mathbf{1}^T$ can return all the solutions. As shown in Fig. 7, because we have $A(i, j)=1$ if $p_j \in S_i$; and 0 otherwise, and $L(j)=1$ if $p_j \in L$; and 0 otherwise, the solutions of the constraint function guarantee that each empty siphon is marked by exactly one token. Moreover, if the constraint function is not satisfied, there is at least one empty siphon.

Secondly, the objective function of the linear programming formulation can return the best solution. In the contribution matrix, the more siphons the message p_j is involved in, the more $\mathbf{1}$'s it has in the j th column. Since the objective function calculates the sum of messages, the solution has the smallest total number of messages if the proposed objective function is minimized.

Finally, we have the following theorem:

Theorem 3: Application of the above algorithm to every C-net can eliminate the message ordering mismatches.

For the scenario in Fig.1, we have $\Omega_N=\{S_{1-3}\}$, $\Omega_Y=\{S_{4-5}\}$, and $P_E=\{p_{11-14}\}$. Because $p_{12} \in S_4$, $p_{14} \in S_4$, $p_{13} \in S_5$, $p_{14} \in S_5$, we have $A=((0,1,0,1); (0,0,1,1))$. The result is $L=(0,0,0,1)^T$. It means that the developer should ask the online shop to provide the delivery information (the token in the interface place p_{14} denotes the delivery information). Moreover, if we check the method that the developer asks the customer to provide the payment information, i.e., $L'=(0,1,0,0)^T$, we find that this method will fail. This is simply because L' is not a solution of the linear programming problem. Because $A*L'=(1,0)^T$, although S_4 is marked, S_5 is still empty.

$$\begin{array}{c}
 p_1 \ p_2 \ p_3 \ \dots \ p_j \\
 S_1 \\
 S_2 \\
 S_3 \\
 \vdots \\
 S_i
 \end{array}
 \begin{pmatrix}
 \mathbf{a}_{11} & \dots & \mathbf{a}_{1j} \\
 \vdots & \ddots & \vdots \\
 \mathbf{a}_{i1} & \dots & \mathbf{a}_{ij}
 \end{pmatrix}
 \begin{pmatrix}
 L_1 \\
 L_2 \\
 L_3 \\
 \vdots \\
 L_j
 \end{pmatrix}
 =
 \begin{pmatrix}
 1 \\
 1 \\
 1 \\
 \vdots \\
 1
 \end{pmatrix}$$

Figure 7. Illustration of linear programming

4.3 Local choice mismatch solution

Different from the message ordering mismatch where there are initial empty siphons, there are no initial empty siphons in local choice mismatch. But some of the siphons may lose their tokens if some transitions fire. We propose a method based on additional information channels in [6] to hold the tokens in siphons such that every siphon is always marked.

4.4 Implementation

The implementation contains three steps. Firstly, developers model the BPEL with Petri nets using the modeling methods in Sec. 3. Secondly, they run the algorithm. Finally, they take action according to the algorithm's results.

For example, in the scenario described in Fig. 1, developers should construct the messages that are denoted by the non-zero elements in L in the algorithm's results. For example, the solution $L=(0,0,0,1)^T$ means that they should construct the delivery information. There are several pieces of evidence that can be used to help them construct such information [1]: (1) Interface-based inference; (2) Log based value/type inference; and (3) Developer input.

We show how to construct the delivery information. The schema of delivery information message normally includes "Order number", "Order date", "Product number/description", "Quantity", "Unit price", "Merchandise net", "Tax amount", "Shipping amount", "Total amount" and "Ship to address". Firstly, the elements "Order date", "Product number/description", "Quantity" and "Ship to address" can be obtained from order information message that was sent by customer by interface-based inference. Secondly, the elements "Unit price" can be obtained from previous logs from online shop by log-based value inference. Then elements "Tax amount", "Shipping amount" and "Total amount" can be calculated. Finally, developers input "Order number".

A prototype web service application was built using web service composition on ActiveBPEL [17] to validate the siphon-based technique described above.

5. Conclusion

Service composition is widely used as a way to realize multiple functional requirements. However, mismatches at the interface and protocol levels may render the composite service unusable. Although the existing studies, e.g., PNML, oWFNs, c-graph and BPEL2PN, can analyze the problems based on reachability analysis, they do not provide a direct solution.

The main contribution of this paper is to propose a siphon-based analysis technique that yields a variant of component service without mismatches. Without checking the state space, our approach provides an optimized and also automatic solution for correcting protocol-mismatches. This approach greatly reduces the amount of interactions with developers.

