Protocol-Level Service Composition Mismatches: A Petri Net Siphon Based Solution

PengCheng Xiong, Georgia Institute of Technology, USA
Calton Pu, Georgia Institute of Technology, USA
MengChu Zhou, Tongji University, China

ABSTRACT

Protocol-level mismatch is one of the most important problems in service composition. The state-of-the-art method to deal with protocol mismatch is to generate adaptors to check deadlock-freeness based on a reachability graph. When this property is violated, the generation process will repeat itself until no deadlock state is found; however, the main drawback of this method is that it does not take into account the future deadlock state and requires many interactions with a developer. In this regard, it suffers from low efficiency. In this paper, the authors model multiple web service interaction with a Petri net called Composition net (C-net). The protocol-level mismatch problem is transformed into the empty siphon problem of a C-net. The authors take future deadlock states into consideration through this model, while finding the optimal solution that involves fewest interactions with a developer. The proposed method is proved to achieve higher efficiency for resolving protocol-level mismatch issues than traditional ones.

Keywords: Deadlock, Multiple Web Service Interaction, Petri Nets, Protocol-Level Mismatch, Service Composition

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 (Nezhad, 2007). Table 1 gives a detailed taxonomy of already known service composition mismatches.

Interface Mismatches

Interface mismatches include message signature mismatches, message split/merge and message missing/extra mismatches (Benatallah, 2005; Benatallah, 2006) as shown in Figure 1.

Message signature mismatches can be defined as: a service client’s message has a different name and/or data type and/or data range from that of a service provider. For example, the message with the name “ClientOrder” on the service client’s site may correspond to the message with the name “Order”...
on the provider’s site. Note that, the case when the provider’s message has a different data range from that of the client is also defined as parameter constraint mismatch in (Benatallah, 2005).

Message split mismatch can be defined as: a service client requires multiple messages to achieve certain functionality while a service provider can offer only a single message. For example, a provider’s message $mp$ contains $n$ parts, i.e., $mp_1, mp_2, \ldots, mp_n$ while a client considers $mp_1, mp_2, \ldots, mp_n$ as $n$ separate messages.

Message merge mismatch can be defined as: a service client requires a single message to achieve certain functionality while a service provider can offer multiple messages. This is the reverse case of message split mismatch. For example, a client requires message $mc$ that is a combination of messages $mc_1, mc_2, \ldots, mc_n$ from a provider.

Missing message mismatches can be defined as: a service client expects a message that a service provider does not issue. For example, a provider is sending an acknowledgement message that a client does not expect. Extra message mismatches are defined as: a provider issues an extra message that a client does not expect. This is the opposite case of missing message mismatches. For example, a provider is sending an acknowledgement message that a client does not expect. Figure 1 shows the cases for interface mismatches.

Services can be composed if there is no interface mismatches. There are significant research results towards service interface level

---

**Table 1. Taxonomy of service composition mismatches**

<table>
<thead>
<tr>
<th>Service composition mismatches</th>
<th>Interface mismatches</th>
<th>Protocol level mismatches</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>message signature mismatches</td>
<td>unspecified reception</td>
</tr>
<tr>
<td></td>
<td>message split/merge mismatches</td>
<td>mutual waiting mismatches</td>
</tr>
<tr>
<td></td>
<td>message missing/extra mismatches</td>
<td>non-local choice mismatches</td>
</tr>
</tbody>
</table>

---

**Figure 1. Illustration for interface mismatches**

(a) Message signature mismatches

OrderRequest
  \(<\text{ClientOrder}>\) --\(<\text{Order}>\)
  \(<\text{PaymentInfo}>\) --\(<\text{PaymentInfo}>\)

OrderInfo
  \(<\text{Order}>\)
  \(<\text{Acknowledgement}>\)

(b) Missing message mismatches

OrderRequest
  \(<\text{ClientOrder}>\) --\(<\text{Order}>\)
  \(<\text{PaymentInfo}>\) --\(<\text{PaymentInfo}>\)
  \(<\text{Acknowledgement}>\) --? 

OrderInfo
  ? --\(<\text{Acknowledgement}>\)

(c) Extra message mismatches

OrderRequest
  \(<\text{ClientOrder}>\) --\(<\text{Order}>\)
  \(<\text{Order ID}>\)
  \(<\text{Order date}>\)
  \(<\text{Billing Address}>\)
  \(<\text{PaymentInfo}>\)

OrderInfo
  \(<\text{ClientOrder}>\)
  \(<\text{PaymentInfo}>\)
  \(<\text{Shipping Address}>\)
  \(<\text{Billing Address}>\)

(d) Message split mismatch

OrderRequest
  \(<\text{ClientOrder ID}>\)
  \(<\text{ClientOrder date}>\)
  \(<\text{Shipping Address}>\)
  \(<\text{Billing Address}>\)
  \(<\text{PaymentInfo}>\)

OrderInfo
  \(<\text{Order}>\)
  \(<\text{PaymentInfo}>\)

(e) Message merge mismatch
Semantic-Based Access to Composite Mobile Services
[www.igi-global.com/article/semantic-based-access-composite-mobile/58978?camid=4v1a](www.igi-global.com/article/semantic-based-access-composite-mobile/58978?camid=4v1a)

Drivers of Cloud Computing Adoption for E-Government Services Implementation
[www.igi-global.com/chapter/drivers-of-cloud-computing-adoption-for-e-government-services-implementation/217896?camid=4v1a](www.igi-global.com/chapter/drivers-of-cloud-computing-adoption-for-e-government-services-implementation/217896?camid=4v1a)