Institut für Architektur von Anwendungssystemen
Universität Stuttgart
Universitätsstraße 38
70569 Stuttgart
Germany

Studienarbeit Nr. 2437

Implementation of a Transformation from BPEL4Chor to BPEL

Jinhui Huang

Studiengang:	Informatik
Prüfer:	Jun.-Prof. Dr.-Ing. Dimka Karastoyanova
Betreuer:	M.Sc. Wirt.-Inf. Andreas Weiß
begonnen am:	27.08.2013
beendet am:	06.03.2014
CR-Klassifikation:	D.2.11, D.3.3, H.4.1

Abstract

This thesis is engaged with implementing the conceptual approach to transform BPEL4Chor to BPEL. The transformation process takes topology; grounding and PBDs defined in BPEL4Chor as input, and outputs abstract BPEL processes and WSDL file. The transformation process is implemented using JAXB.

Contents

1 Introduction

BPEL4Chor, which has been developed in a cooperation of the Institute of Architecture of Application Systems at the University of Stuttgart and the Hasso-Plattner-Institute at the University of Potsdam, was built to model choreographies of simulation workflows. Choreographies are coordinated interactions between participants, and during the interactions are taking place, there is no central controlling mechanism needed. BPEL (Business Process Execution Language) is a well-known XML-based mark-up language for composing a set of discrete services into process flows, which expose also WSDL interface as Web Services.

The conceptual approach for an auto transformation from BPEL4Chor to BPEL has been proposed in [Rei07], in this thesis, an implementation of this conceptual approach has been introduced. The implementation used Java Architecture for XML Binding (JAXB), which is a Java standard that defines an API for reading and writing Java objects to and from XML documents.

Extending the work of [Rei07], the basis of both two Specifications is introduced in Chapter 2. The implementation is introduced in Chapter 3. In Chapter 4, some of the issues that have been found during the implementation are discussed. The summary of the implementation is given in Chapter 5. The result of the transformation process and some experiments in BPEL can be seen in Chapter 6.

2 Background

In this chapter, the WS-BPEL and BPEL4Chor are introduced, this work has already been done in [Rei07], so in this part the concepts that the author of [Rei07] didn't involve is mainly introduced.

2.1 WS-BPEL 2.0

BPEL (Business Process Execution Language for Web Services, also WS-BPEL) is a language used for specifying business process behavior, BPEL is based on the composition, orchestration, and coordination of Web Services [BPEL 2.0].

With BPEL, business processes can be described in two ways:

1. Executable BPEL processes: They specify the exact details of business processes and can be executed by a BPEL process server. The executable BPEL process can provide the composite Web Services as a new Web Service.

2. Abstract BPEL processes: They are partially specified processes; some of the concrete operational details are hidden. They could be used to describe the observable behavior of services offered by an Executable Process, or to define a process template.

WS-BPEL relies on several XML-based specifications, WSDL 1.1 and XML Schema 1.0 are used for definition of data types, messages and service interfaces. XPath 1.0 is used as the query and expression language.

The BPEL process logic is performed by activities; there are two kinds of activities:

1. Basic activities: they are elemental steps of process. Such as `<invoke>` activity that can invoke Web Service of partner, or `<receive>` activity that can receive request message of partner.

2. Structured activities: they are used to describe in which order a collection of activities is executed. Structured activities contain other activities. Such as `<if>` activity is used for conditional behavior, `<while>` allows activities it contains to be executed repeatedly based on a given condition.

2.1.1 Basic activities

The `<invoke>` activity is used to invoke Web Services. Typically, it invokes `operations` that the Web Services provide.

WSDL defines four `operations` that an endpoint can support:

1. One-way. The endpoint receives a message.

2. Request-response. The endpoint receives a message, and sends a correlated message.

3. Solicit-response. The endpoint sends a message, and receives a correlated message.

4. Notification. The endpoint sends a message. [WSDL1.1]

BPEL only uses one-way and request-response `operations`. [BPEL 2.0]

A business process' <receive> activity is used to provide Web Service to the process' partners, the Web Service is provided through certain operation that is related to the <receive> activity (relating with partnerLink, see also section 2.1.3), this operation can receive the request message of the partners. <receive> activity is one of the inbound message activities (like <receive>, <pick> and <onEvent>).

A business process can only be instantiated by a <receive> activity (or a <pick> activity), so <receive> activity (or a <pick> activity) is also called a *start activity*.

If the process provides a Web Service that contains a request-response operation, the request to this request-response operation that is received by a <receive> activity can be replied by a <reply> activity, both the <receive> activity and the <reply> activity should be related to the same request-response operation. A <reply> activity sends a response to the <invoke> activity that invokes the Web Service (with the request-response operation) that the process provides. If the process provides a Web Service that contains a one-way operation, the request that is sent to this one-way operation can be replied with a <invoke> activity.

The <assign> activity is used to manipulate the data inside the process, like copying data from one variable to another variable, or inserting new data using XPATH expression into a variable.

2.1.2 Structured activity

The <sequence> activity is used to wrap a set of activities which should be performed sequentially. Generally, the set of activities coordinating the flow of messages across the services integrated within the business process is wrapped in a <sequence>.

The <forEach> activity enables to repeatedly execute its contained scope activity particular times. It also makes it possible to invoke Services dynamically (see section 3.2.5).

The introduction to other structured activities can be seen in [Rei07].

2.1.3 PartnerLink, portType, and operation

<partnerLinks> define the shape of a relationship with a partner by defining the portTypes used in the interactions. Each basic activity that could interact with other partner should set the partnerLink attribute properly, the partnerLink should indicate which portTypes are used to receive or send the message. Each partnerLink is associated with one or two portTypes, depends on the operation that the portTypes provide. Firgure 2.1 presents the class diagram of how partnerLink is connected to the operation.

In BPEL, the message is delivered through ports, these ports are bound with portTypes, and these portTypes provide operations, these operations are defined to send or receive the messages. ports, portTypes and operations are defined in WSDL file that describes the service or services that the process provides.

Each partnerLink is identified by its name attribute, if the partnerLinkType specifies one role, the partnerLink has only one myRole attribute, or one partnerRole attribute, in this case, the interaction between the process and its partner is one-way interaction or synchronous interaction. If the partnerLinkType specifies two roles, the partnerLink has both myRole and partnerRole attribute, in this case, the interaction between the process and its partner is asynchronous interaction (using WS-Addressing).

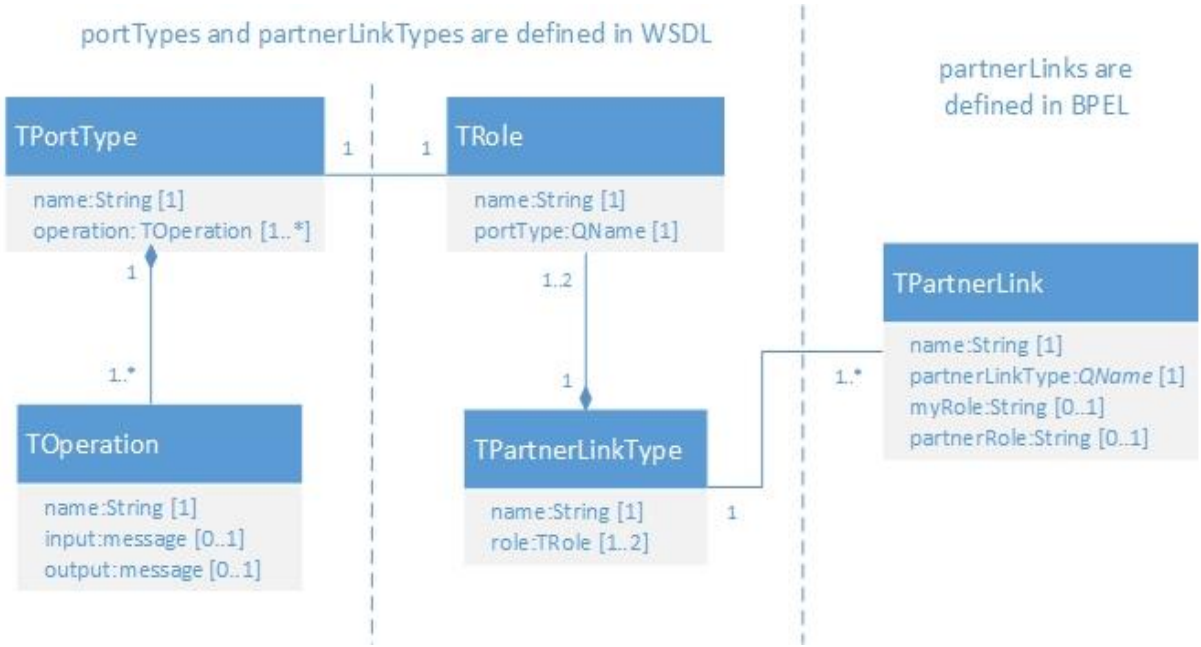


Figure 2.1: class diagram for PartnerLink

The more detailed description of how to declare `partnerLink`s and `partnerLinkType`s will be introduced in following section.

2.2 Interaction pattern between BPEL process

BPEL processes interact with partners in two ways:

- The BPEL process invokes `operation`s on other services or processes.
- The `operation`s on the services that the BPEL process provides, is invoked by other services or clients.

There are some common interaction patterns between BPEL process service component and its partner, the three basic patterns are:

- one-way (using one one-way operation)
- synchronous (using request-response operation)
- asynchronous (using two one-way operations)

They will be introduced as follows, this part of work based mainly on [BB09] and my own experiments (see section 6.1), note that only the basic patterns are introduced, the more complex situations have been left out.

2.2.1 One-way message

The sender process sends a message to the receiver process, and the receiver does not need to reply. The message is received by a One-way `operation`, so the `operation` should be provided by the `portType` that is defined in the receiver process' WSDL.

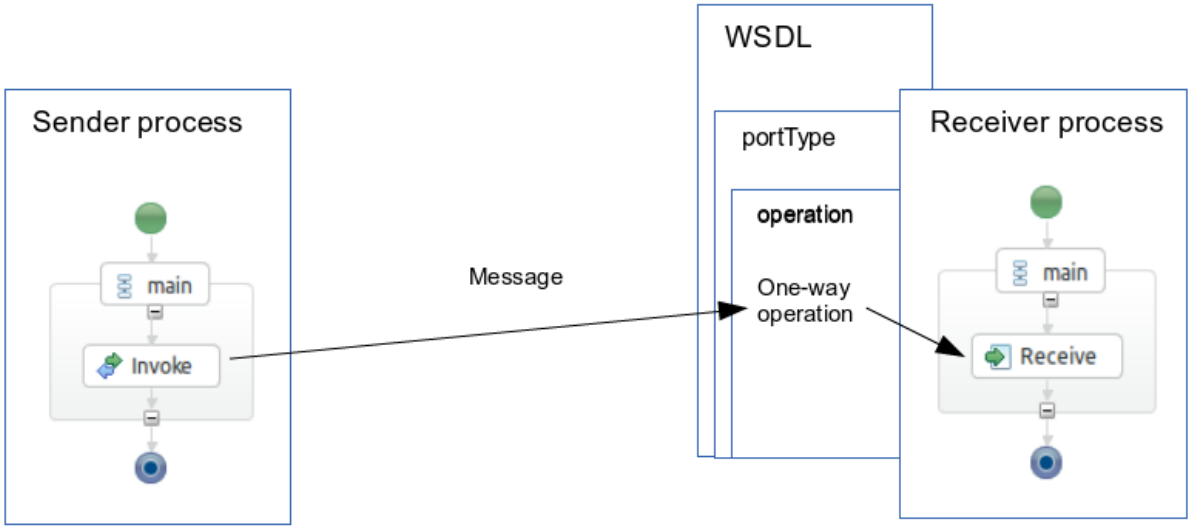


Figure 2.2: One-way message

Example 2.2.1.1 `<invoke>` activity in sender process' BPEL file:

```
<invoke name="invoke"
      partnerLink="sender_receiver_realizedBy_receivePT"
      portType="prefixOfReceiver:receivePT"
      operation="receive"
      inputVariable="message">
</invoke>
```

Example 2.2.1.2 `<partnerLink>` *sender_receiver_realizedBy_receivePT* in sender process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receivePT"
partnerLinkType="prefixOfReceiver:sender_receiver_realizedBy_receivePT_PLT"
      partnerRole="receiver"
</partnerLink>
```

In sender process' `partnerLink` *sender_receiver_realizedBy_receivePT* there is a `partnerRole` attribute set to be *"receiver"*, this indicates that, the operation *"receive"* in `portType` *"prefixOfReceiver:receivePT"* that fulfills the interaction is provided by its partner.

Example 2.2.1.3 `<receive>` activity in receiver process' BPEL file:

```
<receive name="receive"
      partnerLink="sender_receiver_realizedBy_receivePT"
      portType="prefixOfReceiver:receivePT"
      operation="receive"
      variable="message">
</receive>
```

Example 2.2.1.4 `<partnerLink>` *sender_receiver_realizedBy_receivePT* in receiver process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receivePT"
      partnerLinkType="sender_receiver_realizedBy_receivePT_PLT"
      myRole="receiver"
</partnerLink>
```

In receiver process' `partnerLink` *sender_receiver_realizedBy_receivePT* there is a `myRole` attribute set to be *"receiver"*, this indicates that, the `operation` *"receive"* in `portType` *"prefixOfReceiver:receivePT"* that fulfills the interaction is provided by itself.

Example 2.2.1.5 `<partnerLinkType>` *sender_receiver_realizedBy_receivePT_PLT*:

```
<plnk:partnerLinkType name="sender_receiver_realizedBy_receivePT_PLT"
      plnk:role="receiver"
      plnk:portType="prefixOfReceiver:receivePT"
</plnk:partnerLinkType>
```

There is only one `role` attribute is set to *"receiver"*, this indicates that, this interaction only requires one `portType`. The `<partnerLinkType>` could be defined in receiver process' WSDL file, or in an extra WSDL file that only contains the definition of `<partnerLinkType>`, using extensibility mechanism of WSDL 1.1.

2.2.2 Synchronous interaction

The sender process sends a request to its partner, and receives an immediate reply. The sender process is blocked while waiting the reply. The request message is received by a request-response `operation`, and the response message is sent by the same `operation`, so the `operation` should be provided by the `portType` that is defined in the receiver process' WSDL file.

