Service Protocol Adaptability Assessment for Supporting Mediated Service Interactions

ZhangBing Zhou¹, Lei Shu², Jianwei Niu³
¹School of Information Engineering, China University of Geosciences, China
²Department of Multimedia Engineering, Osaka University, Japan
³School of Computer Science and Engineering, Beihang University, China
zbzhou@cugb.edu.cn, lei.shu@ist.osaka-u.ac.jp, niujianwei@buaa.edu.cn

Abstract

Discovery and selection of a suitable provider service, which can interact with a certain requester service for achieving a certain goal, is a fundamental promise of service oriented computing paradigm. Current techniques analyzing service interactions focus on either compatibility analysis or adapter synthesization. By using our walk computation technique, in this paper we propose a novel adaptability assessment technique, which (i) computes an adaptation degree that specifies to what extent two service protocols are adaptable, and (ii) provides a set of conditions that determine when these two service protocols can interact properly through a certain adapter. The condition set and the adaptation degree are complementary criteria to be applied by the requestor for identifying the most suitable provider service. Indeed, the techniques of adaptability assessment and adapter synthesization complement each other for facilitating mediated service interactions. The adaptability assessment, and the walk computation technique we developed, have been implemented and plugged into ProM as a component.

Keywords: Adaptability assessment, Mediated service interaction, Walk computation, Service protocol.

1 Introduction

Service computing, as a distributed computing paradigm, enables the collaboration between loosely coupled, standard based, and protocol independent service components [38]. Web services, as the main pillar of service oriented technology, have been widely accepted by academia and industry and are supported by many vendors to provide standard means of interoperation in business-to-business applications [14][34][44]. Independently developed software components are encapsulated according to Web services standards, and run on a variety of platforms and frameworks. Interactions are conducted between Web services for fulfilling certain requirements prescribed by requestors.

Effectively discovering and then selecting a suitable provider service, which can interact with a requester service to achieve a certain pre-defined goal, is a fundamental promise of service-oriented computing paradigm [38] and cloud computing [3], and is important to achieve the vision of Internet of Things [43]. Given the inherent autonomy, heterogeneity, and continuous evolution of Web services, mismatches usually exist between Web services in their signatures, behavioural interfaces, and semantic and non-functional properties. An adapter [17][36][48][50] is usually necessary to be synthesized for reconciling these mismatches, in order to facilitate an interaction. We call such an interaction a mediated service interaction, which is observed a common style of service interactions of practical relevance [42]. In this context, efficient techniques that can: (1) identify, classify, and reconcile mismatches between Web services, the so-called adapter synthesization [16]; (2) determine which provider service among multiple candidates is appropriate with respect to certain requirements specified by a requestor, called adaptability assessment in this paper, are essential to achieve a mediated service interaction. By adaptation [16], we mean the act of identifying, classifying, and reconciling mismatches between Web services. Given two Web services, adaptability assessment means the act of determining: (i) to what extent these two services are adaptable, and (ii) under which conditions a mediated service interaction can be properly conducted.

It is worth mentioning that adapter synthesization and adaptability assessment complement each other for supporting mediated service interactions. Specifically, when a requestor raises the interaction requirements, she would apply the technique of adaptability assessment for examining whether or not, as well as under which conditions, interactions fulfilling her requirements can be conducted. When the result of this analysis is positive, she would synthesize an adapter for reconciling mismatches occurred in service interactions. Generally, what adaptability assessment interests are the types of resolvable mismatches, while technical details of resolving these mismatches are the focal point of adapter synthesization techniques.

*Corresponding author: Lei Shu; E-mail: lei.shu@ist.osaka-u.ac.jp
As presented in [16], current techniques analyzing service interactions focus on either compatibility analysis [2][5][33] or adapter synthesisization [17][27][36][42][48]. Adaptability can somewhat be assessed using techniques that synthesize adapters. However, these techniques cannot provide the level of assessment we target. Specifically, being able to synthesize an adapter implies that service protocols can interact in some situations, but which interaction can be conducted and under which conditions are not specified. A service protocol is the specification of a Web service from a behavioral interface perspective [16]. In a nutshell, an adapter cannot differentiate the different adaptation possibilities between service protocols. Hence, these service protocols are assumed to be the same to the requestor although they are actually different in their adaptation possibilities. Moreover, an adapter does not prescribe conditions that determine when two service protocols can be adapted. Consequently, an interaction may fail either because it is an unadaptable interaction, or because some conditions cannot be satisfied with respect to exchanged message instances.

In this paper we propose a novel technique which aims to assess the adaptability of service protocols. We conduct an assessment from a behavioural interface perspective. Given two service protocols, we assess the adaptability by applying our walk computation technique. A walk [6] of a service protocol is part of this service protocol, such that, this part can involve in a certain interaction. This paper has the following major contributions:

1. Adaptability assessment provides an overall evaluation about all possible interactions between two service protocols. Therefore, a requester can distinguish between candidate provider services according to the supportability of the interactions prescribed according to her requirements, and then select one which is the most suitable. This is important especially to critical, longrunning, and non-repeatable interactions.

2. Adaptability assessment raises the state of service (or software component) adaptation, such that, an adapter is required to be synthesized when expected interactions are examined achievable.

3. The walk computation technique we proposed is the first to compute all walks in a directed cyclic graph. This technique is general and can be applied to any topic where walks in a directed cyclic graph are required to be computed.

The reminder of this paper is organized as follows. In Section 2, we introduce background concepts including service protocol and mediated service interaction, and present mismatch patterns which are considered reconcilable in the following sections. In Section 3, we give an overview of our adaptability assessment approach, which is to be detailed in the following sections. In Section 4, we present our algorithm that computes the complete adaptation graph for two service protocols. This complete adaptation graph captures all mediated interactions that can be conducted according to the adaptation mechanisms of a certain adapter. In Section 5, we introduce our walk computation technique, which is applied to compute all instance sub-protocols of a service protocol, and all pairs of instance sub-protocols captured by the complete adaptation graph. A mediated service interaction occurs between a pair of instance sub-protocols. In Section 6, we assess the adaptability by computing an adaptation degree and providing a set of conditions. In Section 7, we present our survey about the size and the complexity of BPEL service protocols, and show our prototype implementation. We discuss the related techniques in Section 8, and finally conclude this paper in Section 9.

2 Preliminaries

In this section we introduce the concepts of service protocol and mediated service interaction.

2.1 Service Protocol

We adopt guarded finite state automata [20] to model service protocols. Transitions are triggered by exchanging messages between service protocols. Conditions may be defined on transitions. A transition can be enabled when the associated conditions hold. We use the terms of condition and guard interchangeably in this paper.