There are some limitations that can lead to interesting future work. First, our algorithm cannot lead to a solution if the method of adding information is not applicable. Second, although the search for siphons can be performed offline and the computation of minimum siphons is simple, in some complex structured C-net, such computation can be expensive. Some polynomial complex algorithms to find and control siphons should be explored for C-nets by making full use of their special structural information. Some recent advance by Wang *et al.* [18] may provide good help along this direction.

6. References

- [1] H. Nezhad, B. Benatallah, A. Martens, F. Curbera and F. Casati, "Semi-Automated Adaptation of Service Interactions", *Proc. of the 16th Intern. World Wide Web Conf.*, Banff, Alberta, Canada, May 2007, pp. 993-1002.
- [2] Y. Wang and E. Stroulia, "Flexible interface matching for web-service discovery," *Proc. of the 4th Intern. Conf. on Web Info. Sys. Eng.*, Rome, Italy, Dec. 2003, pp. 147-156.
- [3] X. Dong, A. Halevy, J. Madhavan, E. Nemes and J. Zhang, "Similarity Search for Web Services", *Proc. of the 30th Intern. Conf. on Very Large Data Bases*, Toronto, Canada, Sept. 2004, pp. 372-383.
- [4] D.M. Yellin and R.E. Strom, "Protocol specifications and component adaptors", *ACM Trans. on Prog. Lang. and Sys.*, Vol. 19, Iss. 2, Mar. 1997, pp. 292-333.
- [5] A. Martens, "Usability of Web services", *Proc. of the 4th Intern. Conf. on Web Info. Systems Eng. Workshops*, Rome, Italy, Dec. 2003, pp. 182-190.
- [6] P.C. Xiong, Y.S. Fan and M.C. Zhou, "A Petri Net Approach to Analysis and Composition of Web Services", to appear in *IEEE Trans. on Sys., Man and Cybern., Part A*, 2009.
- [7] H. Foster, S. Uchitel, J. Magee and J. Kramer, "Tool support for model-based engineering of Web service compositions", *Proc. of the 2005 IEEE Intern. Conf. on Web Services*, Orlando, FL, USA, July 2005, Vol. 1, pp. 95-102.
- [8] X. Fu, T. Bultan and J. Su, "Analysis of Interacting BPEL Web Services", *Proc. of the 13th Intern. World Wide Web Conf.*, New York, NY, May 2004, pp. 621-630.
- [9] A. Martens, R. Hamadi and B. Benatallah, "A Petri Net based Model for Web Service Composition", *Proc. of the 14th Australian Database Conf.*, Adelaide, Australia, Feb. 2003, pp. 191-200.
- [10] C. Ouyang, E. Verbeek, W.M.P. van der Aalst, S. Breutel, M. Dumas and A.H.M. ter Hofstede, "WofBPEL: A Tool for Automated Analysis of BPEL Processes", *Proc. of the 3rd Intern. Conf. on Service Oriented Computing*, Amsterdam, Netherlands, Dec. 2005, pp. 484-489.
- [11] N. Lohmann, "A feature-complete Petri net semantics for WS-BPEL 2.0", *Proc. of the 4th Intern. Workshop on Web Services and Formal Methods*, Brisbane, Australia, September 2007, pp. 77-91.
- [12] M.C. Zhou and K. Venkatesh, *Modeling, Simulation and Control of Flexible Manufacturing Systems: A Petri Net Approach*, World Scientific, Singapore, 1998.
- [13] Pengcheng Xiong, Yushun Fan and Mengchu Zhou, "QoS-aware Web Service Configuration," *IEEE Trans. on Sys., Man and Cybern., Part A.*, Vol. 38, Iss. 4, July 2008, pp. 888-895.
- [14] J. Ezpeleta, J.M. Colom and J. Martinez, "A Petri Net based Deadlock Prevention Policy for Flexible Manufacturing Systems," *IEEE Trans. on Robotics and Automation*, Vol. 11, No. 2, Apr. 1995, pp. 173-184.
- [15] F. Chu and X. L. Xie, "Deadlock analysis of Petri nets using siphons and mathematical programming," *IEEE Trans. on Robotics and Auto.*, vol. 13, no. 6, pp. 793-804, Dec. 1997.
- [16] Z.W. Li and M.C. Zhou, "Elementary Siphons of Petri Nets and Their Applications to Deadlock Prevention in Flexible Manufacturing Systems," *IEEE Trans. on Sys., Man and Cybern., Part A*, Vol. 34, No. 1, Jan. 2004, pp. 38-51.
- [17] <http://www.activevos.com/>
- [18] A.R. Wang, Z.W. Li, J.Y. Jia and M.C. Zhou, "An Effective Algorithm to Find Elementary Siphons in a Class of Petri Nets," to appear in *IEEE Trans. on Sys., Man and Cybern., Part A*, 2009.