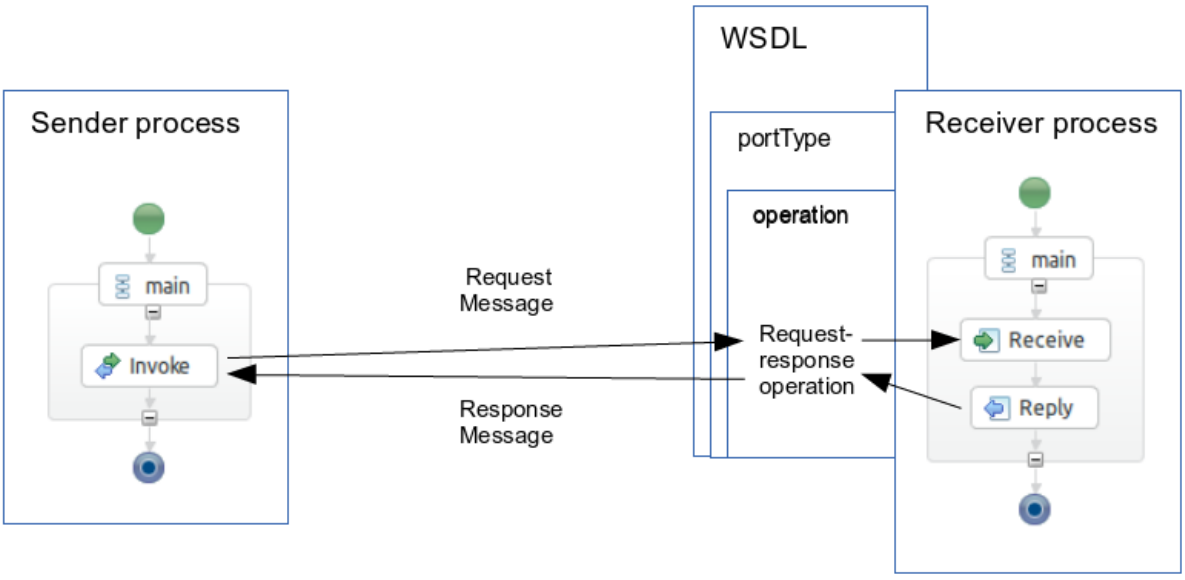


Figure 2.3: Synchronous interaction

Example 2.2.2.1 `<invoke>` activity in sender process' BPEL file:

```
<invoke name="invoke"
      partnerLink="sender_receiver_realizedBy_receiveAndReplyPT"
      portType="prefixOfReceiver:receiveAndReplyPT"
      operation="receiveAndReply"
      inputVariable="requestMessage"
      outputVariable="responseMessage">
</invoke>
```

Example 2.2.2.2 `<partnerLink>` *sender_receiver_realizedBy_receiveAndReplyPT* in sender process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receiveAndReplyPT"
      partnerLinkType="prefixOfReceiver:sender_receiver_realizedBy_receiveAndReplyPT_PLT"
      partnerRole="receiver"
</partnerLink>
```

Example 2.2.2.3 `<receive>` activity in receiver process' BPEL file:

```
<receive name="receive"
      partnerLink="sender_receiver_realizedBy_receiveAndReplyPT"
      portType="prefixOfReceiver:receiveAndReplyPT"
      operation="receiveAndReply"
      variable="requestMessage">
</receive>
```

Example 2.2.2.4 `<reply>` activity in receiver process' BPEL file:

```
<reply name="reply"
      partnerLink="sender_receiver_realizedBy_receiveAndReplyPT"
      portType="prefixOfReceiver:receiveAndReplyPT"
      operation="receiveAndReply"
      variable="responseMessage">
</receive>
```

Example 2.2.2.5 `<partnerLink>` *sender_receiver_realizedBy_receiveAndReplyPT* in receiver process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receiveAndReplyPT"
      partnerLink="prefixOfReceiver:sender_receiver_realizedBy_receiveAndReplyPT_PLT"
      myRole="receiver"
</partnerLink>
```

Example 2.2.2.6 `<partnerLinkType>` sender_receiver_realizedBy_receiveAndReplyPT_PLT:

```
<plnk:partnerLinkType
      name="sender_receiver_realizedBy_receiveAndReplyPT_PLT"
      plnk:role="receiver"
      plnk:portType="prefixOfReceiver:receiveAndReplyPT"
</plnk:partnerLinkType>
```

The synchronous interaction requires only one request-response operation, so there is only one `role` attribute is set to *"receiver"*.

2.2.3 Asynchronous interaction

In an asynchronous interaction, the sender process sends a request message, but does not being blocked, the response message will be received from receiver process later.

As Figure 2.4 demonstrates, this asynchronous interaction consists of two pairs of `<invoke>` and `<receive>` activities. Note the difference between asynchronous and a synchronous interaction: a synchronous BPEL process uses a reply activity to send the response and an asynchronous BPEL process uses an invoke activity.

If there are multiple potential partners that would initiate sender process, then at any time, there might be many active sender process instances. The BPEL server must be able to direct responses to the correct BPEL process service component instance.

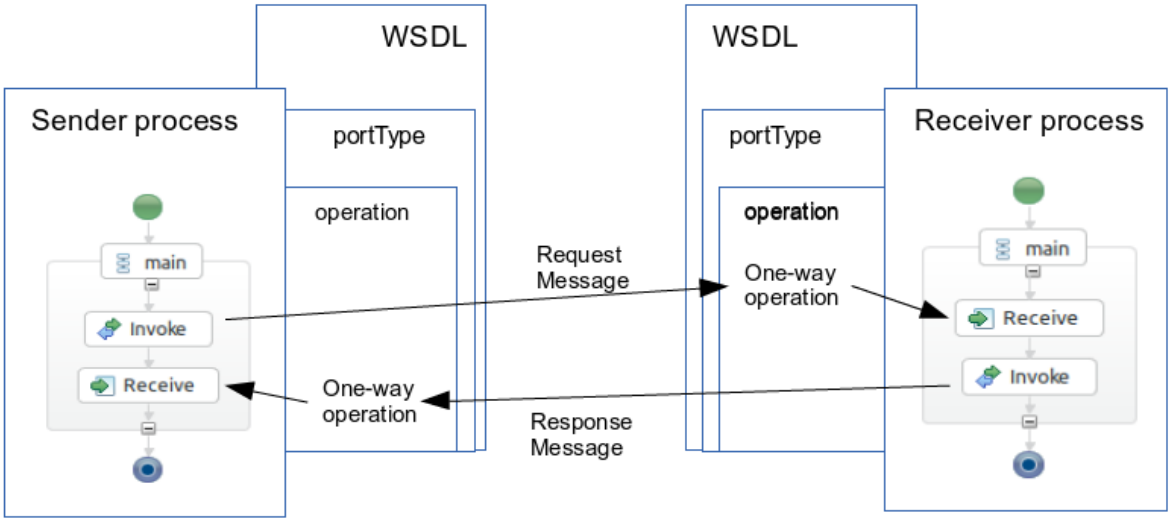


Figure 2.4: Asynchronous interaction

There are two ways to realize an asynchronous interaction: [Sha11], [Sad13]

- WS-Addressing
- Correlation Set

WS-Addressing is a transport-neutral mechanism for addressing Web services and messages [WS-Addressing 1.0]. It can be used to identify asynchronous messages to ensure that asynchronous callbacks are directed to the appropriate client.

According to [BPEL 2.0]:

At the time this specification was completed, various Web Service standards work, such as WSDL 2.0 and WS-Addressing, were ongoing and not ready for consideration for WS-BPEL 2.0. Future versions of WS-BPEL may provide support for these standards.

But WS-Addressing is actually supported by BPEL servers which run WS-BPEL 2.0, like Apache ODE and Oracle SOA Suite [BB09]. And there is actually no explicit declaration for WS-Addressing in BPEL file, the BPEL server will automatically use it, transparent to the BPEL process designer.

When using WS-Addressing method, for an asynchronous interaction, we need to declare one `partnerLinkType` instantiated by two `partnerLink`s, each for a message, in each `partnerLink` we need to define both `myRole` and `partnerRole` attribute, and the value of `myRole` and `partnerRole` attribute in each `partnerLink` are converse. (Appendix 6.1.1: asynchronous interaction using WS-Addressing)

Example 2.2.3.1 `<partnerLink>` *sender_receiver_realizedBy_receivePT* in sender process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receivePT"
      partnerLinkType="prefixOfReceiver:sender_receiver_realizedBy_receivePT_PLT"
      partnerRole="receiver"
      myRole="sender"
</partnerLink>
```

Example 2.2.3.2 `<partnerLink>` *sender_receiver_realizedBy_receivePT* in receiver process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receivePT"
      partnerLinkType="prefixOfReceiver:sender_receiver_realizedBy_receivePT_PLT"
      partnerRole="sender"
      myRole="receiver"
</partnerLink>
```

Note that these two `partnerLink`s could have the same name, because they are declared in different namespaces of different process' BPEL file.

Example 2.2.3.3 `<partnerLinkType>` *sender_receiver_realizedBy_receivePT_PLT*:

```
<plnk:partnerLinkType name="sender_receiver_realizedBy_receivePT_PLT"
      <plnk:role name="receiver"
            portType="prefixOfReceiver:receiveRequestPT"/>
      <plnk:role name="sender"
            portType="prefixOfSender:receiveResponsePT"/>
</plnk:partnerLinkType>
```

As the Example in Appendix 6.1.1 demonstrates, the `portType`s could be separately declared in each participant process' WSDL file.

Example 2.2.3.4 `<portType>` *receiveRequestPT* in receiver process' WSDL file:

```
<portType name="receiveRequestPT"
      <operation name="receiveReqest">
            input message="prefixOfReceiver:requestMessage"/>
      </operation>
</portType>
```

Example 2.2.3.5 `<portType>` *receiveResponsePT* in sender process' WSDL file:

```
<portType name="receiveResponsePT"
      <operation name="receiveResponse">
            input message="prefixOfSender:responseMessage"/>
      </operation>
</portType>
```

The `portType` *receiveResponsePT* is implemented by the sender process for receiving the *responseMessage*, the `portType` *receiveRequestPT* is implemented by the receiver process for receiving the *requestMessage*, the `partnerLinkType` *sender_receiver_realizedBy _receivePT _PLT* indicates that the interaction is an asynchronous one, and is fulfilled by `portType` *receiveRequestPT* and *receiveResponsePT*.

Example 2.2.3.6 `<invoke>` activity in sender process' BPEL file:

```
<invoke name="invoke"
      partnerLink="sender_receiver_realizedBy_receiveRequestPT"
      portType="prefixOfReceiver:receiveRequestPT"
      operation="receiveRequest"
      inputVariable="requestMessage"
</invoke>
```

Example 2.2.3.7 `<receive>` activity in sender process' BPEL file:

```
<receive name="receive"
      partnerLink="sender_receiver_realizedBy_receiveResponsePT"
      portType="prefixOfReceiver:receiveResponsePT"
      operation="receiveResponse"
      variable="responseMessage">
</receive>
```

Correlation Set provides another way to direct responses to the correct BPEL process instance. Correlation Set is a BPEL mechanism that is based on message body contents, they are used for the correlation of asynchronous messages. This method is designed for services that do not support WS-Addressing or for more complex interactions, for example, when the interaction is in the form A > B > C > A instead of A > B > A.

Correlation Set is declarative mechanism which specifies correlated groups of operations within a process instance[BPEL 2.0]. A set of correlation tokens is defined as a set of `properties` using `propertyAlias`, these `properties` are shared by all messages in the correlated group.

As the asynchronous interaction consists of two one-way messages, upon the `partnerLink` and `partnerLinkType` that are declared for the one-way messages, we only need to add proper `<correlation>` and `<correlationSet>` tags into both sender and receiver process' BPEL file. Also, the appropriate `<vprop:property>` and `<vprop:propertyAlias>` tags need to be declared in both sender and receiver process' WSDL file.

Example 2.4.1 `<correlationSet>` *messageID* in sender process' BPEL file:

```
<correlationSet name="messageID"
      properties="prefix:messageID"
</correlationSet>
```

Unlike using the WS-Addressing method, when using Correlation Set method, each `partnerLink` could have only one `partnerRole` or `myRole` attribute. Therefore, the `partnerLinkType` could have only one `role` attribute. (Appendix 6.1.2: asynchronous interaction using Correlation Set)

Example 2.4.2 `<partnerLinkType>` *sender_receiver_realizedBy_receiveRequestPT_PLT* in receiver process' WSDL file:

```
<plnk:partnerLinkType
      name="sender_receiver_realizedBy_receiveRequestPT_PLT"
      plnk:role="receiver"
      plnk:portType="prefixOfReceiver:receiveRequestPT"
</plnk:partnerLinkType>
```

`partnerLinkType` *sender_receiver_realizedBy_receiveRequestPT_PLT* could be defined in receiver process' WSDL file.

Example 2.4.3 `<partnerLinkType>` *receiver_sender_realizedBy_receiveResponsePT_PLT* in sender process' WSDL file:

```
<plnk:partnerLinkType
      name="receiver_sender_realizedBy_receiveResponsePT_PLT"
      plnk:role="sender"
      plnk:portType="prefixOfSender:receiveResponsePT"
</plnk:partnerLinkType>
```

`partnerLinkType` *receiver_sender_realizedBy_receiveResponsePT_PLT* could be defined in sender process' WSDL file.

Note that there are two `partnerLinkTypes` to fulfill an asynchronous communication using Correlation Set.

The `portType` *receiveResponsePT* is implemented by the sender process for receiving the *responseMessage*, it is declared in sender process' WSDL file.

Example 2.4.4 `<partnerLink>` *sender_receiver_realizedBy_receiveRequestPT* in sender process' BPEL file:

```
<partnerLink name="sender_receiver_realizedBy_receiveRequestPT"
      partnerLinkType="prefixOfReceiver:sender_receiver_realizedBy_receiveRequestPT_PLT"
      partnerRole="receiver"
</partnerLink>
```

Note that in Example 2.4.4 `partnerLink` *sender_receiver_realizedBy_receivePT*, `partnerRole` is set to *"receiver"* because the `portType` *prefixOfReceiver:receiveRequestPT* bound with the `role` *"receiver"* is implemented by receiver process.

Example 2.4.5 `<partnerLink>` *receiver_sender_realizedBy_receiveResponsePT* in sender process' BPEL file:

```
<partnerLink name="receiver_sender_realizedBy_receiveResponsePT"
      partnerLinkType="prefixOfSender:receiver_sender_realizedBy_receiveResponsePT_PLT"
      myRole="sender"
</partnerLink>
```

Note that in Example 2.4.5 `<partnerLink>` *receiver_sender_realizedBy_receiveResponsePT*, `myRole` is set to *"sender"* because the `portType` *prefixOfSender:receiveResponsePT* bound with the `role` *"sender"* is implemented by sender process.

Example 2.4.6 `<invoke>` activity using `<correlationSet>` *messageID* in sender process' BPEL file:

```
<invoke name="invoke"
      partnerLink="sender_receiver_realizedBy_receiveRequestPT"
      portType="prefixOfReceiver:receiveRequestPT"
      operation="receiveRequest"
      inputVariable="requestMessage">
      <correlations>
            <correlation set="messageID" initiate="join"> </correlation>
      </correlations>
</invoke>
```

Example 2.4.7 `<receive>` activity using `<correlationSet>` *messageID* in sender process' BPEL file:

```
<receive name="receive"
      partnerLink="sender_receiver_realizedBy_receiveResponsePT"
      portType="prefixOfSender:receiveResponsePT"
      operation="receiveResponse"
```

```
      variable="responseMessage">
      <correlations>
            <correlation set="messageID" initiate="no"> </correlation>
      </correlations>
</receive>
```

2.3 BPEL4Chor

The introduction to BPEL4Chor can be seen in [Rei07], in this section, the usage of BPEL4Chor Designer is introduced. BPEL4Chor Designer generates BPEL4Chor in a model-driven-architecture approach. Based on Graphical Modeling Framework (GMF) and the Graphical Editing Framework (GEF), BPEL4Chor Designer was built as a plugin for Eclipse [Sone13].

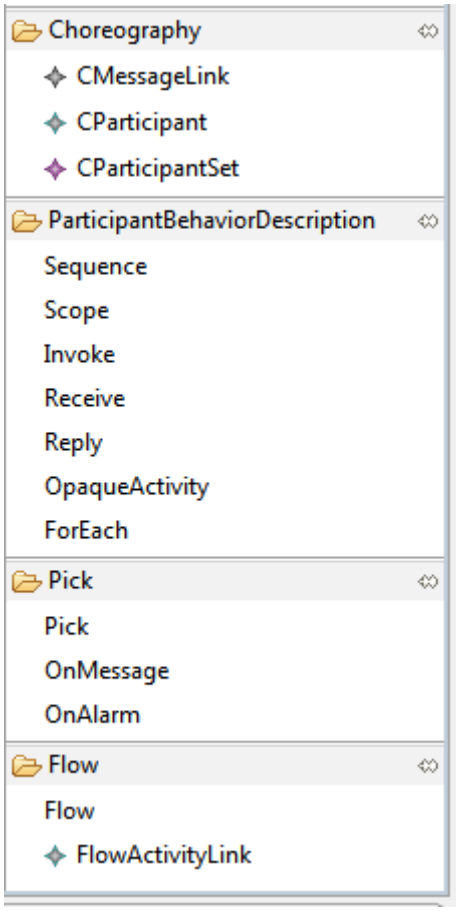


Figure 2.5: Palette view of BPEL4Chor Designer

Figure 2.5 shows all the components that can be modeled by BPEL4Chor Designer. CParticipant, CparticipantSet are used to model participants and participantSets in topology, CMessageLink is used to model messageLinks between the participants or participantSets. All the basic activities that can be modeled by BPEL4Chor Designer are `invoke`, `receive`, `reply` and `opaqueActivity`. The structured activities that can be modeled are `sequence`, `scope`, `forEach`, `pick` and `flow`.