Definition 2.1 (Service Protocol). A service protocol is a tuple \( p = (M, S, s, F, C, T) \) where:

1. \( M \) is a finite set of message names. Given \( m \in M \), we use \( m(+) \) and \( m(-) \) to denote the incoming or outgoing of \( m \) [5], \( m \) contains a set of attributes.
2. \( S \) is a finite set of states. \( s \) is the initial state. \( F \) is a finite set of final states.
3. \( C \) is a finite set of conditions defined upon message attributes.
4. \( T \subseteq S \times M \times C \times T \) is a finite set of transitions. Each transition \( \tau = (s, s, m(+ / -), c | \text{true}) \in T \) defines (i) a source state \( s \), (ii) a target state \( s \), (iii) an incoming or outgoing message \( m \), and (iv) a condition \( c \in C \). True is the condition by default.

Figure 1 shows service protocols of two soft-drink providers (i.e., \( SP \) and \( SP_A \)) and a requestor \( SR \). \( SP \) and \( SP_A \) differ in that \( SP \) allows (i) canceling an interaction when a requestor does not satisfy the price, and (ii) adjusting \( SD \) Items when a requestor does not satisfy the selected \( SD \) Items. Some messages in \( SP \) (or \( SP_A \)) and \( SR \) are exchanged in different orders, like the exchange order for messages custInfo and sdItems. In addition, an ACK
received by SR is not provided by SP (or SP_A). \( Cd_i (i \in [1, 7]) \) are conditions defined upon transitions.

2.2 Mediated Service Interaction

Given two service protocols, two different styles of service interactions may be conducted depending on whether there are mismatches existing between them or not:

1. **Direct service interaction**, as shown in Figure 2(a). No mismatch exists between service protocols. Concretely, during an interaction, when one service protocol sends a message, another service protocol is always ready to receive this message. Note that in Figure 2, \( P1 \) and \( P2 \) represent service protocols. \( A, B, \) and \( C \) are the messages sent or received by \( P1 \) and \( P2 \).

2. **Mediated service interaction**, as shown in Figure 2(b). Mismatches exist between service protocols. Therefore, an adapter, acting as a centric arbiter to reconcile these mismatches, makes these service protocols compatible. Given an adapter, the kinds of reconcilable mismatch depend on its adaptation mechanisms. As reviewed by [16], adapter synthesization is an active research topic and several adapters have been proposed.

According to [4][27], mismatches between service protocols can be classified into the following categories:

1. **Attribute granularity difference in messages**. This mismatch happens when the attribute sets of corresponding messages in different service protocols are not equivalent. Most adapters can reconcile this mismatch by splitting and merging messages.

2. **Sending and then receiving message mismatch**. This mismatch means that the messages to be received by a service protocol have been sent by partner service protocols previously, instead of at the current stage. Concretely, assuming two messages \( m_1 \) and \( m_2 \) are specified in both \( p_1 \) and \( p_2 \). This mismatch happens, for instance, when \( p_1 \) sends \( m_1 \) and then receives \( m_2 \), but \( p_2 \) sends \( m_2 \) and then receives \( m_1 \). Most adapters can reconcile this mismatch by (i) temporarily saving \( m_1 \) for \( p_2 \) and \( m_2 \) for \( p_1 \) firstly, and (ii) forwarding \( m_2 \) to \( p_1 \) and \( m_1 \) to \( p_2 \) afterwards.

---

**Figure 1** Service Protocols of Two Soft-Drink Provider and A Requestor Services. A Requestor, Using SR, Chooses a Provider Service from SP and SP_A to Interact with, for Buying Some Soft-Drinks Online

**Figure 2** Direct and Mediated Service Interaction
(3) **All receiving message mismatch.** This mismatch means that all service protocols expect to receive messages, but these messages have not been sent by the partner service protocols. This mismatch happens, for instance, when both \( p_1 \) and \( p_2 \) expect to receive messages \( m_1 \) and \( m_2 \), but \( m_1 \) and \( m_2 \) have not been sent by \( p_1 \) and \( p_2 \).

- If \( m_1 \) or \( m_2 \) is an acknowledgement message \( \text{ACK} \), this mismatch can be reconciled by most adapters via generating such \( \text{ACK} \) and forwards it to \( p_1 \) or \( p_2 \). An \( \text{ACK} \) message has no attributes.
- If both \( m_1 \) and \( m_2 \) are messages with attributes, this mismatch may be reconciled by some adapters [36][50].

In [36], a missing message may be provided (manually) through mechanisms called *evidences*. In [50], a mock-up message may be provided if the dependency between an activity currently enabled and one of its directly-succeeding activities is neither control nor mandatory data dependent. Note that this mismatch is not always reconcilable.

We refer to [4][27] for the details of adaptation model. In this paper we use \( \text{APT} \) to represent any adapter that can reconcile the following mismatches: (i) attribute granularity difference in messages, (ii) sending and then receiving message mismatch, and (iii) all receiving message mismatch where a message expected by \( p_1 \) or \( p_2 \) is an \( \text{ACK} \) message. We consider these kinds of mismatches since:

1. These kinds of mismatches are reconcilable according to the adaptation mechanisms of most adapters [36][42][48].
2. Techniques reconciling these kinds of mismatches are relatively easy to be achieved through synthesizing an adapter [27][35-36][42][48][50].
3. These kinds of mismatches represent typical mismatches between service protocols [4][17][32].

Mismatches between \( \text{SR} \) and \( \text{SP} \) (or \( \text{SP}_A \)) can be reconciled according to the adaptation mechanisms of \( \text{APT} \). Regarding \( \text{SR} \) and \( \text{SP} \), interactions can lead them to (i) \( \text{SD Items Shipped} \). Concretely, the interaction leads \( \text{SP} \) (or \( \text{SP}_A \)) to \( \text{SD Items Shipped} \), and leads \( \text{SR} \) to \( \text{SD Items Shipped} \), or (ii) both \( \text{SP} \) and \( \text{SR} \) to their cancellation.

Regarding \( \text{SP}_A \) and \( \text{SR} \), by using \( \text{APT} \), interactions can lead them to \( \text{SD Items Shipped} \). In other words, the interaction leads \( \text{SP}_A \) to \( \text{SD Items Shipped} \), and leads \( \text{SR} \) to \( \text{SD Items Shipped} \).

### 3 Approach Overview

Before conducting an interaction, a requestor usually needs to examine whether her *expected* interactions can be achievable or not with respect to her requirements [51]. For instance for \( \text{SP} \) and \( \text{SR} \), a requestor needs to examine whether an interaction can lead them to \( \text{SD Items Shipped} \). The fact that an adapter can be synthesized implies that service protocols can interact in some situations. However, whether a certain interaction can be supported or not cannot be determined. In this section we propose to assess the adaptability of two service protocols depending on the adaptation mechanisms of \( \text{APT} \). The result includes the following two aspects:

1. An **adaptation degree**, which specifies to what extent these two service protocols are adaptable.
2. A **condition set**, which specifies the conditions under which an (expected) interaction can be properly performed.

In the following we introduce our technique for assessing the adaptability. States and transitions of a service protocol involved in different interactions are usually different. We use the notion of *instance sub-protocol* (see Definition 5.1) to represent the part of a service protocol involved in a certain interaction. For two service protocols, an interaction actually occurs between a pair of their instance sub-protocols.

Mediated service interactions occur between two service protocols depending on the adaptation mechanisms of a certain adapter. We use the notion of *complete adaptation graph* to capture all mediated interactions occurred between two service protocols. A mediated interaction is an interaction between a pair of instance sub-protocols of these two service protocols. Therefore, the complete adaptation graph captures all mediated service interactions occurred between pairs of their instance sub-protocols.

Based on (i) all instance sub-protocols of service protocols, and (ii) all mediated service interactions captured by the complete adaptation graph, we assess the adaptability by (i) providing a set of conditions that determine when mediated service interactions can be conducted, and (ii) computing an adaptation degree that specifies to what extent two service protocols are adaptable.

An overview of our approach is illustrated by Figure 3. \( \text{GFS4} \) means guarded finite state automata. Generally, this approach includes the following steps:

- **Step 1:** Given two service protocols and depending on the adaptation mechanisms of a certain adapter, we construct the complete adaptation graph. This step is presented in Section 4.
- **Step 2:** By applying the walk computation technique we developed, we compute (i) all instance sub-protocols in service protocols, and (ii) all mediated service interactions captured by the complete adaptation graph. These interactions correspond to the walks of this graph. The walk computation technique is presented in Section 5. We show in Section 5.1 about how to compute instance sub-protocols of a service protocol.
4 Adaptation Graph Construction

An adaptation graph captures message exchanges between two service protocols through APT. A sequence of message exchanges starting at the initial states of these two service protocols and ending at a pair of their final states is a complete sequence, which reflects a mediated interaction between these two service protocols. The part of a service protocol involved in a mediated service interaction is one of its instance sub-protocols. We represent an adaptation graph in terms of guarded finite state automata. A state in an adaptation graph is a pair of states of the two service protocols involved in a certain interaction, and a transition corresponds to either:

1. A synchronous message exchange between \( p_1 \) and \( p_2 \) through APT: This kind of transition occurs, for instance, when \( p_1 \) sends a message \( m \), \( p_2 \) is ready to receive \( m \) immediately. Then, \( p_1 \) sends \( m \) to APT, and APT forwards \( m \) to \( p_2 \) immediately. For simplicity, we use one transition which sends \( m \) from \( p_1 \) to APT, and another from APT to \( p_2 \), to represent this message exchange. We use the notation: \( p_1 \text{-to-} p_2: m \), to represent this message exchange.

2. \( p_1 \) sending a message \( m \) to APT: This kind of transition occurs, for instance, when \( p_1 \) sends \( m \), \( p_2 \) is not ready to receive \( m \) immediately. Hence, \( p_1 \) sends \( m \) to APT. We use the notation: \( p_1 \text{-to-} APT: m \), to represent this message exchange.

3. APT forwarding a message \( m \) to \( p_2 \), where \( m \) is either a message generated by \( p_1 \), or an ACK message generated by APT: This kind of transition occurs, for instance, when \( p_2 \) wants to receive \( m \). \( m \) is (i) sent by \( p_1 \), or (ii) not sent by \( p_1 \) but \( m \) is an ACK message which can be provided by APT. We use the notation: \( APT\text{-to-}p_2: m \) or \( APT\text{-to-}p_2: \text{ACK} \), to represent this message exchange.

For the latter two kinds of transition, the state of the service protocol, which sends or receives a message to or from APT, evolves to one following state. The state of the partner service protocol is not evolved.

Conditions associated with transitions in an adaptation graph are gotten from those in service protocols. Conditions are not evaluated when an adaptation graph is built, since their evaluation depends on exchanged message instances.
An adaptation graph is complete if it captures all mediated service interactions that can be conducted between two service protocols according to the adaptation mechanisms of a certain adapter. For instance, Figure 4 shows the complete adaptation graph for SP and SR where APT is used to reconcile the mismatches. We denote this graph as adapt\textsubscript{graph} in this paper. In the following we present our algorithm that constructs the complete adaptation graph for two service protocols.

4.1 Generating Complete Adaptation Graph

In this section, we introduce the algorithm genCompAdapt-Graph() that generates the complete adaptation graph for two service protocols $p_1 = (M_1, S_1, s_1, F_1, C_1, T_1)$ and $p_2 = (M_2, S_2, s_2, F_2, C_2, T_2)$. \textit{adapt}\textsubscript{graph} = $(M, S, s, F, C, T)$ is the complete adaptation graph generated according to the adaptation mechanisms of APT. We first introduce some variables and functions as follows:

Variables:
- $\text{Cand}$: a finite set of tuples in terms of $(s_{p_1}, M_{p_1}, s_{p_2}, M_{p_2})$.
- $s_{p_1}$ (and $s_{p_2}$) refers to the state of service protocol $p_1$ (and $p_2$) at a certain point during an interaction.
- $M_{p_1}$ (and $M_{p_2}$) is a set of messages that $p_1$ (and $p_2$) has sent and received when $p_1$ (and $p_2$) evolves to the state $s_{p_1}$ (and $s_{p_2}$).
- $M_{p_1}$ (or $M_{p_2}$): a finite set of messages sent and received by $p_1$ (or $p_2$).
- $T_{p_1}$ (or $T_{p_2}$): a finite set of transitions in $p_1$ (or $p_2$).

Functions:
- $\text{getTransitions}(p, s)$: to get all transitions in a service protocol $p$, such that the source state of each transition is the state $s$.
- $\text{getPolarity}(p, m)$: to get the polarity (“+” for incoming or “−” for outgoing) of a certain message $m$ in a service protocol $p$.
- $\text{clean(adapt}\textsubscript{graph})$: to remove from adapt\textsubscript{graph} all states, as well as related messages, transitions and conditions, that cannot lead to any final state.

![Figure 4 adapt\textsubscript{graph}: The Complete Adaptation Graph for SP and SR according to APT](image-url)
Now we introduce Algorithm 1 that generates the complete adaptation graph \( \text{adapt}_\text{graph} \) for two input service protocols \( p_1 \) and \( p_2 \). Algorithm 1 examines each candidate in \( \text{Cand} \). The set \( \text{Cand} \) initially contains one tuple \((s_1, \emptyset, s_2, \emptyset)\) where, \( s_1 \) and \( s_2 \) are the initial states of \( p_1 \) and \( p_2 \), respectively. \( \emptyset \) means an empty set. \( M_{p_1} \) and \( M_{p_2} \) are empty since no message has been exchanged at this stage (line 1).