After a participant or participantSet is modeled, a Participant Behavior Description (PBD) with a `sequence` is automatically generated in it (Figure 2.6).

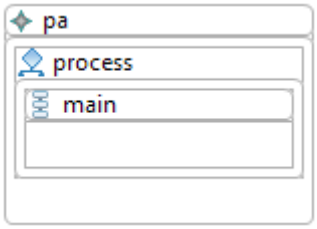


Figure 2.6: PBD in a participant

Then the activities can be inserted to the PBD. Note that for activity `forEach`, a `scope` is automatically generated; for activity `scope`, a `sequence` need to be inserted firstly, then the basic activities can be inserted into this `sequence`; for activities `onMessage` and `onAlarm`, a `sequence` or `scope` must be inserted before any basic activities can be inserted. Figure 2.7 shows a diagram that is generated by the BPEL4Chor Designer.

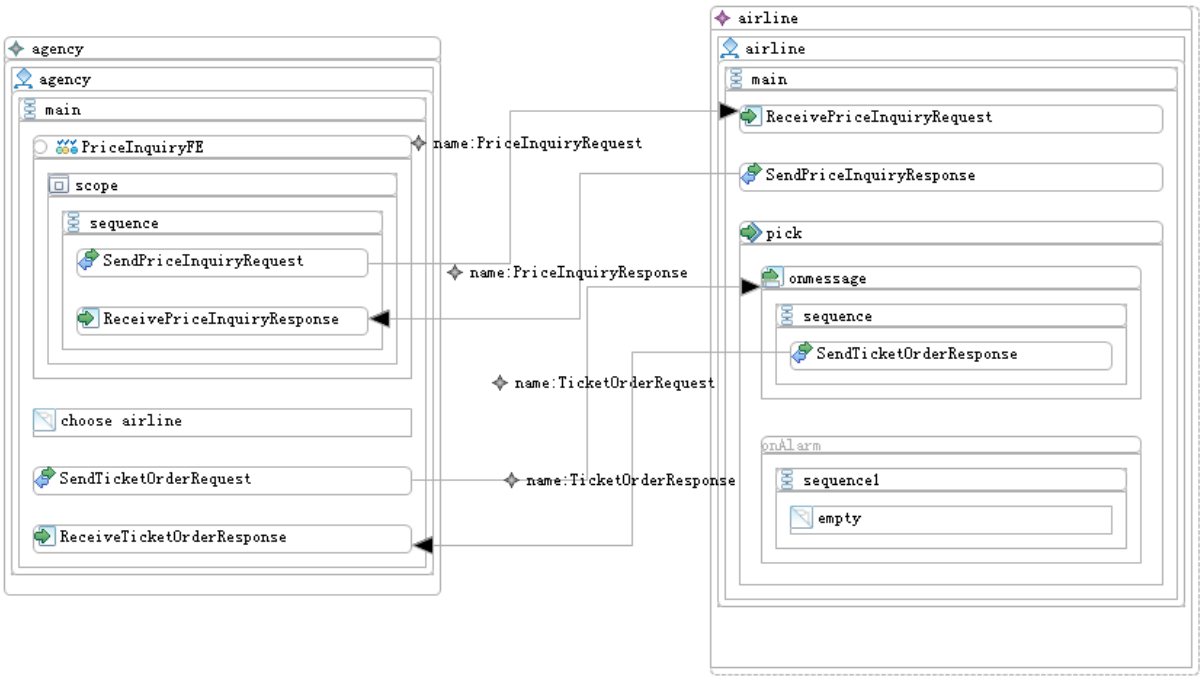


Figure 2.7: diagram travelExample generated by BPEL4Chor Designer

The diagram is modeled based on the choreography example in [DKLW07]. Participant *agency* sends *PriceInquiryRequest* message to a ParticipantSet *airline* using a `forEach` loop, after comparing the price that are returned from each airline, the participant *agency* sends a *TicketOrderRequest* message to the selected airline.

Each messageLink should be configured in the properties view of the component CMessageLink (Figure 2.8), this step is used to generate the messageLinks in topology file. After the messageLink is configured, the messageLinks in grounding file could be generated by configuring the *MessageLinks Grounding* in *Groundings* property in properties view of the entire choreography (Figure 2.9).

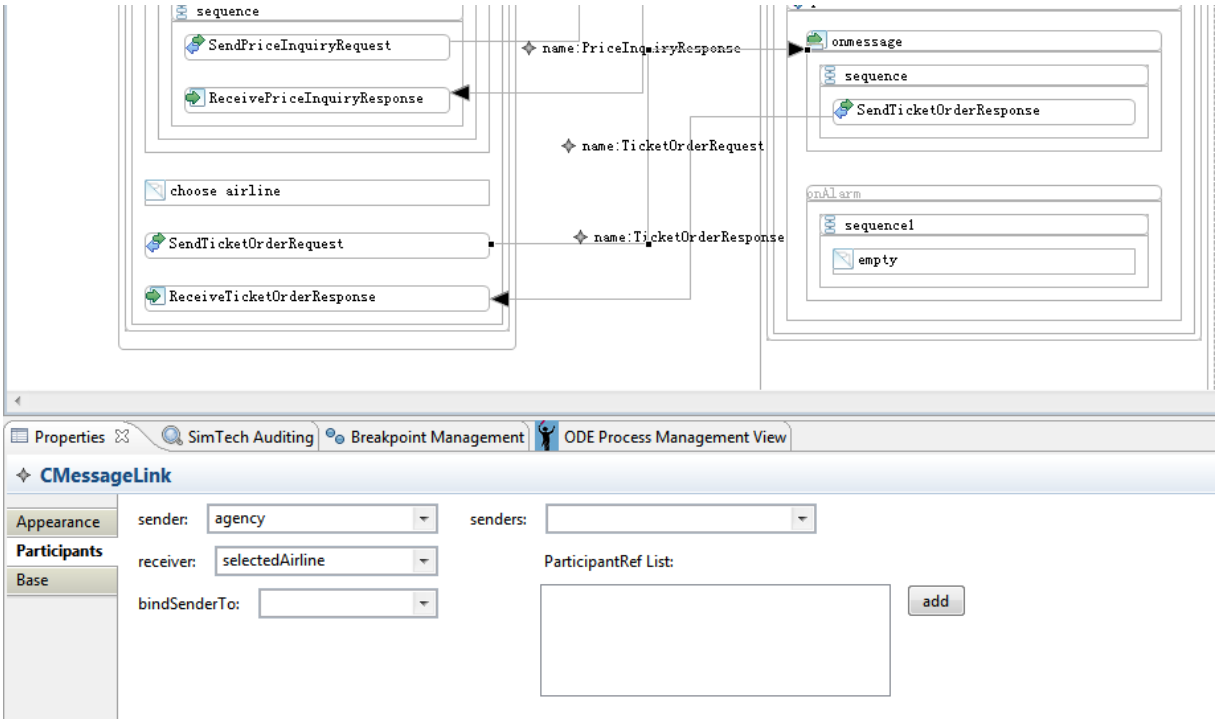


Figure 2.8: Properties view of component CMessageLink

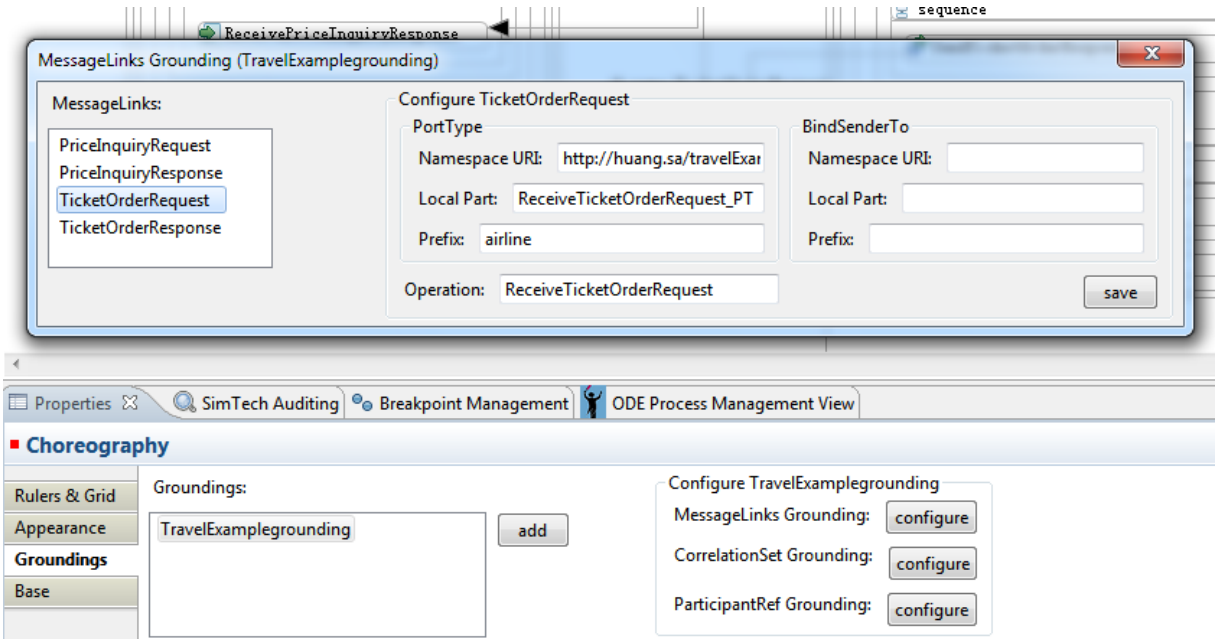


Figure 2.9: Property *Groundings* in properties view of choreography

The constructs `correlationSet` and `messageExchange` should be configured in the properties view of PBDs in which these constructs are declared. For example, the `correlationSet` *agency_TicketOrder_CS* declared in PBD *agency* is modeled by configuring the property *Correlations* in the properties view of PBD *agency* (Figure 2.10), this step is used to generate the construct `correlationSet` in PBD file, after that, the *Correlation Grounding* in property *Groundings* of the choreography should be configured (Figure 2.11).

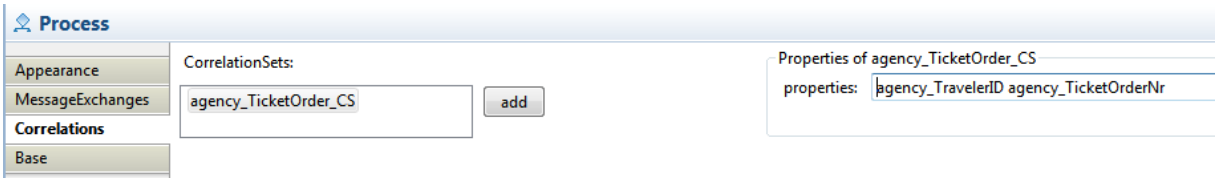


Figure 2.10: Property *Correlations* in properties view of PBD

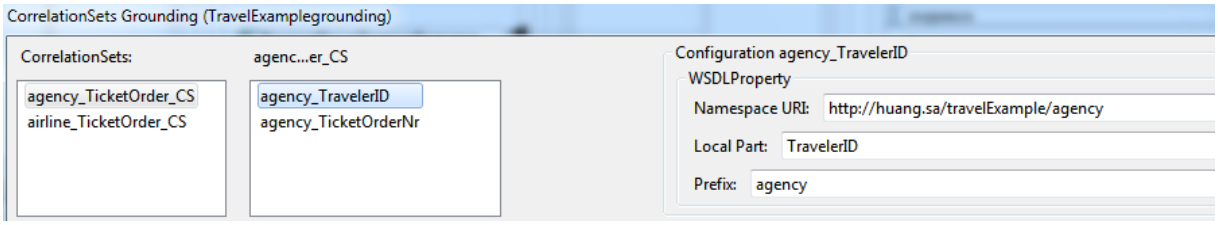


Figure 2.11: Configuring *CorrelationSet Grounding* in properties view of choreography

After all the groundings are configured, the BPEl4Chor can be generated. The result is shown as follows.

Listing 2.3.1: topology file travelExampleTopology.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<topology xmlns="urn:HPI_IAAS:choreography:schemas:choreography:topology:2006/12"
                  xmlns:agency="http://huang.sa/travelExample/agency"
                  xmlns:airline="http://huang.sa/travelExample/airline"
                  name="travelExampleTopology"
                  targetNamespace="http://huang.sa/travelExample">
  <participantTypes>
    <participantType name="agency_0_type" participantBehaviorDescription="agency:agency"/>
    <participantType name="airline_1_type" participantBehaviorDescription="airline:airline"/>
  </participantTypes>
  <participants>
    <participant name="agency" type="agency_0_type"/>
    <participantSet forEach="agency:ff71e4d4-ce5c-4346-a370-c97a3e803867"
                  name="airline"
                  type="airline_1_type">
      <participant forEach="agency:ff71e4d4-ce5c-4346-a370-c97a3e803867"
                  name="currentAirline"
                  scope="agency:06b17b89-802a-4ea7-88c3-a5516cf448c3"/>
      <participant name="selectedAirline"/>
    </participantSet>
  </participants>
  <messageLinks>
    <messageLink messageName="PriceInquiryRequest"
                  name="PriceInquiryRequest"
                  receiveActivity="8e846abc-0957-46fc-a898-1d4045aeae6e"
                  receiver="currentAirline"
                  sendActivity="8f0170a2-1e24-435a-b787-fd1dda88a228"
                  sender="agency"/>
    <messageLink bindSenderTo="currentAirline"
                  messageName="PriceInquiryResponse"
                  name="PriceInquiryResponse"
                  receiveActivity="f6850b41-312e-477e-a333-38d42926252d"
                  receiver="agency"
                  sendActivity="592877a1-03ba-44c0-9843-c19d7242dcc1"
                  senders="airline"/>
    <messageLink messageName="TicketOrderRequest"
                  name="TicketOrderRequest"
                  receiveActivity="4b833553-1b00-4fe0-b379-98940965bba8"
                  receiver="selectedAirline"
                  sendActivity="ea171520-a045-4489-86c4-739e127f4d34"
                  sender="agency"/>
    <messageLink bindSenderTo="selectedAirline"
                  messageName="TicketOrderResponse"
                  name="TicketOrderResponse"
                  receiveActivity="e910dc55-96ff-4e01-9a53-ba4946a279c9"
                  receiver="agency"
                  sendActivity="eb8abc2c-a379-4256-acc6-0263bd3ea185"
                  senders="airline"/>
```