Given a candidate \((s_{p_1}, M_{p_1}, s_{p_2}, M_{p_2})\) in \( \text{Cand} \), we retrieve all transitions in \( p_1 \) (and \( p_2 \)) such that \( s_{p_1} \) (and \( s_{p_2} \)) is the source state of these transitions (line 3). The processing of this candidate is as follows:

1. We first deal with paired transitions in \( T_{p_1} \) and \( T_{p_2} \) (lines 4-11). A pair of transitions means that, a transition \( \tau_1 = (s_{p_1}, s_{t_1}, m, + / -, c_{p_1}) \) exists in \( T_{p_1} \), a transition \( \tau_2 = (s_{p_2}, s_{t_2}, m, + / -, c_{p_2}) \) exists in \( T_{p_2} \), and \( \tau_1 \) and \( \tau_2 \) relate to the same message \( m \) but with different polarities (line 4). \( m \) is exchanged from \( p_1 \) to \( p_2 \) or vice versa through \( \text{APT} \) (lines 5-10). If this pair of transitions \( \tau_1 \) and \( \tau_2 \) has not been used to generate a new transition in \( \text{adapt}_\text{graph} \) (line 6), \( \tau_1 \) and \( \tau_2 \), and the corresponding states, messages and conditions, are incorporated into \( \text{adapt}_\text{graph} \). The target states and the updated message sets constitute a new candidate (line 8).

2. We then handle the remaining transitions in \( T_{p_1} \) and \( T_{p_2} \) (lines 12-26). We show in lines 12-25 how to handle the transitions in \( T_{p_2} \). Transitions in \( T_{p_1} \) are dealt with similarly (line 26).

   - For an outgoing transition in \( T_{p_1} \), \( p_1 \) sends the related message \( m_1 \) to \( \text{APT} \) (lines 13-15). The message set related to \( s_{p_2} \) (i.e., \( M_{p_2} \)) is updated by incorporating \( m_1 \) since \( p_1 \) may consume \( m_1 \) afterwards.
   - For an incoming transition in \( T_{p_2} \), if the related message \( m_1 \) has been sent by \( p_1 \) (i.e., \( m_1 \in M_{p_2} \)), \( \text{APT} \) forwards \( m_1 \) to \( p_1 \) (lines 16-18). Otherwise, if the related message is an \( \text{ACK} \) message (line 19), \( \text{APT} \) generates such \( \text{ACK} \) and forwards it to \( p_1 \) (lines 19-21).

   If this outgoing or incoming transition \( \tau \) has not been used to generate a new transition in \( \text{adapt}_\text{graph} \) (lines 13, 16, and 19), \( \tau \), with the corresponding states, messages and conditions, are incorporated into \( \text{adapt}_\text{graph} \). The target state of \( \tau \), and the updated message sets, constitute a new candidate (lines 15, 18, and 21).

3. The current candidate is removed from the candidate set (line 27).

After handling all candidates in \( \text{Cand} \), the function \( \text{clean} \) is applied for removing from \( \text{adapt}_\text{graph} \).
all states, as well as the related messages, transitions and conditions, which cannot lead to any final state (line 29). In addition, the final states of adapt\(_{\text{graph}}\) are derived from \(S\) (line 29). The remaining graph is a deterministic FSA since lines 7, 14, 17, and 20 guarantee that each transition relates to one source and one target states. Note that a candidate can be generated (lines 8, 15, 18, and 21) if the related transition has not been incorporated into adapt\(_{\text{graph}}\). Hence, a candidate is being generated and processed once.

When handling the transitions in \(p_s\) and \(p_t\), the target state pair may already be in \(S\) (lines 9 and 23). There are two reasons for this situation: (i) the target state pair has been explored by other source state pairs, or (ii) some transitions share the source and target states, e.g., the two transitions goldenPrice(\(-\)) and normalPrice(\(-\)) in \(SP\) share the source and target states as Cust. Info. Handling and Price Processing.

### 4.2 Computational Complexity Analysis

The time complexity of Algorithm 1 is \(O(k^2 \times n^4)\) in the worst case, as shown in Table 1. \(k\) is the upper bound number of transitions between a pair of source and target states. \(n\) is the maximum number of states in service protocols. Service protocols \(p_s\) and \(p_t\) are strongly connected in the worst case, i.e., for any two states \(s_1\) and \(s_2\) in a service protocol except the initial and final states, \(k\) transitions link \(s_1\) to \(s_2\) and vice versa. Then, there are \(k \times (n - 1) \times (n - 2)\) transitions in this service protocol, and \(n^2\) state pairs to be checked. For each state pair, \(2k \times (n - 2)\) candidates are possibly generated. Hence, there are \(2k \times n^2 \times (n - 2)\) candidates at most to be examined.

<table>
<thead>
<tr>
<th>Table 1 Computational Complexity of Steps</th>
<th>time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>line 3</td>
<td>(O(k \times n^2))</td>
</tr>
<tr>
<td>lines 3-27</td>
<td>(O(k^2 \times n^4))</td>
</tr>
<tr>
<td>line 29</td>
<td>(O(n))</td>
</tr>
<tr>
<td>Algorithm 1</td>
<td>(O(k^2 \times n^4))</td>
</tr>
</tbody>
</table>

Next we explore the time complexity of handling a candidate. The time complexity of line 3 is \(O(k \times n^2)\) since there are \(k \times (n - 1) \times (n - 2)\) transitions at most in a service protocol. There are \(k \times (n - 1)\) transitions at most in \(T_{S_{p_1}}\) and \(T_{S_{p_2}}\) for each candidate. The complexity of handling one (pair of) transition(s) depends on line 6, 13, 16, and 19, where there are \(2k \times (n - 1) \times (n - 2)\) transitions at most in \(T\). Hence, the time complexity of handling a candidate is \(O(k^3 \times n^4)\).

Consequently, the overall time complexity is \(O(k^3 \times n^4)\) in the worst case. The function clean(adapt\(_{\text{graph}}\)) used in line 29 can be reduced to a graph reachability problem [26] which can be implemented efficiently.

As observed by [2][18][20][23][31][37][47] and according to our survey presented in Section 7.1, a service protocol is usually a fairly simple model. Indeed, a service protocol, as well as a service in general, is designed by human beings. Hence, \(k\) is typically quite small. In addition, the number of states in a service protocol is typically not large. Indeed, \(n\) is typically fewer than 100 [45], and \(n\) ranging from 50 to 100 refers to a large service protocol [2][22]. Hence, Algorithm 1 is applicable to construct the complete adaptation graph for service protocols.

We realize that the size of the complete adaptation graph (i.e., the number of states and transitions) is not large. Since transitions in a service protocol are constrained by dependencies, a state in one service protocol can combine with limited states (assuming \(l\) is the upper bound) in another service protocol to form states and transitions in the complete adaptation graph. Hence, the complete adaptation graph has \(l \times n\) states at most, with \(k\) transitions between a pair of states at most.