```
  </messageLinks>
</topology>
```

Listing 2.3.2: grounding file travelExampleGrounding.xml

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<grounding xmlns="urn:HPI_IAAS:choreography:schemas:choreography:grounding:2006/12"
                   xmlns:agency="http://huang.sa/travelExample/agency"
                   xmlns:airline="http://huang.sa/travelExample/airline"
                   xmlns:top="http://huang.sa/travelExample"
                   topology="top:travelExampleTopology">
  <messageLinks>
    <messageLink name="PriceInquiryRequest"
                   operation="ReceivePriceInquiryRequest"
                   portType="airline:ReceivePriceInquiryRequest_PT"/>
    <messageLink name="PriceInquiryResponse"
                   operation="ReceivePriceInquiryResponse"
                   portType="agency:ReceivePriceInquiryResponse_PT"/>
    <messageLink name="TicketOrderRequest"
                   operation="ReceiveTicketOrderRequest"
                   portType="airline:ReceiveTicketOrderRequest_PT"/>
    <messageLink name="TicketOrderResponse"
                   operation="ReceiveTicketOrderResponse"
                   portType="agency:ReceiveTicketOrderResponse_PT"/>
  </messageLinks>
  <participantRefs>
    <participantRef WSDLproperty="airline:currentAirline" name="currentAirline"/>
    <participantRef WSDLproperty="airline:selectedAirline" name="selectedAirline"/>
  </participantRefs>
  <properties>
    <property WSDLproperty="agency:TravelerID" name="agency_TravelerID"/>
    <property WSDLproperty="agency:TicketOrderNr" name="agency_TicketOrderNr"/>
    <property WSDLproperty="airline:TravelerID" name="airline_TravelerID"/>
    <property WSDLproperty="airline:TicketOrderNr" name="airline_TicketOrderNr"/>
  </properties>
</grounding>
```

Listing 2.3.3: PBD agency.bpel

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
        1.0.xsd"
        abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12"
        name="agency"
        targetNamespace="http://huang.sa/travelExample/agency">
  <correlationSets>
    <correlationSet name="agency_TicketOrder_CS"
                   properties="agency_TravelerID agency_TicketOrderNr"/>
  </correlationSets>
  <sequence name="main" wsu:Id="f05399cc-0002-444e-a038-b60609fa61bd">
    <forEach name="PriceInquiryFE" parallel="yes" wsu:Id="ff71e4d4-ce5c-4346-a370-c97a3e803867">
      <scope name="scope" wsu:Id="06b17b89-802a-4ea7-88c3-a5516cf448c3">
        <sequence name="sequence" wsu:Id="6d59311a-24b5-4500-a8d1-d5f7c3169f99">
          <invoke name="SendPriceInquiryRequest"
                  wsu:Id="8f0170a2-1e24-435a-b787-fd1dda88a228"/>
          <receive name="ReceivePriceInquiryResponse"
                  wsu:Id="f6850b41-312e-477e-a333-38d42926252d"/>
        </sequence>
      </scope>
    </forEach>
    <opaqueActivity name="choose airline" wsu:Id="e236bcc7-9d65-4d1f-bfbf-0565139ef547"/>
    <invoke name="SendTicketOrderRequest"
           wsu:Id="ea171520-a045-4489-86c4-739e127f4d34">
      <correlations>
        <correlation initiate="yes" set="agency_TicketOrder_CS"/>
      </correlations>
    </invoke>
    <receive name="ReceiveTicketOrderResponse" wsu:Id="e910dc55-96ff-4e01-9a53-ba4946a279c9">
      <correlations>
        <correlation set="agency_TicketOrder_CS"/>
      </correlations>
    </receive>
  </sequence>
</process>
```

Listing 2.3.4 PBD airline.bpel

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
        xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
        1.0.xsd"
        abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12"
        name="airline"
        targetNamespace="http://huang.sa/travelExample/airline">
  <correlationSets>
    <correlationSet name="airline_TicketOrder_CS"
                    properties="airline_TravelerID airline_TicketOrderNr"/>
  </correlationSets>
  <sequence name="main" wsu:Id="87271be6-0eec-4b12-ae81-98f0f9480726">
    <receive createInstance="yes"
             name="ReceivePriceInquiryRequest"
             wsu:Id="8e846abc-0957-46fc-a898-1d4045aeae6e"/>
    <invoke name="SendPriceInquiryResponse"
            wsu:Id="592877a1-03ba-44c0-9843-c19d7242dcc1"/>
    <pick name="pick" wsu:Id="f4027552-7c16-4cba-b997-9c0b58c0843e">
      <onMessage name="onmessage" wsu:Id="4b833553-1b00-4fe0-b379-98940965bba8">
        <correlations>
          <correlation initiate="yes" set="airline_TicketOrder_CS"/>
        </correlations>
        <sequence name="sequence" wsu:Id="7932d66b-e175-4efb-ad28-800c5361a58c">
          <invoke name="SendTicketOrderResponse"
                  wsu:Id="eb8abc2c-a379-4256-acc6-0263bd3ea185">
            <correlations>
              <correlation set="airline_TicketOrder_CS"/>
            </correlations>
          </invoke>
        </sequence>
      </onMessage>
      <onAlarm>
        <for>"'Pt2M'"</for>
        <sequence name="sequence1" wsu:Id="1a9ab838-e048-4e04-85f3-afef9e92ebe5">
          <opaqueActivity name="empty" wsu:Id="5a23bf95-ab29-41fc-8ca7-0aa184cecc65"/>
        </sequence>
      </onAlarm>
    </pick>
  </sequence>
</process>
```

The transformation from BPEL4Chor to abstract BPEL is introduced in the following chapter.

3 Implementation

The implementation of the definitions and algorithms in [Rei07] is using JAXB, I will briefly introduce JAXB at first. Because the algorithms in [Rei07] are based on JDOM, so I have modified some algorithms so that it is possible to implement them, the modifications will be introduced in corresponding sections. The entire project can be seen under https://code.google.com/p/huang-sa/source/browse/Implementation. And the output BPEL files of the transformation can be seen at Appendix 6.2.

3.1 Introduction to JAXB

Java Architecture for XML Binding (JAXB) provides a fast and convenient way to bind XML schemas and Java [JAXB]. As Figure 3.1 demonstrates, the general steps of the binding process are described as follows:

First, the JAXB binding compiler take an XML schema as input and turn it into JAXB classes and the generated classes, course files, application code are compiled. Then, XML documents that instantiates the source XML schema can be unmarshalled by the JAXB binding Framework, a content tree of data objects that instantiate the generated JAXB classes are generated. The application can then modifiy the objects by using the methods that are also generated by the binding compiler. At last, the content tree of modified objects is marshalled into one or more XML output documents.

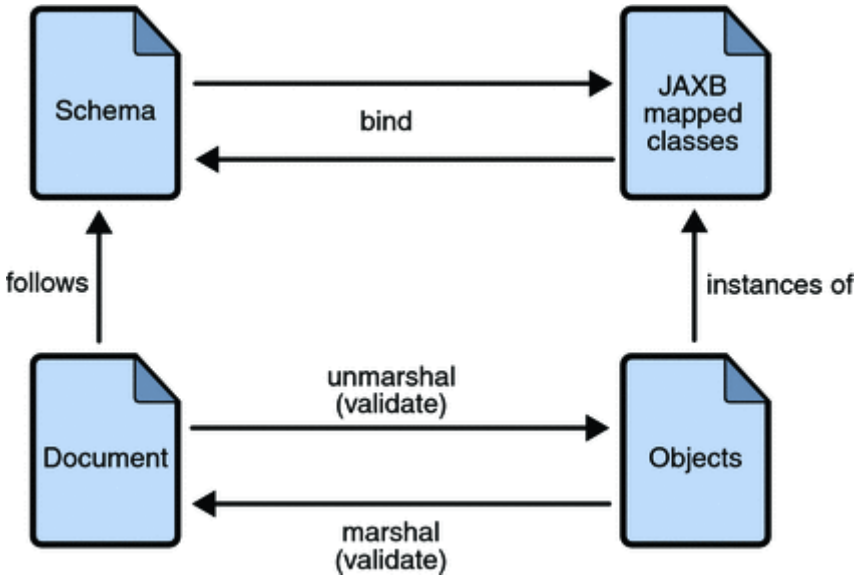


Figure 3.1: JAXB binding process [JAXB]

The detailed usage of JAXB will be explained in the following sections.

3.2 Introduction to the transformation process

The sequence diagram of the main function of the transformation is shown in Figure 3.2. All the data structures are implemented and stored by class Data, class Data is used to store and aquire the information that are retrieved from BPEL4Chor topology and grounding. The Definition 3.1 – 3.36 in [Rei07] are implemented in class Data using proper data structures. Class TopologyHander is used to fetch desired information from topology file, class GroundingHandler is used to fetch information from grounding file. Class PBDHandler is

used to transform the PBD into abstract BPEL file. Class WSDLHandler is used to generate corresponding wsdl file. These classes will be introduced in following sections.

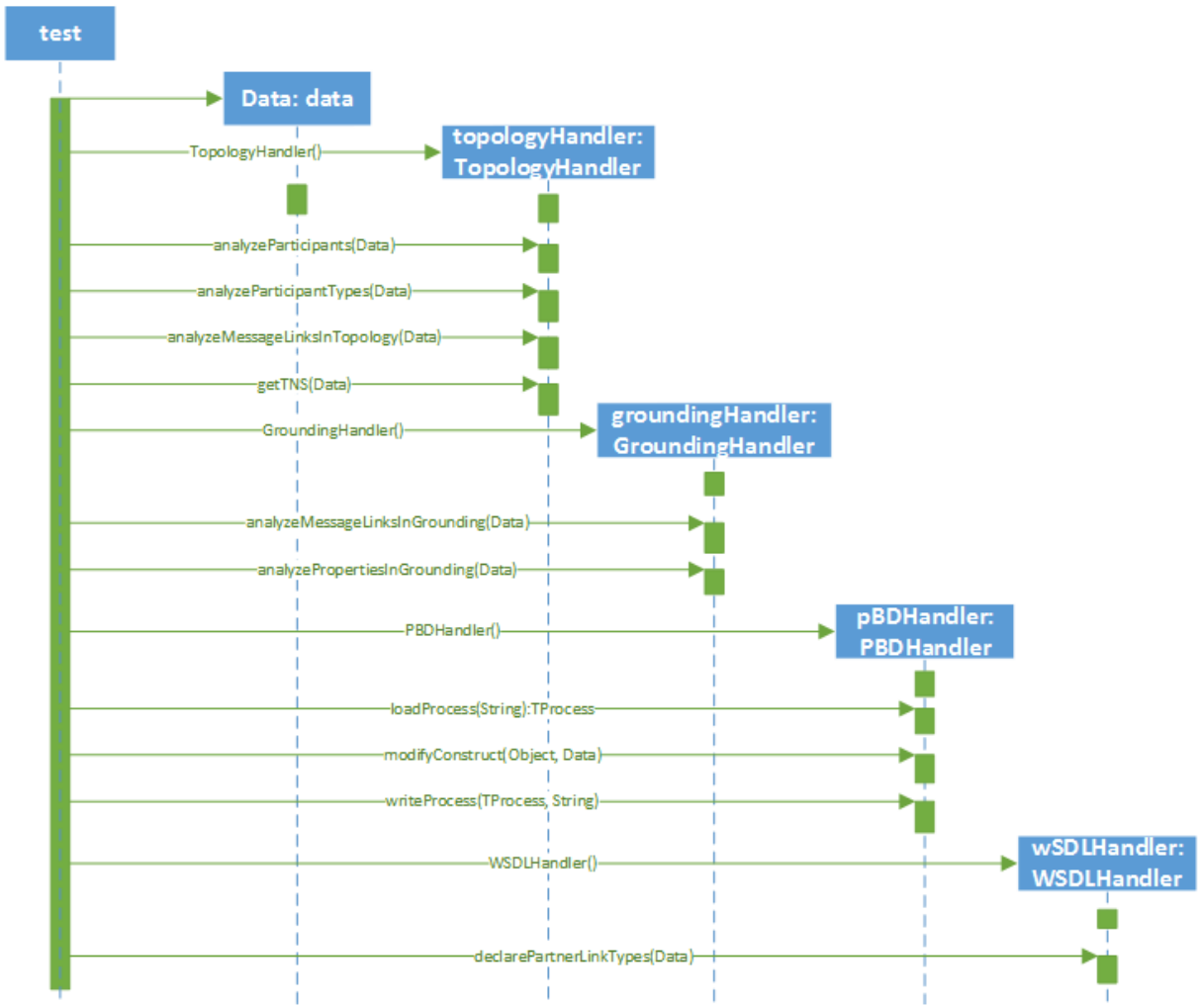


Figure 3.2: sequence diagram

3.2.1 Class Data

There are mainly two kinds of data structure are used to implement the definitions in [Rei07], they are HashSets and HashMaps. HashSets are used to implement the Definitions that define sets of data, like Definition 3.4(set of participant types), Definition 3.5(set of processes) and so on. HashMaps are used to implement the Definitions that define functions, like Definition 3.9(the function assigning a participant type to each participant reference) and so on, during the functions in class TopologyHandler and GroundingHandler are being called, data are assigned to these HashMaps, and these data are fetched to be used to turn PBD into BPEL file, and to generate wsdl file, when the functions in PBDHandler and WSDLHandler are being called (Listing 3.2.1.1 – 3.2.1.3).

Listing 3.2.1.1: HashMap *paType_PaMap* and *process_PaTypeMap* in class Data

```
private HashMap<String, String> paType_PaMap = new HashMap<String, String>();
```

Listing 3.2.1.2: function setPaType_PaMap(String pa, String paType) in class Data

```
public void setPaType_PaMap(String pa, String paType) {
```

```
  paType_PaMap.put(pa, paType);
}
```

Listing 3.2.1.3: function *getPaType_PaMap()* in class Data

```
public HashMap<String, String> getPaType_PaMap() {
  return paType_PaMap;
}
```

3.2.2 Class TopologyHandler

The class *TopologyHandler* is designed based on algorithm 3.1 in [Rei07].

The class diagram is shown as Figure 3.3. The field *jaxbContext* that instantiates class *JAXBContext* is the entry point to the JAXB API, it owns the method *unmarshall()* and *marshall()* and so on[JAVA6], the field *topology* is the object that instantiates the class *TTopology* that is generated by the method *unmarshall()*. The constructor method *TopologyHandler()* calls method *loadTopology(String filePath)* in which method *unmarshall()* is called to unmarshall the input topology file into objects.

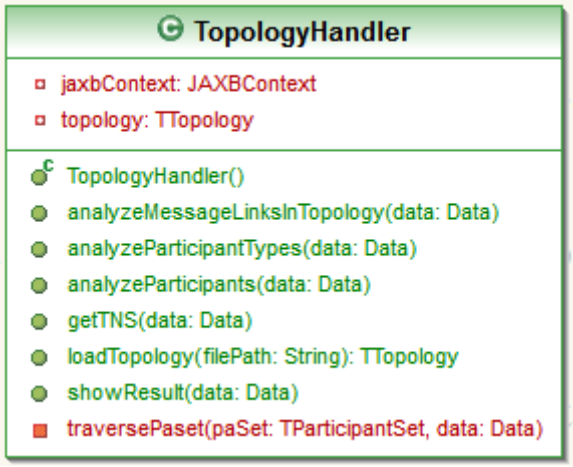


Figure 3.3: class diagram of class *TopologyHandler*

The method *analyzeMessageLinksInTopology(Data data)* implements the algorithm 3.1. In [Rei07] it is not discussed how to implement some of the functions like $type_{Pa}$: $Pa \longrightarrow PaType$ in Definition 3.9, so I implement these functions using methods *analyzeParticipantTypes(Data data)*, *analyzeParticipants(Data data)* (Listing 3.2.2.1) and *traversePaset(TParticipantSet paSet, Data data)*.

Listing 3.2.2.1: method *analyzeParticipants(Data data)* in class TopologyHandler

```
01 public void analyzeParticipants(Data data){
02   TParticipants tpas = topology.getParticipants();
03     for(Object paOrPaset : tpas.getParticipantOrParticipantSet()){
04     if(paOrPaset instanceof TParticipant){
05       data.setParticipantSet(((TParticipant)paOrPaset).getName());
06       data.setPaType_PaMap(((TParticipant)paOrPaset).getName(),
  ((TParticipant)paOrPaset).getType());
07       if(((TParticipant)paOrPaset).getForEach() != null){
08         data.setScopeSet(((TParticipant)paOrPaset).getForEach());
09         }
10       if(((TParticipant)paOrPaset).getScope() != null){
11         data.setScopeSet(((TParticipant)paOrPaset).getScope());
12       }
13     }else{
14       data.setPaType_PaMap(((TParticipantSet)paOrPaset).getName(),
  ((TParticipantSet)paOrPaset).getType());
15       traversePaset(((TParticipantSet)paOrPaset), data);
16     }
```

```
17    }
18 }
```

The method *getParticipants()* is one of the methods that generated by the JAXB binding compiler (code 3.2.2.1 line 02), it returns an object that instantiates the class *TParicipants*, which has a field *List<Object> participantOrParticipantSet,* this List contains the objects participant or participantSet that instantiate the class *TParicipant* and *TParticipantSet* that are generated by the JAXB binding compiler.

Listing 3.2.2.2: method *getParticipants()* in class *TTopology*:

```
public TParticipants getParticipants() {
  return participants;
}
```

The for loop begins from code 3.2.2.1 line 03 traverses the *List<Object> participantOrParticipantSet*, for the objects that are instance of *TParticipant*, the pair of value *((TParticipant)paOrPaset).getName(), ((TParticipant)paOrPaset).getType()* are added to the HashMap *paType_PaMap(String pa, String paType)*. If the objects are instance of *TParticipantSet*, the pair of value *((TParticipantSet)paOrPaset).getName(), ((TParticipantSet)paOrPaset).getType()* are added to the HashMap *paType_PaMap(String pa, String paType)*.

The method *traversePaset(TParticipantSet paSet, Data data)* is used to traverse the possibly nested participantSet.

The method *getTNS(Data data)* is used to get the attribute *targetNamespace* of the topology file.

3.2.3 Class Comm

The class *Comm* is designed on the basis of Definition 3.27(the relation *Comm*). It is used to store the interactions between the participant processes. The relation *Comm* is defined in form ((A, A_pt), (b, b_pt)), so the class *Comm* has four fields, as Figure 3.4 illustrates.

The Definition 3.27 describes only two kinds of interactions between participants, one-way message and asynchronous interaction using WS-Addressing. So the class *Comm* can only represent these two kinds of interactions (see also section 4.1).

The class *Comm* will be used in class *GroundingHandler* and *WSDLHandler*, the desired information will be fetched from grounding file and be used to generate corresponding element *comm*, and the value pairs of functions defined in Definition 3.28 (the function assigning a pair of partner link declarations to each element of *Comm*) and Definition 3.29 (the function assigning a partner link type to each element of *Comm*) will be updated. These functions will be used later to generate `partnerlink`s and `partnerlinkType`s.

Figure 3.4: class diagram of class *Comm*

3.2.4 Class GroundingHandler

The class *TopologyHandler* is designed based on algorithms 3.2 – 3.4 and 3.11 in [Rei07].

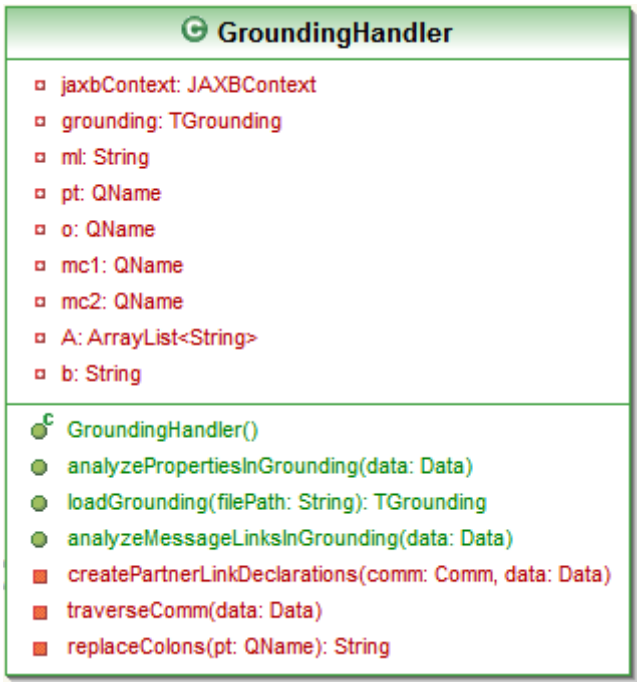


Figure 3.5: class diagram of class *GrondingHandler*

As Figure 3.5 demonstrates, the class *GroundingHandler* also contains field *jaxbContext: JAXBContext*, which is used to unmarshall the input grounding file into object *grounding: TGrounding*. In method *analyzeMessageLinksInGrounding(Data data)* (Listing 3.2.3.1), we traverse the result of *grounding.getMessageLinks()*, which is a *List<TMessageLink>*, from each object *tml: TMessageLink*, we fetch the desired information and store the information in object *data*, and then the method *traverseComm(Data data)* (Listing 3.2.3.2) is called, to traverse the elements in relation *Comm*, and compare them to the information we fetched

from current messageLink, depends on the result of the compare, a new element *comm* is declared or not.

Listing 3.2.4.1: Method *analyzeMessageLinksInGrounding(Data data)* (part) in class *GroundingHandler*

```
public void analyzeMessageLinksInGrounding(Data data){
  TMessageLinks tmls = grounding.getMessageLinks();
  for(TMessageLink tml : tmls.getMessageLink()){
    ml = tml.getName().getLocalPart();
    pt = tml.getPortType();
      ......
  }
  traverseComm(data)
}
```

Listing 3.2.4.2: Method *traverseComm(Data data)* (part) in class *GroundingHandler*

```
01 private void traverseComm(Data data) {
02   HashSet<Comm> currentCommSet = new HashSet<Comm>();
03   currentCommSet.addAll(data.getCommSet());
04   Iterator<Comm> i = currentCommSet.iterator();
05   int condition = 0;
06   Comm selectedComm = new Comm();
07
08   do {
09     if(currentCommSet.isEmpty()){
10     System.out.println(" CommSet is empty");
11     break;
12     }
13     Comm currentComm = i.next();
14     if((currentComm.getA().equals(A))
15     && ((data.getPortTypeSet().contains(currentComm.getA_pt())) ||
  (currentComm.getA_pt()==null))
16     && (currentComm.getB().equals(b))
17     && ((currentComm.getB_pt().equals(pt)))){
18
19       condition = 1;
20       selectedComm = currentComm;
21       break;
22     } else if(......){
23       ......
24   } while(i.hasNext());
25
26   switch(condition){
27   case 1:
28     data.setPartnerLink_MCMap(mc1, data.getPartnerLinks_CommMap().get(selectedComm).get(0));
29     data.setPartnerLink_MCMap(mc2, data.getPartnerLinks_CommMap().get(selectedComm).get(1));
30     break;
31     ......
32   }
33 }
```

At the beginning, there is no element in relation *Comm*, so I used a do-while loop to check which condition is met(Listing 3.2.4.2 line 08 - 24), if there is no element in relation *Comm*, the default case in switch statement(line 26 - 32) will be executed, a new element *comm* will be declared.

During the entire do-while loop , new element *comm* might be added to *commSet* in each loop, so a *HashSet<Comm> currentCommSet* (line 02)which contains all elements in original *commSet* is traversed, and new element will be added to original *commSet*.

Method *analyzePropertiesInGrounding(Data data)* is desigend based on algorithm 3.11, it traverses the result of *grounding.getProterties()*, which is a *List<TProperty>*, fetches the disired information and stores the information into object *data* (see also section 4.2).

3.2.5 Class PBDHandler

The class PBDHandler is designed based on algorithms 3.6 – 3.10, 3.12 – 3.17. Figure 3.5 shows the class diagramm of class PBDHandler.

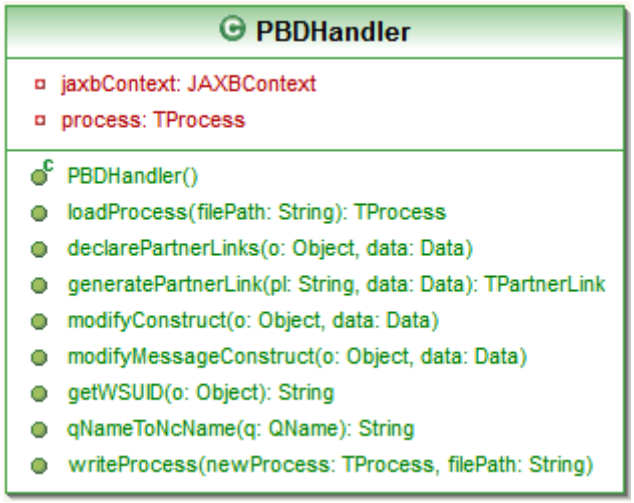


Figure 3.6: class diagramm of class PBDHandler

Method *declarePartnerLinks(Object o, Data data)* (Listing 3.2.5.1) is based on algorithm 3.6, In BPEL file, `partnerLink`s could be declared under the tag `<process>` or `<scope>`, the information for which `partnerLink`s should be declared in which `process` or `scope`, is stored in *HashMap<QName, HashSet<String>> partnerLinks_scopeMap*, whose content is updated during the method *createPartnerLinkDeclaration(Comm comm, Data data)* in class *GroundingHandler* is being called.

Listing 3.2.5.1: Method *declarePartnerLinks(Object o, Data data)* (part1) in class *PBDHandler*

```
01 public void declarePartnerLinks(Object o, Data data){
02   if(o instanceof TProcess){
03     Iterator<QName> iter = data.getPartnerLinks_scopeMap().keySet().iterator();
04     while(iter.hasNext()){
05     QName key = iter.next();
06     if(key.getLocalPart().equals(((TProcess) o).getName())){
07       HashSet<String> sc = data.getPartnerLinks_scopeMap().get(key);
08       TPartnerLinks tpls = new TPartnerLinks();
09       for(Iterator<String> iter_1 = sc.iterator(); iter_1.hasNext();){
10         String pl = iter_1.next();
11         tpls.getPartnerLink().add(generatePartnerLink(pl, data));
12       }
13     ((TProcess) o).setPartnerLinks(tpls);
14     }
15   }
```

We take the situation that the input object is an instance of TProcess as example. First we traverse the *HashMap<QName, HashSet<String>> partnerLinks_scopeMap*(line 03 – 12), the QName *key* identifies a process, if the process' name equals the input process' name, we need to declare the `partnerlink`s which is stored as the value of this key in *partnerLinks_scopeMap*, we traverse this value, which is a *HashSet<String>*(line 09 - 12), fetch each String out of this HashSet and generates an object *TPartnerLink*(line 11), an add this object into object *tpls: TPartnerLinks*, at last we add the object *tpls* to the process.

The situation for *TScope* is almost the same but the identifier of scope is different, as in BPEL4Chor, we use *wsu:id* as the identifier of a scope, the attribute *wsu:id* can be fetched by the method *((TScope) o).getOtherAttributes()* (Listing 3.2.5.2).

Listing 3.2.5.2: Method *declarePartnerLinks(Object o, Data data)* (part2) in class *PBDHandler*

```
......
 if(o instanceof TScope){
   Iterator<QName> iter = ((TScope) o).getOtherAttributes().keySet().iterator();
   while(iter.hasNext()){
     QName key = iter.next();
     String value = ((TScope) o).getOtherAttributes().get(key);
     for(Iterator<QName> iter_1 = data.getScopeSet().iterator(); iter_1.hasNext();){
     ......
```

Method *modifyConstruct(Object o, Data data)* is based on algorithms 3.7, 3.9, and 3.12 – 3.16.

In [Rei07] it is not discussed how to deal with construct `<sequence>`, this might be a drawback, because in BPEL, the activities that actually realize the business logic are generally contained by a `<sequence>`, and there is usually a `<sequence>` in each `<scope>` of a `<forEach>`. Although it is syntactically correct to directly declare activities like `invoke` or `assign` under the tag `<process>`, as in schema of BPEL file defined (Listing 3.2.5.3), therefore in class *TProcess* (Listing 3.2.5.4) these activities are also declared as fields, but as in section 2.3 discussed, the PBD files that are generated by BPEL4Chor Designer are automatically filled with `sequence`s, so in this implementation, some modification have been done to the algorithms in [Rei07].

Listing 3.2.5.3: Schema for Abstract Process Common Base for WS-BPEL 2.0 (part)

```
<xsd:complexType name="tProcess">
  <xsd:complexContent>
    <xsd:extension base="tExtensibleElements">
      <xsd:sequence>
        <xsd:element ref="extensions" minOccurs="0"/>
        <xsd:element ref="import" minOccurs="0" maxOccurs="unbounded"/>
        <xsd:element ref="partnerLinks" minOccurs="0"/>
        <xsd:element ref="messageExchanges" minOccurs="0"/>
        <xsd:element ref="variables" minOccurs="0"/>
        <xsd:element ref="correlationSets" minOccurs="0"/>
        <xsd:element ref="faultHandlers" minOccurs="0"/>
        <xsd:element ref="eventHandlers" minOccurs="0"/>
        <xsd:group ref="activity" minOccurs="0"/>
      </xsd:sequence>
      ......
```

Listing 3.2.5.4: Class *TProcess* (part)

```
public class TProcess
  extends TExtensibleElements {
    protected TPartnerLinks partnerLinks;
    protected TCorrelationSets correlationSets;
    protected TAssign assign;
    protected TForEach forEach;
    protected TInvoke invoke;
    ......
```

My *modifyConstruct(Object o, Data data)* is designed as a recursive function, this method only need to be called once for each BPD file, it takes an *object: TProcess* as input parameter in the main function, each construct that is declared under this process corresponds to an object that is generated by the JAXB compiler, for each of these objects the method *modifyConstruct(Object o, Data data)* should be recursively called. Consider that there are about 30 constructs could be declared under `<process>`, I only implement part of them based on [Son13] (see also section 2.3).

Listing 3.2.5.5: Method *modifyConstruct(Object o, Data data)* (part1) in class *PBDHandler*

```
public void modifyConstruct(Object o, Data data){
  if(o instanceof TProcess){
    if(((TProcess) o).getCorrelationSets() != null){
      for(TCorrelationSet tcorr:((TProcess) o).getCorrelationSets().getCorrelationSet()){
        modifyConstruct(tcorr, data);
      }
    }
    declarePartnerLinks(o, data);
    TSequence tseq = ((TProcess) o).getSequence();
    modifyConstruct(tseq, data);
  }
```

Listing 3.2.5.5 shows what happens for the situation that the method takes an *object: TProcess* as input, we check if there is construct `correlationSets` declared in this `process`, if there is, each `correlationSet` under this `correlationSets` will be modified (Listing 3.2.5.6). The method *declarePartnerLinks(Object o, Data data)* is used to declare corresponding `partnerLink`s under this `process`. And then the `sequence` contained in this `process` is modified, for the purpose of simplification, I only considered the situation that there is only one `sequence` under a `process`.

Listing 3.2.5.6: Method *modifyConstruct(Object o, Data data)* (part2) in class *PBDHandler*

```
if(o instanceof TCorrelationSet){
  ArrayList<String> properties = new ArrayList<String>();
  for(String prop:((TCorrelationSet) o).getProperties()){
    String prefix = data.nsprefix_Property(data.property_CorrPropName(prop));
    String property = prefix.concat(":").concat(prop);
    properties.add(property);
  }
  ((TCorrelationSet) o).getProperties().clear();
  ((TCorrelationSet) o).getProperties().addAll(properties);
}
```

Listing 3.2.5.6 shows what happens when the method takes an *object: TCorrelationSet* as input, this part is based on algorithm 3.13.