### 5 Walk Computation

In this section, we introduce our walk computation technique, which is used to compute all instance sub-protocols in a service protocol, and to compute all instance sub-protocol pairs captured by the complete adaptation graph. Below we introduce what an instance sub-protocol is:

**Definition 5.1 (Instance Sub-protocol).** An instance sub-protocol of a service protocol \(p = (M, S, s, F, C, T)\) is a tuple ISP = \((M, S, s, f_i, C, T_i)\) where:

1. \(M_i \subseteq M, S_i \subseteq S, s_i = s, f_i \subseteq F, C_i \subseteq C, T_i \subseteq T\).
2. There exists an interaction of \(p\) where \(T_i\) is the set of transitions in \(p\) involved in this interaction, such that \(T_i = T_f\).

Before presenting our walk computation technique, we introduce two background concepts path and walk. Generally, a path in a service protocol \(p\) is an alternating sequence of states and transitions of \(p\) starting at the initial state of \(p\) and ending at one of its final states. A path reflects a possible execution of \(p\). The states and transitions in this path are the same as those of a certain instance sub-protocol. Given an instance sub-protocol that contains loop blocks, the instance sub-protocol corresponds to many, or infinite, paths, which are different because of the traversal numbers of states and/or transitions. Hence, an instance sub-protocol corresponds to a bundle of paths which have the same sets of states and transitions.

A walk in a graph is a sub-graph of this graph, such that there exists a path of this graph and this path contains the same sets of vertices and edges as this sub-graph [6] [15]. A service protocol \(p\) can be reduced to a directed
cyclic graph when considering states and transitions only. Then, an instance sub-protocol in p is a walk in p, such that they share the same sets of states and transitions. Hence, the problem of generating all instance sub-protocols in p is reduced to the problem of computing all walks in p. Formally, a walk in p is defined as follows:

**Definition 5.2 (Walk).** Let \( p = (M, s, F, C, T) \) be a service protocol. A walk in p is a tuple \( w_{lk} = (M_{s}, s, f_{a}, C_{a}, T_{a}) \) where:

1. \( M_{s} \subseteq M, S_{a} \subseteq S, f_{a} \subseteq F, C_{a} \subseteq C, T_{a} \subseteq T. \)
2. There exists a path in p which (i) is an alternating sequence of states and transitions of p, and (ii) starts at s and ends at \( f_{a} \). Let \( S_{a} \) be the state set, and \( T_{a} \) be the transition set, of this path. Then:
   - \( S_{a} = S_{a} \) and \( T_{a} = T_{a} \).
   - Let \( M_{s} = \{ m \mid \exists (s_{s}, s, m(+) or -, c) \in T_{a} \}, \) Then, \( M_{s} = M_{s} \).
   - Let \( C_{a} = \{ c \mid \exists (s_{s}, s, m(+) or -, c) \in T_{a} \}, \) Then, \( C_{a} = C_{a} \).

A walk in p requires that (i) the **origin** of the walk is the initial state of p, and (ii) the **terminus** of the walk is one final state of p. If a walk starts or ends at any state of p, the walk may correspond to part of an execution of p which corresponds to part of an instance sub-protocol. Such a walk is not relevant to our adaptability assessment which depends on instance sub-protocols.

In the following sections we present the procedure of computing (i) all instance sub-protocols of a service protocol, and (ii) all pairs of instance sub-protocols captured by the complete adaptation graph, by applying our walk computation technique.

### 5.1 Computing Instance Sub-Protocols in Service Protocol

Given a service protocol \( p \), we compute all instance sub-protocols through the following steps:

- **Step 1: Pre-processing.** We first derive a finite state automata (i.e., FSA) from \( p \) by:
  1. abstracting away conditions and self-loops in \( p \);
  2. folding multiple transitions in \( p \) into a single transition if they share the same source and target states.

We then identify **back transitions** in the generated FSA. A transition \( \tau \) in the generated FSA is considered as a back transition when:

1. its target state is not a final state;
2. the **distance** from the initial state of this FSA to the target state of \( \tau \) is not longer than that to the source state of \( \tau \).

As examples, back transitions in \( SP, SP_A, SR, \) and adaptgraph are marked using dotted lines.

- **Step 2: Walk computation.** We compute all walks in the generated FSA as follows:

We first compute all walks in the generated FSA without considering back transitions. This is achieved by applying a breadth-first search.

Based on walks with \( i \) back transitions \((i = 0, 1, 2, ..., k - 1)\), we compute walks that contain \( i + 1 \) back transitions. \( k \) is the number of back transitions in the generated FSA. Given a walk \( w_{lk} \) with \( i \) back transitions and one of these \( k \) back transition \( \tau \), we compute new walks with \( i + 1 \) back transitions. \( s_{s} \) and \( s_{t} \) denote the source and target states of \( \tau \), respectively. We distinguish between four cases to compute all walks with \( i + 1 \) back transitions:

- **Case 1:** Neither \( s_{s} \) nor \( s_{t} \) belongs to \( w_{lk} \), or \( \tau \) exists in \( w_{lk} \) already. In this case, no new walk is generated.
- **Case 2:** Both \( s_{s} \) and \( s_{t} \) belong to \( w_{lk} \), but \( \tau \) does not exist in \( w_{lk} \).
- **Case 3:** \( s_{s} \) but not \( s_{t} \) belongs to \( w_{lk} \).
- **Case 4:** \( s_{t} \) but not \( s_{s} \) belongs to \( w_{lk} \).

We present in [49] about the procedure of computing walks for the latter three cases.

- **Step 3: Post-processing.** Based on walks computed in the walk computation step, we generate all instance sub-protocols of \( p \) by:

1. unfolding transitions that are folded in the pre-processing step;
2. considering self-loops, and then, (iii) integrating conditions that are abstracted away in the pre-processing step.

We detail in [49] about how to compute all instance sub-protocols for a service protocol. As an example, Figure 5 shows the procedure of computing all instance sub-protocols for \( SR \). We do not present Step 3: Post-processing in Figure 5, since this step is just to integrate conditions with the corresponding transitions.

By applying the above steps, 27 instance sub-protocols for \( SP, 5 \) instance sub-protocols for \( SP_A \), and 4 instance sub-protocols for \( SR \), are computed. Figure 6 shows two instance sub-protocols where, \( isp_{sp} \) and \( isp_{sr} \) correspond to the instance sub-protocols in \( SP \) and \( SR \), respectively.

Except Step 2, the other two steps are computationally relatively light (i.e., linear or polynomial time). Step 2 takes exponential time to the size of the service protocol in general. It is worth mentioning that, according to our survey presented in Section 7.1, and as discussed in [37] [45], a service protocol tends to be small in size and simple in model, since a service protocol in particular, and a service in general, are designed by humans. Hence, our walk computation technique is feasible to be applied in the context of adaptability assessment of service protocols.

### 5.2 Computing Instance Sub-Protocol Pairs in Adaptation Graph

Given the instance sub-protocols of service protocols generated in the previous section, we compute all pairs of
Figure 5 Instance Sub-protocol Computation for SR

Figure 6 Instance Sub-protocols of SP and SR
instance sub-protocols captured by the complete adaptation graph. The steps are as follows:

- **Step 1:** We compute all mediated service interactions captured by the complete adaptation graph by using our walk computation technique presented in the previous section. A mediated service interaction is reflected by a walk of the complete adaptation graph, and this walk starts at the initial state of the complete adaptation graph and ends at one of its final states. For $adapt_{graph}$ as shown in Figure 4, 15 mediated service interactions are computed. In Figure 4, we specify a mediated service interaction by marking its states and transition in terms of bold lines. We use the symbol: $medInt$, to denote it.

- **Step 2:** For a certain mediated service interaction, we project it to a certain service protocol $p$. The result is a complete execution path [5], which starts at the initial state and ends at one final state of $p$. The projection is an operator [5] which identifies transitions in a mediated service interaction associated with a service protocol, and restores their polarities.

  For instance, $SP \rightarrow SR$: normalPrice in $adapt_{graph}$ is projected to one transition in $SP$ (i.e., normalPrice(−)) and another transition in $SR$ (i.e., price(+)), and $SP \rightarrow APT$: schedShip is projected to a transition in $SP$ (i.e., schedShip(−)).

- **Step 3:** Given a complete execution path, we identify which instance sub-protocol of a service protocol this path corresponds to. This is achieved by comparing the state and transition sets of this complete execution path with those of an instance sub-protocol of the service protocol. Consequently, the pair of instance sub-protocols that a certain mediated service interaction corresponds to is determined.

  For $medInt$, the complete execution path with respect to $SP$ corresponds to $ispSP$ shown in Figure 6, and the complete execution path with respect to $SR$ corresponds to $ispSR$. For $adapt_{graph}$, 15 mediated service interactions are projected to 10 pairs of instance subprotocols where, $isp_{sp}$ and $isp_{sr}$ is a pair of instance sub-protocols involved in $medInt$.

  When the pairs of instance sub-protocols captured by the complete adaptation graph are determined, the adaptability of service protocols is ready to be assessed.

### 6 Adaptable Assessment

In this section we assess the adaptability by computing an adaptation degree and providing a condition set. Given two service protocols $p_1$ and $p_2$ and their complete adaptation graph $adapt_{graph}$, an adaptation degree is computed by means of Equation (1). The function $ispComp(p_1, adapt_{graph})$ counts the number of all instance sub-protocols in $p_1$ that are captured by the instance sub-protocol pairs of $adapt_{graph}$. If the parameter $adapt_{graph}$ is set to null, the number of instance sub-protocols in $p_1$ is returned. We have presented how to generate (i) all instance sub-protocols of a service protocol (such as $SP$ and $SR$), and (ii) all pairs of instance sub-protocols captured by the complete adaptation graph (like $adapt_{graph}$), in Section 5.

$$adaptDegree(p_1, p_2) = \frac{ispComp(p_1, adapt_{graph})}{ispComp(p_1, null)} \quad (1)$$

Adaptable is an asymmetric relation between service protocols. For $SP$ and $SR$, $adaptDegree(SP, SR) = 10/27 \approx 0.37$, whereas $adaptDegree(SR, SP) = 4/4 = 1$. For $SP$ and $SR$, $adaptDegree(SP_A, SR) = 2/5 = 0.4$, whereas $adaptDegree(SR, SP_A) = 1/4 = 0.25$. Hence, given a requester service like $SR$, the adaptation degree distinguishes between candidate provider services (such as $SP$ and $SP_A$) according to their different adaptation possibilities.

We explore the problem of distinguishing between partial and full adaptability depending on an adaptation degree. When $adaptDegree(p_1, p_2) = 1$, $p_1$ is fully adaptable with $p_2$, since each instance sub-protocol in $p_1$ can perform an interaction with at least one instance sub-protocol in $p_2$. When $adaptDegree(p_1, p_2) = 0$, $p_1$ is not adaptable with $p_2$, since no instance sub-protocol in $p_1$ can have an adaptable instance sub-protocol in $p_2$. Otherwise, $p_1$ is assumed partially adaptable with $p_2$.

An adaptation graph captures message exchanges between the pairs of instance sub-protocols. Given a pair of instance sub-protocols, the conjunction of corresponding conditions associated with their transitions is another prerequisite for ensuring a mediated service interaction. Such a must-be-held condition set is generated through examining all pairs of instance sub-protocols.

For $SP$ and $SR$, the condition set is $CND = \{Cd1 \land Cd3 \land Cd5, Cd2 \land Cd3 \land Cd5, Cd1 \land Cd3 \land Cd5 \land Cd7, Cd2 \land Cd3 \land Cd5 \land Cd7, Cd1 \land Cd6, Cd2 \land Cd6, Cd1 \land Cd6 \land Cd7, Cd2 \land Cd6 \land Cd7, Cd1 \land Cd2 \land Cd6 \land Cd7\}$. For $SP$ and $SR$, the condition set is $\{Cd1 \land Cd3 \land Cd5, Cd2 \land Cd3 \land Cd5\}$.

As depicted at Figure 4, the mediated service interaction $medInt$ is marked using bold lines on their states and transitions. The must-be-held condition with respect to $medInt$ is $Cd2 \land Cd3 \land Cd5$. Note that some conditions in $CND$, such as $Cd1 \land Cd2 \land Cd3 \land Cd5 \land Cd7$, contain both $Cd1$ and $Cd2$. This indicates that, in some mediated service interactions, when the transition $adjustSDItem(-)$ in $SR$ has been executed and $SP$ loops back to the state $Cust. Info. Handling$, the
condition $Cdl$, which cannot be satisfied previously, can be satisfied at this stage.

7 Experiment

In this section we present our survey about the size (i.e., the number of activities) and the complexity (i.e., the number of instance sub-protocols) of BPEL service protocols. The result shows that they are neither large in size nor complex with respect to their number of instance sub-protocols. We then introduce our prototype implementation.

7.1 Survey: BPEL Service Protocols

A survey about the size of BPEL processes and the number of their instance sub-protocols is shown in Table 2. The column “Source” indicates where the samples come from and the column “BPEL Process” shows the name of the sample processes. The columns “#activity” and “#ISPs” represent the number of activities, and the number of instance subprotocols, in these BPEL processes, respectively.

We get two BPEL processes from [42] used by Tan et al. to verify their adaptation mechanisms, three from SUPER project used by its use cases, two from active endpoints\(^1\), two from OMII BPEL project\(^2\), and four from Oracle SOA DEMO\(^3\).