Listing 3.2.5.7 Method *modifyConstruct(Object o, Data data)* (part3) in class *PBDHandler*

```
if(o instanceof TSequence){
  for(Object tact:((TSequence) o).getActivity()){
    if(tact instanceof TInvoke){
      modifyMessageConstruct(tact, data);
    }
      ......
    if(tact instanceof TForEach){
      modifyConstruct(tact, data);
    }
      ......
  }
  ((TSequence) o).getOtherAttributes().clear();
}
```

Listing 3.2.5.7 shows what happens when the method takes an *object: TSequence* as input. For the situation that the object corresponds an activity that is used to communicate with other participant, the corresponding construct is modified by the method *modifyMessageConstruct(Object o, Data data)*, which is based on algorithm 3.10. For other objects, method *modifyConstruct(Object o, Data data)* is recursively called. The last statement is used to clear the attribute *wsu:id*.

The way that the author of [Rei07] used to modify the construct `<forEach>` is also not correct. Based on [AcOS09], [Kai11], and [BPEL 2.0], I will introduce my program as follows.

The endpoint reference of each participant reference is stored in an array, which is declared in BPEL file as Listing 3.2.5.8 shows:

Listing 3.2.5.8: Variable *ArrayOfEPRs*

```
<bpel:variables>
   <bpel:variable name="ArrayOfEPRs" element="messageType:ArrayOfEPRs"/>
</bpel:variables>
```

The element *"messageType:ArrayOfEPRs"* is defined as follows:

Listing 3.2.5.9: Element *ArrayOfEPRs*

```
<element name="ArrayOfEPRs">
  <complexType>
    <sequence>
      <element name="ArrayOfEPRs"
              type="wsa:EndpointReferenceType"
              minOccurs="1"
              maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
```

The *wsa:EndpointReference* is defined in [WS-Addressing 1.0], it conveys the information needed to address a Web service endpoint, which in this case is provided by a BPEL process. To initiate the array *ArrayOfEPRs*, all the endpoint references of the processes need to be assigned to the elements in the *ArrayOfEPRs*, the initiation could be done as Listing 3.2.5.10.

Listing 3.2.5.10: Initiation of array ArrayOfEPRs

```
<bpel:assign name="Assign">
  <bpel:copy>
    <bpel:from>
      <bpel:literal xml:space="preserve">
         <sref:service-ref xmlns:sref="http://docs.oasis-open.org/wsbpel/2.0/serviceref">
          <EndpointReference xmlns="http://www.w3.org/2005/08/addressing">
            <Address>http://exampleService</Address>
          </EndpointReference>
         </sref:service-ref>
      </bpel:literal>
    </bpel:from>
    <bpel:to variable="ArrayOfEPRs">
      <bpel:query><![CDATA[messageType:ArrayOfEPRs[1]]]></bpel:query>
    </bpel:to>
  </bpel:copy>
</bpel:assgn>
```

As in [AcOS09] introduced, the array in BPEL is indexed from 1, not 0.

So for the participantSet declared in topology file showed in Listing 3.2.5.11, the construct forEach declared in PBD file showed in Listing 3.2.5.12 will be modified as Listing 3.2.5.13.

Listing 3.2.5.11: `participantSet` *airline* in topology file travelExampleTopology

```
<participantSet forEach="agency:ff71e4d4-ce5c-4346-a370-c97a3e803867"
                name="airline"
                type="airline_1_type">
<participant forEach="agency:ff71e4d4-ce5c-4346-a370-c97a3e803867"
             name="currentAirline"
             scope="agency:06b17b89-802a-4ea7-88c3-a5516cf448c3"/>
  <participant name="selectedAirline"/>
</participantSet>
```

Listing 3.2.5.12: construct `forEach` in PBD file agency

```
<forEach name="PriceInquiryFE"
         parallel="yes"
         wsu:Id="ff71e4d4-ce5c-4346-a370-c97a3e803867">
  <scope name="scope" wsu:Id="06b17b89-802a-4ea7-88c3-a5516cf448c3">
    <sequence name="sequence" wsu:Id="6d59311a-24b5-4500-a8d1-d5f7c3169f99">
        <invoke name="SendPriceInquiryRequest"
                wsu:Id="8f0170a2-1e24-435a-b787-fd1dda88a228"/>
       <receive name="ReceivePriceInquiryResponse"
                wsu:Id="f6850b41-312e-477e-a333-38d42926252d"/>
    </sequence>
  </scope>
</forEach>
```

Listing 3.2.5.13: transformed construct forEach (part)

```
<forEach counterName="i" parallel="yes" name="PriceInquiryFE">
  <startCounterValue>1</startCounterValue>
  <finalCounterValue>"size of arrayOfEPRs"</finalCounterValue>
  <scope name="scope">
    <partnerLinks>
      <partnerLink name="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"
        partnerLinkType="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-plt"
        myRole="agency"
        partnerRole="currentAirline"/>
    </partnerLinks>
    <sequence name="sequence">
      <assign>
        <copy>
          <from variable="arrayOfEPRs[i]"/>
          <to partnerLink="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"/>
        </copy>
      </assign>
      <invoke partnerLink="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"
              portType="airline:ReceivePriceInquiryRequest_PT"
              operation="ReceivePriceInquiryRequest"
              name="SendPriceInquiryRequest"/>
            ……
    </sequence>
  </scope>
</forEach>
```

As Listing 3.2.5.13 shows, in each loop of `forEach` *PriceInquiryFE*, an endpoint reference stored in arrayOfEPRs, which indexed by counter *i*, is copied to the `partnerLink` that indicates the communication between the other participant *airline* and the process *agency* in which the `forEach` is declared.

Listing 3.2.5.14 shows what has been done for completing the modification.

Listing 3.2.5.14: method *modifyConstruct(Object o, Data data)* (part4) in class *PBDHandler*

```
01 if(o instanceof TForEach){
02   TForEach tfe = (TForEach)o;
03   String counterName = "i";
04   tfe.setCounterName(counterName);
05   ......
06   String startCounterValue = "1";
07   startCounter.getContent().add(startCounterValue);
08   ......
09   String finalCounterValue = "size of arrayOfEPRs[".concat(counterName).concat("]");
10   finalCounter.getContent().add(finalCounterValue);
11   tfe.setStartCounterValue(startCounter);
12   tfe.setFinalCounterValue(finalCounter);
13
14   TScope tsc = tfe.getScope();
15   TAssign tassign = new TAssign();
16   TCopy tcopy = new TCopy();
17   TFrom tfrom = new TFrom();
18   tfrom.setVariable("arrayOfEPRs[".concat(counterName).concat("]"));
19   tcopy.setFrom(tfrom);
20   TTo tto = new TTo();
```

```
21   /**
22    *  set the proper partnerLink
23    */
24   Iterator<QName> iter = tsc.getOtherAttributes().keySet().iterator();
25   while(iter.hasNext()){
26     QName key = iter.next();
27     String value = tsc.getOtherAttributes().get(key);
28     for(Iterator<QName> iter_1 = data.getScopeSet().iterator(); iter_1.hasNext();){
29       QName sc = iter_1.next();
30       if(value.equals(sc.getLocalPart())){
31         HashSet<String> pls = data.getPartnerLinks_scopeMap().get(sc);
32         for(Iterator<String> iter_2 = pls.iterator(); iter_2.hasNext();){
33         String pl = iter_2.next();
34         tto.setPartnerLink(pl);
35         }
36       }
37     }
38   }
39   tcopy.setTo(tto);
40   tassign.getCopyOrExtensionAssignOperation().add(tcopy);
41   /**
42    * add assign to activity list
43    */
44   TSequence tseq = tsc.getSequence();
45   for(int i = (tseq.getActivity().size() -1); i>=0; i--){
46     tseq.getActivity().add(i+1,tseq.getActivity().get(i));
47     tseq.getActivity().remove(i);
48   }
49   tseq.getActivity().add(0, tassign);
50   modifyConstruct(tsc, data);
51   ((TForEach) o).getOtherAttributes().clear();
52 }
```

Statements Line 03 – 12 in Listing 3.2.5.14 set the *counterName*, *startCounterValue* and *finalCounterValue*. Statements Line 14 – 51 modifies the `scope` declared in this `forEach`. Line 24 – 40 shows how to set the proper `partnerLink` in the construct `to`, like the method *declarePartnerLinks(Object o, Data data)* (Listing 3.2.5.1), information stored in *HashMap<QName, HashSet<String>> partnerLinks_scopeMap* is used to decide which `partnerLink` should be set to the construct `to`. Line 45 – 50 shows how to add the newly declared and modified object *tassign: TAssign* to the object *tseq: TSequence*, because the construct `assign` should be declared before all the activities that are already existed in the `sequence`, so all the objects in the List<Object>, which is the result of *tseq.getActivity()* will be moved to the place where its original index plus 1. So that the object *tassign* could be added to the List with index 0.

3.2.6 Class WSDLHandler

The class WSDLHandler is designed based on algorithm 3.19.

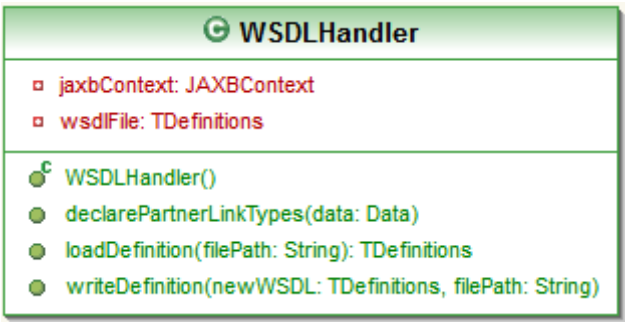


Figure 3.7: class diagram of class WSDLHandler

In the schema file of WSDL there is no definition for the partnerLinkType, so I made the following changes to the schema file of WSDL.

First add attribute `xmlns:plnk="http://docs.oasis-open.org/wsbpel/2.0/plnktype"` to the root element schema, and import the schema file of partnerLinkType (Listing 3.2.6.1) then add element *partnerLinkType* to the group *anyTopLevelOptionalElement* (Listing 3.2.6.2), then the class *partnerLinkType* can be contained by the field *List<Object> anyTopLevelOptionalElement* of class *TDefinitions* which is generated by the JAXB compiler.

Listing 3.2.6.1: tag *import* in modified schema file of WSDL (part)

```
<xs:import namespace="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
           schemaLocation="ws-bpel_plnktype.xsd"/>
```

Listing 3.2.6.2: tag *group* in modified schema file of WSDL (part)

```
<xs:group name="anyTopLevelOptionalElement" >
  <xs:element name="partnerLinkType" type="plnk:tPartnerLinkType" />
  ……
```

The generated wsdl file is showed in Listing 3.2.6.3.

Listing 3.2.6.3: generated wsdl file test_LT.wsdl

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
             xmlns:ns2="http://docs.oasis-open.org/wsbpel/2.0/plnktype"
             targetNamespace="http://huang.sa/travelExample">
    <partnerLinkType
         name="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-plt">
        <ns2:role xmlns:airline="http://huang.sa/travelExample/airline"
                  name="selectedAirline" portType="airline:ReceiveTicketOrderRequest_PT"/>
        <ns2:role xmlns:agency="http://huang.sa/travelExample/agency"
                  name="agency" portType="agency:ReceiveTicketOrderResponse_PT"/>
    </partnerLinkType>
    <partnerLinkType
        name="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-plt">
        <ns2:role xmlns:airline="http://huang.sa/travelExample/airline"
                  name="currentAirline" portType="airline:ReceivePriceInquiryRequest_PT"/>
        <ns2:role xmlns:agency="http://huang.sa/travelExample/agency"
                  name="agency" portType="agency:ReceivePriceInquiryResponse_PT"/>
    </partnerLinkType>
</definitions>
```

4 Evaluation

4.1 The drawback of relation Comm

According to [Rei07]:

Definition 3.27 (the relation Comm). The relation Comm $\subseteq (2^{Pa} \times (PT \cup \{ \perp \})) \times (Pa \times PT)$ *is defined as the relation that assigns a subset of participant references and another participant reference to a pair of port types which they use to communicate. Let ((A, c), (b, d)) be an element of the relation Comm. Then, the participant references contained in A communicate with the participant reference b while all participant references contained in A are realized by the port type c, and the participant reference b is realized by the port type d. If* $c = \perp$*, the communication will be one-way. That means the participant references contained in A send something to b, but not vice versa. If* $c \neq \perp$*, the communication will be request-response. That means the participant references contained in A send something to b and vice versa.*

For some reason that the author didn't mention, he left out the synchronous interaction. So the *Comm set* only contains two kinds of communication, one-way communication and asynchronous communication. And for asynchronous communication:

In Figure3.5 we see a request/response communication between the participant references a and b. ..... we need to create ..... one partner link type and two partner link declarations. But this time each of them gets both roles specified.

This means that the author of [Rei07] used the WS-Addressing method. This might not be the perfect choice for interaction in the form A > B > C > A.

Consider the conditions in *traverseComm()*:

From the data that can be fetched from both grounding and topology file, for each messageLink, there are:

A (sender(s)), b (receiver), pt (portType that *realizes* the communication, that is, it provides the `operation` that receive the message), o (`operation` that receives the message), sendActivity, receiveActivity, messageName.

While traversing the Comm set, if

(1): $\exists comm \in Comm: (comm = ((A, c), (b, pt)) \wedge c \in PT \cup \{\perp\})$*:*

This means that we have already had a message link specifying the same direction of communication of ml, and in which the participant reference b has been realized by the same port type.

Let this comm be a one-way communication between A and b, in which A sends Message1 through `operation` o1 (provided by `portType` *pt*) to b. (Figure 4.1.1)

Now the messageLink indicates that Message2 is sent from A to b through the same `portType` pt, i assume that there is another `operation` o2 in pt. The assumption is made under the consideration as follows:

A one-way `operation` can only receive message of the same type.

Example 4.1.1 `<portType>` *pt* in b's WSDL file:

```
<portType name="pt"
      <operation name="o1">
            input message="prefixOfReceiver:requestMessage"/>
      </operation>
</portType>
```

Example 4.1.2 `<message>` *requestMessage* in b's WSDL file:

```
<message name="requestMessage">
      <part name="requestMessage"
            element="prefixOfReceiver:requestMessage"/>
</message>
```

It seems meaningless to me that two messageLinks define messages of the same type using the same `operation`, so i assume that there should be another `operation`.

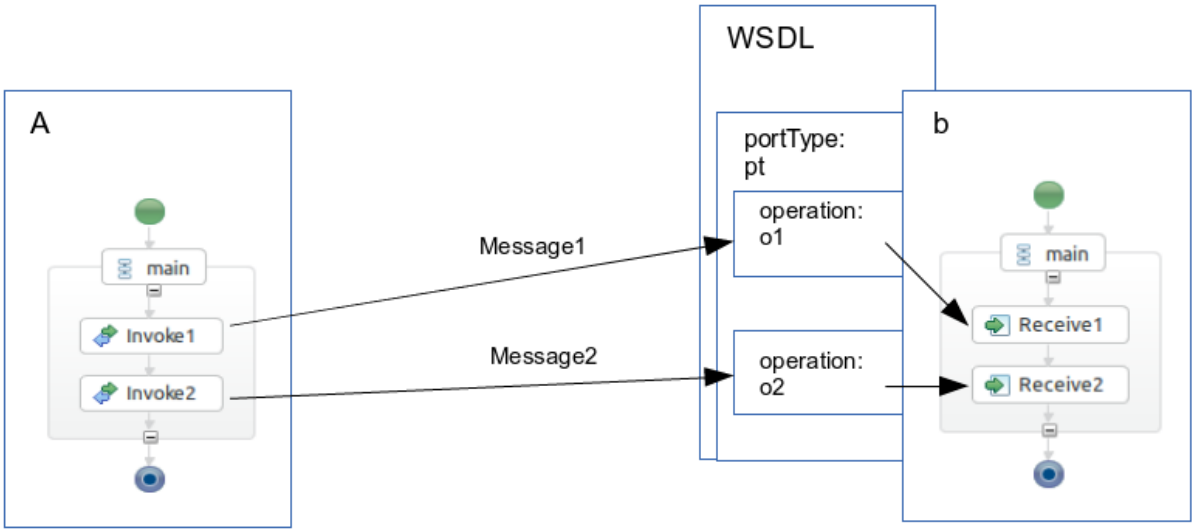


Figure 4.1.1: Condition (1)

For sending and receiving Message2, if it is used for a one-way communication, we don't need to declare new `partnerLink`.

If Message2 is a requestMessage for an asynchronous interaction, then b must provide another portType (which is implemented by A through WS-Addressing) that receives responseMessage.

Note that the `partnerLinks` and `partnerLinkTypes` used for one-way communication and asynchronous communication with WS-Addressing are different:

For one-way communication that already exists in Comm set:

Example 3.1.3 `<partnerLink>` A_*b_realizedBy_pt_o1* in A's BPEL file:

```
<partnerLink name="A_b_realizedBy_pt_o1"
```

```
      partnerLinkType="prefixOfB:A_b_realizedBy_pt_o1_PLT"
      partnerRole="b"
</partnerLink>
```

For asynchronous communication that is indicated by the messageLink:

Example 3.1.4 `<partnerLink>` A_*b_realizedBy_pt_o2* in A's BPEL file:

```
<partnerLink name="A_b_realizedBy_pt_o2"
      partnerLinkType="prefixOfB:A_b_realizedBy_pt_o2_PLT"
      partnerRole="b"
      myRole="A"
</partnerLink>
```

If Message2 is a responseMessage for a asynchronous interaction, then b must have sent a requestMessage to A. The partnerLink and partnerLinkType used for Message2 is also different from that are used for *comm*.

But according to [Rei07]:

So, we do not need to create new partner link declarations and a new partner link type. Instead, we can use the ones which have been created for the former message link.

So, the design of the relation Comm is not suitable for the situation that was just discussed.

4.2 The drawback of current implementation modifying the correlationSet

As the BPEL example in section 6.1.2 demonstrates, the `correlationSet` declared in each participant process could have the same name, as they are declared in different namespace. But when using the BPEL4Chor Designer, during the configuration of the CorrelationSets Grounding (Figure 2.11), if we set the same name to the `correlationSet` in each participant PBD, then the value that was set to the `correlationSet` that was declared lately will overwrite the value that was set to the `correlationSet` previously. So I set different names to the `correlationSet`s that were declared in different participants.

And this drawback also appears in the current implementation, I will explain the problem in the following.

According to [Rei07]:

Definition 3.32 (set of NCNames of correlation properties). The set CorrPropName is defined as the set containing all NCNames that are used as names of correlation properties and that are referenced to WSDL properties in the participant groundings.

Definition 3.33 (set of WSDL properties). The set Property is defined as the set containing all WSDL properties which are referenced in the participant groundings.

The identifier of an element of Property is the QName of the respective WSDLproperty attribute.

Definition 3.34 (the function assigning a property to each property name). The function $property_{CorrPropName}$ *: CorrPropName ⟶ Property is defined as the function that assigns a WSDL*

property to each name of a correlation property which is referenced to the property in the participant groundings.

For the properties declared in Listing 4.2.1, the set *CorrPropName* should contains a set of NCNames: {*agency_TravelerID*, *airline_TravelerID*}, the set *Property* should contains a set of QNames: {*agency:TravelerID*, *airline:TravelerID*}.

Listing 4.2.1: `properties` in a grounding file

```
<properties>
    <property WSDLproperty="agency:TravelerID" name="agency_TravelerID"/>
    <property WSDLproperty="airline:TravelerID" name="airline_TravelerID"/>
</properties>
```

But if the values of name attribute are set to the same NCName, the set *CorrPropName* can only store one of them.

As the current BPEL4Chor Designer can only generate the grounding file with different property names, so I used this approach unchanged.

The method *analyzePropertiesInGrounding(Data data)* (Listing 4.2.2) is called to fetch the property names and store them into *CorrPropNameSet*, and the method *setProperty_CorrPropNameMap(String propName, QName property)* (designed based on Definition 3.34) will store the value pair propName: String and property: QName into a HashMap.

Listing 4.2.2: method *analyzePropertiesInGrounding(Data data)* in class GroundingHandler

```
public void analyzePropertiesInGrounding(Data data){
                TProperties tprops = grounding.getProperties();
                for(TProperty tprop: tprops.getProperty()){
                        String propName = tprop.getName();
                        QName property = tprop.getWSDLproperty();
                        data.setCorrPropNameSet(propName);
                        data.setPropertySet(property);
                        data.setProperty_CorrPropNameMap(propName, property);
                        data.setNsprefix_PropertyMap(property, property.getPrefix());
                }
        }
```

5 Summary

The main task accomplished by this work was to implement the concept of transformation from BPEL4Chor to BPEL proposed in [Rei07]. The algorithms and definitions defined in [Rei07] have been encapsulated into corresponding classes. Class Data wraps the data structures used by the transformation process, and provides the corresponding methods to access these data. Class TopologyHandler provides the methods to fetch the desired information from messageLinks, participants and participantSets in topology file, and store the information in corresponding data structures in object data. Class Comm encapsulates the relation Comm that was designed to represent the patterns of interaction between participants. Class GroundingHandler provides the methods to analyze the messageLinks and properties in grounding file. Class PBDHandler encapsulates the methods for the transformation from PBD file to abstract BPEL file, the information stored in object data is used to modify the corresponding constructs in PBD file. Class WSDLHandler wraps a method that generates a WSDL file which declares the corresponding partnerLinkTypes.

In Chapter 3, some drawbacks of the design in [Rei07] have been put forward and the corresponding solutions have been presented. In section 4.1 a design drawback has been discussed, but due to the complexity and flexibility of BPEL that has been shown in section 2.2 and section 6.1, a feasible solution wasn't able to be given.

6 Appendix

6.1 BPEL Example

The examples are based on the travel agency example used in [DKLW07], All the BPEL processes have been successfully deployed and tested on Apache Ode. All code can be seen at https://code.google.com/p/huang-sa/source/browse/BPEL_Example_In_Appendix/. The way to implement the asynchronous interaction using is very flexible, in section 6.1.1 I demonstrate two ways to implement an asynchronous interaction using WS-Addressing, and in section 6.1.2 I demonstrate the implementation of an asynchronous interaction using Correlation Set.

6.1.1 Asynchronous Interaction using WS-Addressing

First, the asynchronous interaction could be realized by one `portType`, as Figure 6.1.1 demonstrates, in this case `portType` *AirlinePT* should contain both `operation`s *ReceivePriceInquiry* and *CallbackPriceInquiry* that are used by the asynchronous interaction, the *AirlinePT* is declared in process airline's WSDL file (Listing 6.1.1). The `portType` *TripOrderPT* is declared in process agency's WSDL file, the `operation` *ReceiveAndReplyTripOrder* is used to communicate with client, this communication is designed as a synchronous interaction (Listing 6.1.2). There is a `partnerLinkType` *Agency-Airline_RealizedBy_AirlinePT_PLT* declared in the process airline's WSDL file, which is used for indicating the type of the interaction and the portTypes that are used to fulfill the interaction, in this case the `portType`s used by both `role`s are the same(Listing 6.1.3). The entire BPEL project has been uploaded to: https://code.google.com/p/huang-sa/source/browse/BPEL_Example_In_Appendix/asynchronous_Interaction_Using_WS-Addressing/portTypes_In_One_Process/.

Listing 6.1.1: `portType` *AirlinePT* in WSDL file AirlineExampleArtifacts.wsdl

```
<portType name="AirlinePT">
    <operation name="ReceivePriceInquiry">
        <input message="tns:PriceInquiryRequestMessage"/>
    </operation>
    <operation name="CallbackPriceInquiry">
        <input message="tns:PriceInquiryResponseMessage"/>
    </operation>
</portType>
```

Listing 6.1.2: `portType` *TripOrderPT* in WSDL file AgencyExampleArtifacts.wsdl

```
<portType name="TripOrderPT">
    <operation name="ReceiveAndReplyTripOrder">
        <input message="tns:TripOrderRequestMessage"/>
        <output message="tns:TripOrderResponseMessage"/>
    </operation>
</portType>
```

Listing 6.1.3: `partnerLinkType` *Agency-Airline_RealizedBy_AirlinePT_PLT* in WSDL file AirlineExampleArtifacts.wsdl

```
<plnk:partnerLinkType name="Agency-Airline_RealizedBy_AirlinePT_PLT">
    <plnk:role name="airline" portType="tns:AirlinePT"/>
    <plnk:role name="agency" portType="tns:AirlinePT"/>
</plnk:partnerLinkType>
```

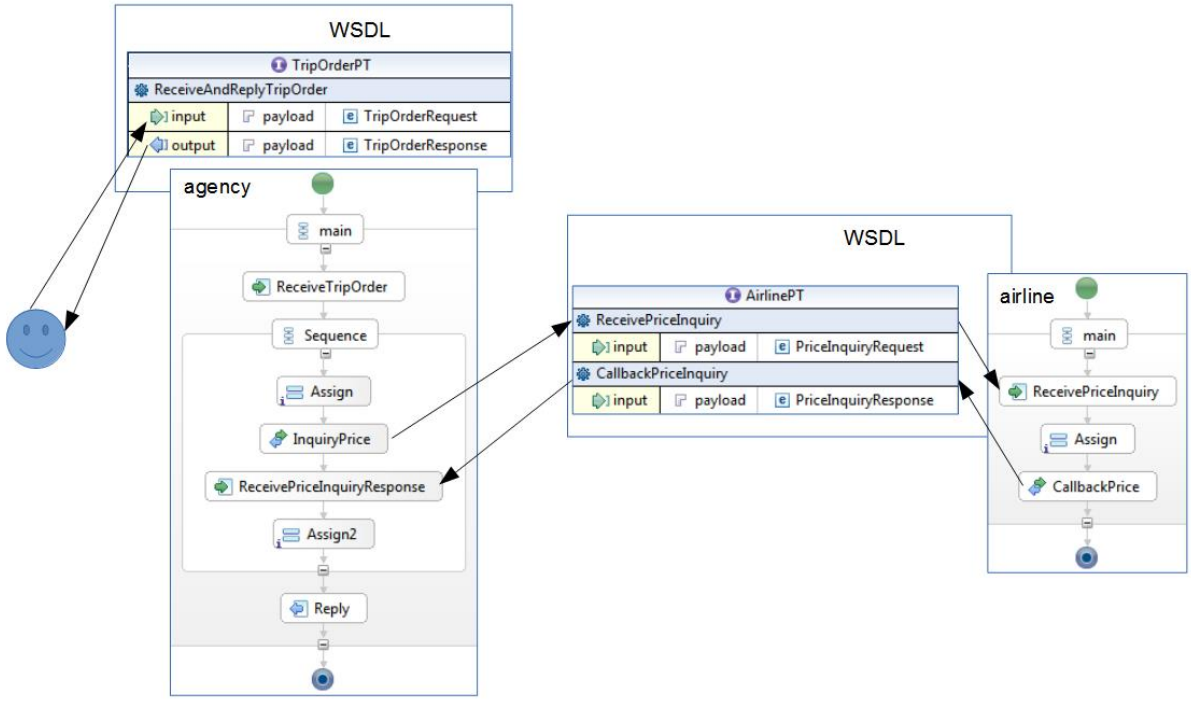


Figure 6.1.1 portType declared in one process

The BPEL processes are tested by the mechanism provied by Eclipse: "Test with Web Services Explorer", the result is shown in Figure 6.1.2.

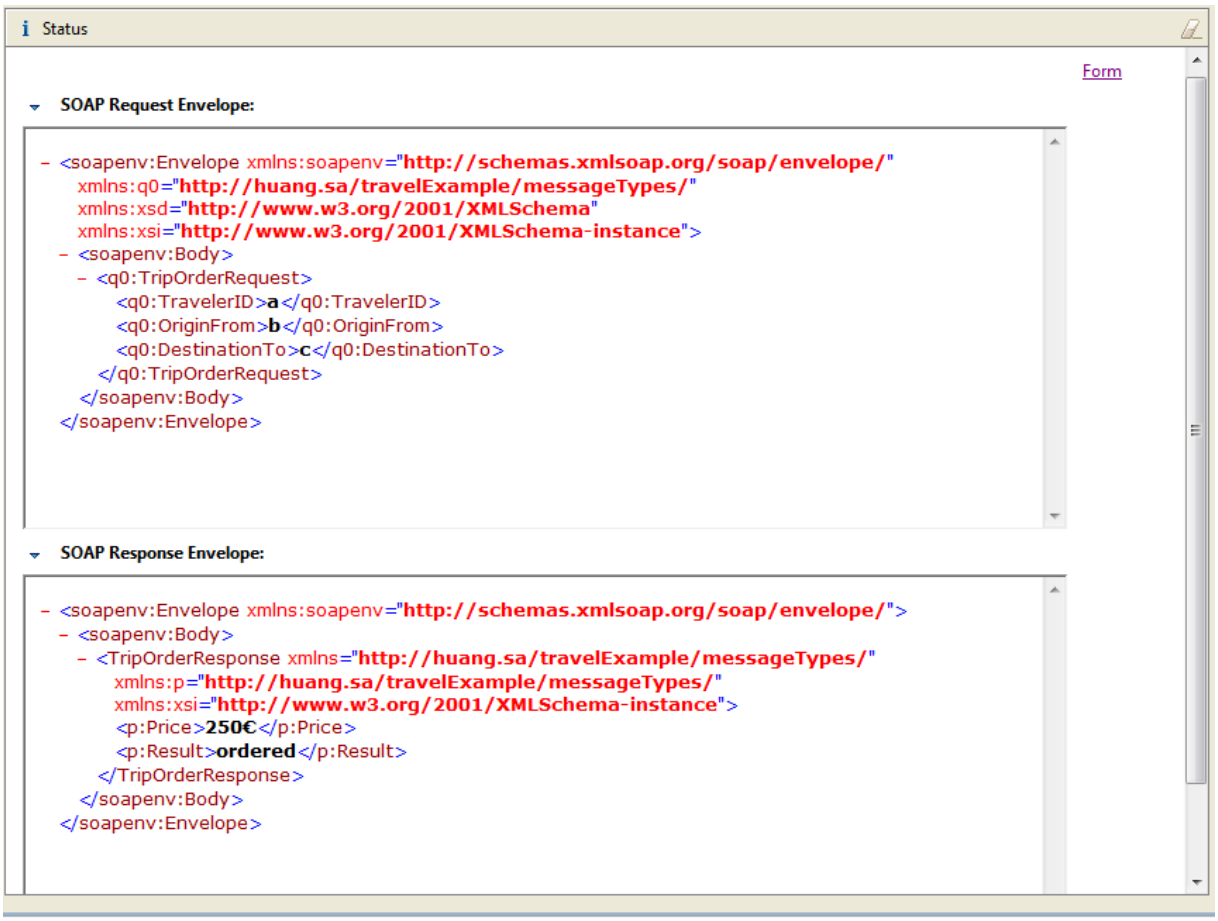


Figure 6.1.2: test result of AgencyExampleArtifacts.wsdl

The second way to implement an asynchronous interaction using WS-Addressing is to declare two `portTypes` separately in both participant's WSDL file. As Firgure 6.1.3 demonstrates, `portType` *AirlinePT* contains only `operation` *ReceivePriceInquiry*, the *AirlinePT* is declared in process airline's WSDL file (Listing 6.1.4). The `portType` *AgencyPT* is declared in process agency's WSDL file, it contains the `operation` *CallbackPriceInquiry* used by the asynchronous interaction, and the `operation` *ReceiveAndReplyTripOrder* used to communicate with client(Listing 6.1.5), The `partnerLinkType` *Agency-Airline_RealizedBy_AirlinePT_PLT* declared in the process airline's WSDL file, which is used for indicating the type of the interaction and the `portTypes` that are used to fulfill the interaction, in this case there are two different `portTypes`(Listing 6.1.6). The BPEL project: https://code.google.com/p/huang-sa/source/browse/BPEL_Example_In_Appendix/asynchronous_Interaction_Using_WS-Addressing/portType_In_Different_Processes/.

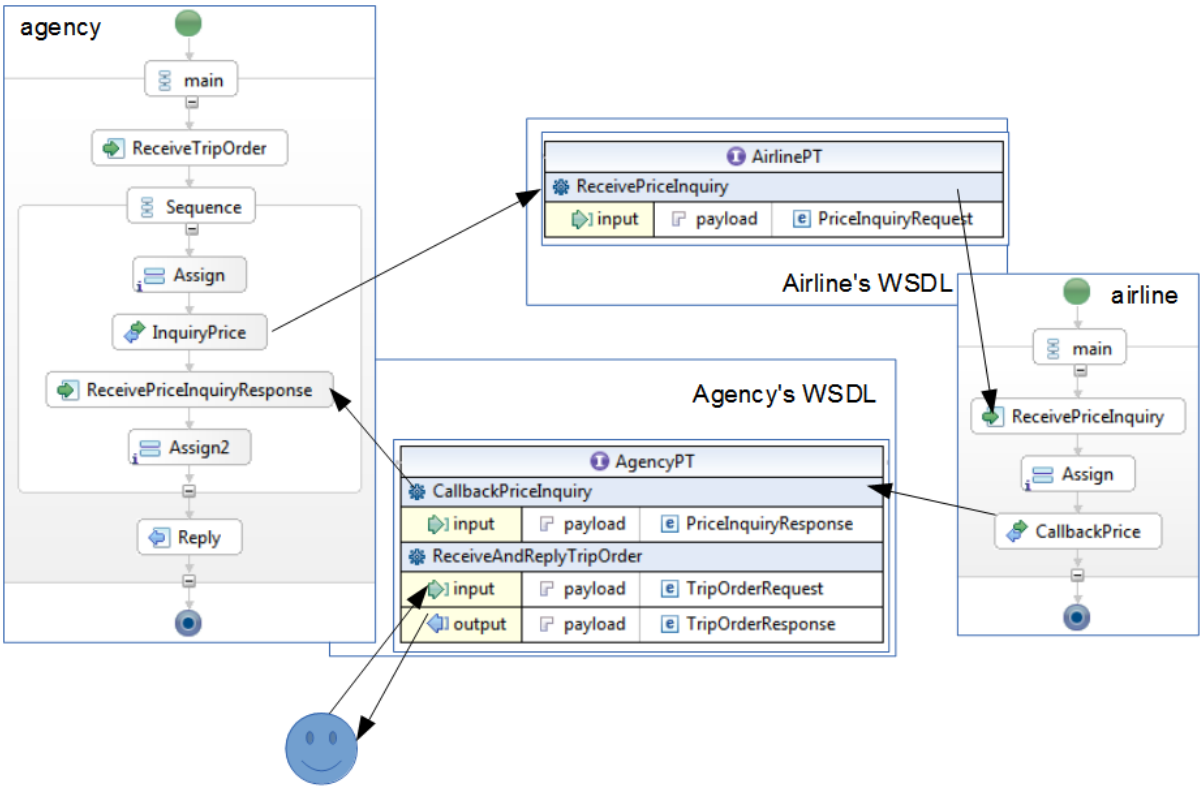


Figure 6.1.3 portTypes declared in different processes

Listing 6.1.4: `portType` *AirlinePT* in WSDL file AirlineExampleArtifacts.wsdl