Apart from $SOAOrderBooking$, the other samples have small number of both activities and instance sub-protocols. We take a close look at $SOAOrderBooking$. This BPEL process has thirty-nine activities, including two empty activities and another twenty-two assign activities. Since empty and assign activities represent internal operations, they are not relevant to exchanging messages between service protocols. There are only fifteen activities which aim to exchange messages with other service protocols. Because of nested Switch blocks, $SOAOrderBooking$ has sixteen instance subprotocols. However, most branches of the Switch blocks specify different internal operations, and hence, some instance sub-protocols are the same if considering message exchanges with other service protocols only. The context of this paper is the interaction between service protocols, and hence, the difference of internal operations is not relevant. Consequently, there are only two instance sub-protocols which are different considering possible interactions with other service protocols.

Besides, Figure 7 in [20] presents twelve realistic protocols. They are consistent with our survey that a service protocol is normally not large in size (i.e., the protocol largest in size has only twelve states and fifteen transitions).

7.2 Implementation

We have implemented the technique presented in this paper and have integrated the prototype as a component of ProM (see http://prom.win.tue.nl/tools/prom/). ProM is a leading process mining toolkit initialized by the Process Mining Group at Eindhoven Technical University. The interested reader can get the source code which is available

<table>
<thead>
<tr>
<th>Sample set</th>
<th>BPEL process</th>
<th>#activity</th>
<th>#ISPs</th>
</tr>
</thead>
<tbody>
<tr>
<td>eBay service</td>
<td>6</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>TPC service</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>Fulfilment</td>
<td>12</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>OrderFulfillment</td>
<td>16</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>ContentProvision</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>loanApprovalProcess</td>
<td>5</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>marketplace</td>
<td>6</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>conditionalworkflow</td>
<td>8</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>echoworkflow</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>BPELProcess1</td>
<td>3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>DHLShipment</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>SelectManufacturer</td>
<td>3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$SOAOrderBooking$</td>
<td>39/15</td>
<td>16/2</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\) http://active-endpoints.com/cec/samples/content/sample-BPEL_Samples/doc/index.html
\(^2\) http://sse.cs.ucl.ac.uk/projects/omii_bpel/
\(^3\) http://www.oracle.com/technology/sample_code/products/bpel/index.html
at the following link: http://www.deri.ie/fileadmin/documents/Prom_Plugin_for_EPC_adaptation.rar. We use EPC Markup Language (EPML) to model service protocols and the adaptation graph. EPML has three first-class elements: (i) Function which is used to represent state, (ii) Event which is used to represent transition, and (iii) Arc which is used to represent link, in service protocols and the adaptation graph. We have added the attribute: condition, to Event to represent conditions associated with transitions.

Figure 7 shows the generated complete adaptation graph for SP and SR in EPML format, where part of this EPML representation is shown in Figure 8.

The procedure of instance sub-protocol computation for service protocols SP, SP_A and SR can be finished within one second. Generally, common use case scenarios, like the service protocols surveyed in the previous section, and the service protocols used in [13][17][27][35-36][42] to evaluate their adaptation mechanisms, are simpler than our service protocols SP and SR depicted at Figure 1. Hence, applying our approach on these use cases to assess the adaptability of their service protocols takes less computation effort than SP and SR. Through assessing the adaptability of a broader and more complicated example in this paper, we show the applicability of our approach on the existing use cases.

Next we present some experiments about the walk computation technique we proposed. Figure 9 shows some sample graphs with same size but different complexity. The corresponding number of walks to be computed, and the computation time, are presented in Table 3. The symbol #walks means the number of walks to be computed, and #time (millisecond) specifies the time that the walk computation technique applies for computing these walks.

8 Related Work

We classify related techniques to the following categories: adaptability analysis and adapter synthesization.

8.1 Adaptability Analysis

The work similar to ours is the adapter compatibility analysis [48], which checks whether two component protocols are adaptable with a certain adapter. The criteria are (i) no unspecified receptions and (ii) deadlock free. No unspecified receptions is restrictive since message production and consumption in mediated service interactions are time-decoupled, and extra messages are usually allowed. Our approach does not have such a limitation. Since conditions in protocols are not considered, this work does not specify conditions that determine when two protocols are adaptable. This work assumes the
Figure 8 Partial EPML Representation of the Complete Adaptation Graph for SP and SR

Figure 9 Graphs with Same Size but Different Complexity

Table 3 The Number of Walks, and the Computation Time, for Sample Graphs

<table>
<thead>
<tr>
<th>Graph</th>
<th>#walks</th>
<th>#time (millisecond)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 9(a)</td>
<td>62</td>
<td>516</td>
</tr>
<tr>
<td>Figure 9(b)</td>
<td>126</td>
<td>1,859</td>
</tr>
<tr>
<td>Figure 9(c)</td>
<td>232</td>
<td>3,994</td>
</tr>
</tbody>
</table>
synchronous semantics, which simplifies the problem, but cannot capture most Web service interactions since they are typically asynchronous. Our approach does not depend on this assumption.

A family of service composition techniques [24-25][42] mediates service compositions, thanks to an observation that service interfaces and protocols usually do not fit with each other exactly, although their functionalities can be complementary. Therefore, an adaptation module is generated and deployed between services to reconcile mismatches. The existence of an adapter means that a mediation-aided composition [42] is achievable. Indeed, these techniques provide a framework that facilitates service composition through an adapter. They complement our adaptability assessment by providing an execution environment, but, an adaptability assessment is out of their scope.

8.2 Adapter Synthesization

Techniques that reconcile mismatches between service protocols can be categorized into: (i) techniques that focus on synthesizing adapters, and (ii) techniques that reconcile mismatches by proposing solutions against mismatch patterns.

8.2.1 Synthesizing Adapters

In [48], the authors propose to synthesize an adapter which reconciles mismatches between incompatible protocols. Interface mapping rules, which are provided by the developer, are used to synthesize adapters. The adapter captures all mediated service interactions between incompatible protocols according to its adaptation mechanisms. This is the first work that synthesizes an adapter automatically.

In [13][39], given component interfaces encoded in labeled transition systems and an adaptation contract, the protocol of an adapter is generated. An adaptation contract is an abstract description of constraints that must be respected in protocol interactions. In [8-9], a formal approach is proposed where component’s behaviors are described using process algebra, and an adapter is built automatically. Following this research line, the authors propose to specify service adaptation contracts semi-automatically [11], and extend the mismatch reconciliation to the runtime [12].

In [29], business processes are viewed in terms of standard patterns. A minimal adapter is generated if business processes are incompatible. Similarly, a minimal adapter is generated in [41] by processing only messages that cause deadlocks. In [21], the authors propose an inspiring approach to synthesize adapters depending on domain-specific transformation rules, i.e., synthesizing an adapter by means of consistently separating data and control, and reducing the problem of adapter synthesization to that of controller synthesization. Besides, [25][40] are promising techniques that synthesize an adapter incrementally if protocols are evolved dynamically. A virtual provider, or a mediator, is proposed in [1] which is defined as a high-level behavioral interface based on abstract state machines [7]. In [19], using deletion and resolution operations, the authors propose a mediator based approach to resolve the incompatibility among behavioral interfaces.

A bundle of techniques [10][42] generates an adapter to reconcile mismatches between services. The existence of an adapter means that a mediation-aided composition [42] can be achieved. These techniques provide a framework that enables an adapter to facilitate service compositions and interactions. They complement the techniques synthesizing adapters by providing an execution environment.

In summary, these techniques aim to synthesize adapters. Synthesizing an adapter means that some interactions between protocols can be conducted by means of this adapter, but it cannot specify which set of interactions between protocols can be conducted and under which conditions. Techniques synthesizing adapters constitute a starting point, but are insufficient, to assessing the adaptability.

8.2.2 Reconciling Mismatches against Mismatch Patterns

A family of techniques classify mismatches between service protocols into a set of mismatch patterns [4][17][32]. Then, adaptation templates [4] or composable adaptation operators [17] are proposed to reconcile these mismatch patterns. Mismatches between two protocols are identified by a developer and the adapter is built manually [27][35].

Apart from the mismatch patterns presented in [4][17], the adapter proposed in [36] can reconcile some deadlock situations using mechanisms called evidences. An evidence is a mechanism that aims to (i) identify messages in common deadlocks occurred in Web service interactions, and (ii) to reconcile a deadlock by generating a message for a service protocol which is expecting such message. Deadlock solution in [36] assumes that business data recommended by an evidence is consistent with the certain interaction context. However, business data recommended by some evidences, such as enumeration with default and log based value/type interface, may violate this assumption, since enumeration business data may not be the default in an interaction, and business data may differ in different interactions. Hence, this solution is correct syntactically (i.e., an interaction can terminate), but may be wrong semantically (i.e., the result is not correct from a functional perspective [28]).

Our space-based process mediator produces a mock-up message [50] to resolve a deadlock, and the mock-up
message is consistent with the certain interaction context. Assuming a message \( \text{msg} \) is to be received by an activity \( \text{act} \), and \( \text{actfol} \) is an activity immediately following \( \text{act} \). We allow to provide a mock-up \( \text{msg} \) if (i) \( \text{msg} \) is not used by the input of \( \text{actfol} \) and the incoming condition of \( \text{actfol} \), or (ii) the message type of \( \text{msg} \), instead of the data in \( \text{msg} \), may be used by the incoming condition of \( \text{actfol} \). This means that the enabling of \( \text{actfol} \) does not depend on (i) the execution of \( \text{act} \) and (ii) the message provided by \( \text{act} \). A mock-up \( \text{msg} \) is generated for \( \text{act} \), and then \( \text{actfol} \) can be enabled. This mismatch solution is aligned with the specification of service protocols, and does not bias the functionality an interaction would achieve. Hence, it is correct semantically. It is worth noting that, given a mock-up message generated for reconciling a deadlock, if it is not replaced by a message provided by users afterwards, it may have negative impact to the following process that depends on this message.

In summary, these techniques propose adaptation mechanisms to reconcile mismatches between protocols. These adaptation mechanisms are used in the adaptability assessment when constructing an adaptation graph.

9 Conclusion

In this paper we proposed to assess the adaptability of service protocols depending on the walk computation technique we developed. Our approach provides a set of conditions and computes an adaptation degree. Based on the set of conditions, the requestor can filter candidate provider services by examining whether interactions fulfilling her requirements can be conducted or not. Then she can select the most suitable provider service among the functionally equivalent candidates according to their adaptation degrees. The selected provider service can adapt with the requestor service to fulfill the requestor’s requirements. When a requestor is not very clear about her requirements, an adaptation degree is the criterion for ranking provider services.

As to the future work, there are an increasing number of services published and accessible on the web [30],[46]. Web services can be regarded as a kind of semantic nodes, and a network of services forms a semantic link network [52] [53]. How to (self-)organize such semantic link network leveraging the relation of adaptability is interesting.

Acknowledgements

This work was partially supported by the Fundamental Research Funds for the Central Universities (China University of Geosciences at Beijing), the National Science Foundation of China (Grant No. 61170296 & 60873241), the Fund of Aeronautics Science (Granted No. 20091951020), and the Program for New Century Excellent Talents in University (Grant No. 291184).

References

[12] Javier Camara, Gwen Salaun and Carlos Canal, Composition and Run-Time Adaptation of Mismatching Behavioural Interfaces, J. Universal
Service Protocol Adaptability Assessment for Supporting Mediated Service Interactions


Biographies

Zhangbing Zhou is an associate professor at the School of Information Engineering, China University of Geosciences (Beijing), China. He received his PhD from Digital Enterprise Research Institute (DERI), National University of Ireland, Galway (NUIG). After receiving his Master of Engineering from the Institute of Automation, Chinese Academy of Sciences in Beijing, 2000, he worked as a software engineer at Huawei Technologies Co. Ltd. from 2000 to 2001, and was served as a Member of Technical Staff and a team leader at Bell Labs China, Lucent Technologies (now Alcatel-Lucent) from 2001 to 2006. His research interests include process-aware information system, service-oriented computing, cloud computing, and sensor network middleware. He has published more than 30 research papers. He serves as a track chair, a program committee member, and a reviewer at many international conferences and journals and has been participating in several national and European research projects.
Lei Shu received the BSc degree in Computer Science from South Central University for Nationalities, China, in 2002 and the MSc degree in Computer Engineering from Kyung Hee University, Korea, in 2005 and the PhD degree in Digital Enterprise Research Institute, from National University of Ireland, Galway, in 2010. He is currently a specially assigned researcher in Department of Multimedia Engineering, Graduate School of Information Science and Technology, Osaka University, Japan. He has published over 100 papers in the area of sensor networks. He had been awarded the MASS 2009 IEEE TCs Travel Grant and the Outstanding Leadership Award of EUC 2009 as Publicity Chair, the Globecom 2010 Best Paper Award. He has served as editors for Wiley, European Transactions on Telecommunications, IET Communications, IET Networks, KSII Transactions on Internet and Information Systems (TIIS), Journal of Communications, Wireless Communications and Mobile Computing, Inderscience, International Journal of Sensor Networks. He has served as more than 40 various Co-Chair for international conferences/workshops, e.g., IWCMC, ICC, ISCC, TPC members of more than 150 conferences, e.g., ICC, Globecom, ICCC, WCNC, ISCC. His research interests include wireless sensor network, sensor network middleware, multimedia communication, and security. He is a member of IEEE and IEEE ComSoc.

Jianwei Niu received his MS and PhD degrees in 1998 and 2002 in computer science from Beijing University of Aeronautics and Astronautics (BUAA, now renamed as Beihang University). He was a visiting scholar at School of Computer Science, Carnegie Mellon University, USA from Jan. 28, 2010 to Feb. 1, 2011. He is a professor in the School of Computer Science and Engineering, BUAA. He is an ACM/IEEE member and has published more than 70 refereed papers and filed more than 20 patents. His current research interests include image/video analysis and mobile computing.