```
<portType name="AirlinePT">
    <operation name="ReceivePriceInquiry">
        <input message="tns:PriceInquiryRequestMessage"/>
     </operation>
</portType>
```

Listing 6.1.5: `portType` *AgencyPT* in WSDL file AgencyExampleArtifacts.wsdl

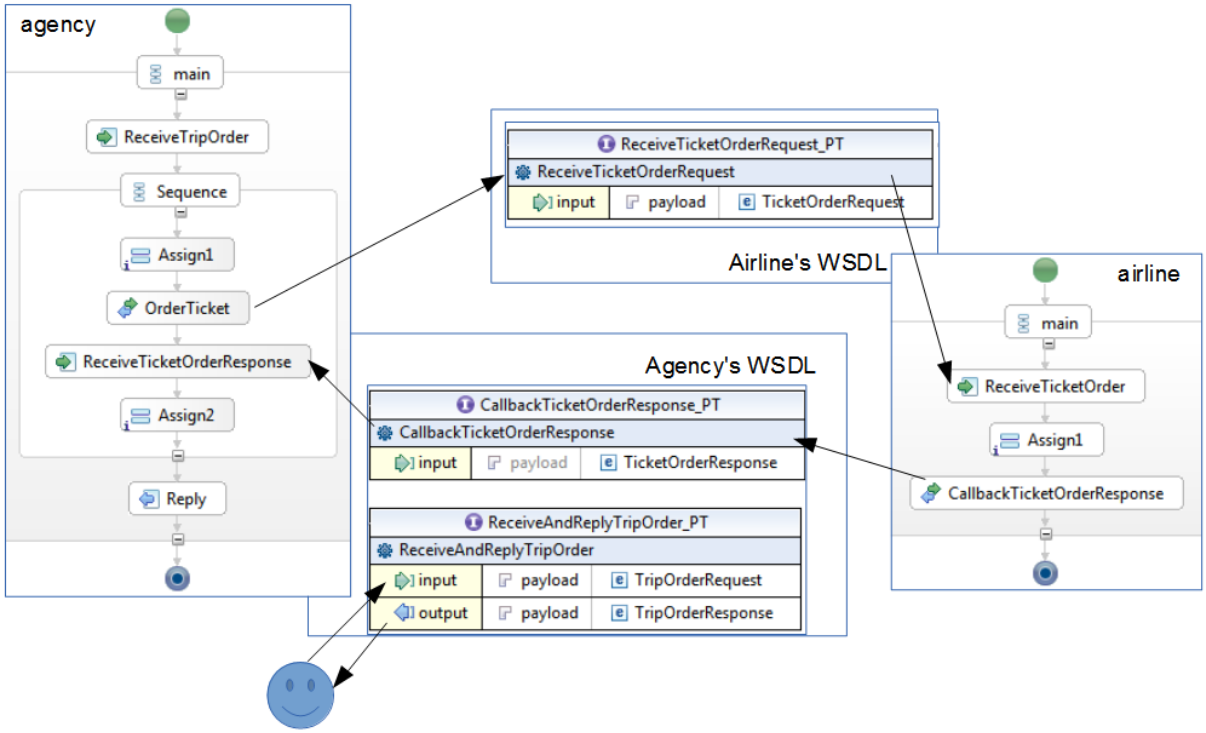
```
<portType name="AgencyPT">
    <operation name="CallbackPriceInquiry">
        <input message="tns:PriceInquiryResponseMessage"/>
    </operation>
```

```
    <operation name="ReceiveAndReplyTripOrder">
        <input message="tns:TripOrderRequestMessage"/>
    <output message="tns:TripOrderResponseMessage"/>
    </operation>
</portType>
```

Listing 6.1.6: `partnerLinkType` *Agency-Airline_RealizedBy_AirlinePT_PLT* in WSDL file AirlineExampleArtifacts.wsdl

```
<plnk:partnerLinkType name="Agency-Airline_RealizedBy_AirlinePT_PLT">
    <plnk:role name="airline" portType="tns:AirlinePT"/>
    <plnk:role name="agency" portType="agency:AgencyPT"/>
</plnk:partnerLinkType>
```

6.1.2 Asynchronous Interaction using Correlation Set



Firgure 6.1.4: asynchronous interaction using Correlation Set

As Figure 6.1.4 demonstrates, there are one separate `portType` declared in both participant process' WSDL file used to fulfill the interaction, *ReceiveTicketOrderRequest_PT* in airline process, and *CallbackTicketOrderResponse_PT* in agency process. A `correlationSet` *CorrelationSet* is declared in both participant process' BPEL file(Listing 6.1.7), note that both process could have the same name of the `correlationSet`, as they are in different namespace. The attribute `property` and `propertyAlias` is declared in both participant process' WSDL file(Listing 6.1.8). The activity *OrderTicket* and ReceiveTicketOrder transports the message that initiate the interaction, so the attribute `initiate` of `correlation` is set to *"yes"*(Listing 6.1.9 and Listing 6.1.10). The entire BPEL project: https://code.google.com/p/huang-sa/source/browse/BPEL_Example_In_Appendix/asynchronous_Interaction_Using_CorrelationSet/.

Listing 6.1.7: `correlationSet` *CorrelationSet* in BPEL file AgencyExample.bpel

```
<bpel:correlationSets>
```

```
    <bpel:correlationSet name="CorrelationSet" properties="tns:TravelerID"></bpel:correlationSet>
</bpel:correlationSets>
```

Listing 6.1.8: `property` *TravelerID* and correspoding `propertyAlias` in WSDL file AgencyExampleArtifacts.wsdl

```
<vprop:property name="TravelerID" type="xsd:string"/>
<vprop:propertyAlias messageType="airline:TicketOrderResponseMessage"
                     part="payload"
                     propertyName="tns:TravelerID">
    <vprop:query><![CDATA[/messageType:TravelerID]]></vprop:query>
</vprop:propertyAlias>
<vprop:propertyAlias messageType="airline:TicketOrderRequestMessage"
                     part="payload"
                     propertyName="tns:TravelerID">
    <vprop:query><![CDATA[/messageType:TravelerID]]></vprop:query>
</vprop:propertyAlias>
```

Listing 6.1.9: invoke activity *OrderTicket* in BPEL file AgencyExample.bpel

```
<bpel:invoke name="OrderTicket"
             partnerLink="Agency-Airline_RealizedBy_ReceiveTicketOrderRequest_PT"
             operation="ReceiveTicketOrderRequest"
             portType="airline:ReceiveTicketOrderRequest_PT"
             inputVariable="TicketOrderRequest">
    <bpel:correlations>
        <bpel:correlation set="CorrelationSet" initiate="yes"></bpel:correlation>
    </bpel:correlations>
</bpel:invoke>
```

Listing 6.1.10: receive activity *ReceiveTicketOrder* in BPEL file AirlineExample.bpel

```
<bpel:receive name="ReceiveTicketOrder"
              partnerLink="Agency-Airline_RealizedBy_ReceiveTicketOrderRequest_PT"
              operation="ReceiveTicketOrderRequest" portType="tns:ReceiveTicketOrderRequest_PT"
              variable="TicketOrderRequest" createInstance="yes">
    <bpel:correlations>
        <bpel:correlation set="CorrelationSet" initiate="yes"></bpel:correlation>
    </bpel:correlations>
```

6.2 Result of the transformation process

Listing 6.2.1: The generated BPEL file test_agency.bpel

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
         name="agency"
         targetNamespace="http://huang.sa/travelExample/agency"
         abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12">
  <partnerLinks>
    <partnerLink name="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT"
       partnerLinkType="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-plt"
       myRole="agency"
       partnerRole="selectedAirline"/>
  </partnerLinks>
  <correlationSets>
    <correlationSet properties="agency:agency_TravelerID agency:agency_TicketOrderNr"
                    name="agency_TicketOrder_CS"/>
  </correlationSets>
  <sequence name="main">
    <forEach counterName="i" parallel="yes" name="PriceInquiryFE">
      <startCounterValue>1</startCounterValue>
      <finalCounterValue>size of arrayOfEPRs[i]</finalCounterValue>
      <scope name="scope">
        <partnerLinks>
          <partnerLink name="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"
             partnerLinkType="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-plt"
             myRole="agency"
             partnerRole="currentAirline"/>
        </partnerLinks>
        <sequence name="sequence">
          <assign>
            <copy>
              <from variable="arrayOfEPRs[i]"/>
              <to partnerLink="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"/>
            </copy>
          </assign>
          <invoke partnerLink="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"
                portType="airline:ReceivePriceInquiryRequest_PT"
                operation="ReceivePriceInquiryRequest"
                name="SendPriceInquiryRequest"/>
          <receive partnerLink="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT"
                portType="agency:ReceivePriceInquiryResponse_PT"
                operation="ReceivePriceInquiryResponse"
                name="ReceivePriceInquiryResponse"/>
        </sequence>
      </scope>
    </forEach>
    <opaqueActivity name="choose airline"/>
    <invoke partnerLink="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT"
            portType="airline:ReceiveTicketOrderRequest_PT"
            operation="ReceiveTicketOrderRequest"
            name="SendTicketOrderRequest">
      <correlations>
        <correlation set="agency_TicketOrder_CS" initiate="yes"/>
      </correlations>
    </invoke>
    <receive partnerLink="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT"
            portType="agency:ReceiveTicketOrderResponse_PT"
            operation="ReceiveTicketOrderResponse"
            name="ReceiveTicketOrderResponse">
      <correlations>
        <correlation set="agency_TicketOrder_CS"/>
      </correlations>
    </receive>
  </sequence>
</process>
```

Listing 6.2.2: The generated BPEL file test_agency.bpel

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<process xmlns="http://docs.oasis-open.org/wsbpel/2.0/process/abstract"
         name="airline"
         targetNamespace="http://huang.sa/travelExample/airline"
         abstractProcessProfile="urn:HPI_IAAS:choreography:profile:2006/12">
  <partnerLinks>
    <partnerLink name="selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-agency"
      partnerLinkType="agency-selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-plt"
      myRole="selectedAirline"
      partnerRole="agency"/>
    <partnerLink name="currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-agency"
      partnerLinkType="agency-currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-plt"
      myRole="currentAirline"
      partnerRole="agency"/>
  </partnerLinks>
  <correlationSets>
   <correlationSet properties="airline:airline_TravelerID airline:airline_TicketOrderNr"
                   name="airline_TicketOrder_CS"/>
  </correlationSets>
  <sequence name="main">
     <receive partnerLink="currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-agency"
              portType="airline:ReceivePriceInquiryRequest_PT"
              operation="ReceivePriceInquiryRequest"
              createInstance="yes"
              name="ReceivePriceInquiryRequest"/>
     <invoke partnerLink="currentAirline_isRealizedBy_airline_ReceivePriceInquiryRequest_PT-agency"
             portType="agency:ReceivePriceInquiryResponse_PT"
             operation="ReceivePriceInquiryResponse"
             name="SendPriceInquiryResponse"/>
     <pick name="pick">
       <onMessage partnerLink="selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-agency"
                  portType="airline:ReceiveTicketOrderRequest_PT"
                  operation="ReceiveTicketOrderRequest">
       <correlations>
         <correlation set="airline_TicketOrder_CS" initiate="yes"/>
       </correlations>
       <sequence name="sequence">
          <invoke partnerLink="selectedAirline_isRealizedBy_airline_ReceiveTicketOrderRequest_PT-agency"
                  portType="agency:ReceiveTicketOrderResponse_PT"
                  operation="ReceiveTicketOrderResponse"
                  name="SendTicketOrderResponse">
            <correlations>
              <correlation set="airline_TicketOrder_CS"/>
            </correlations>
          </invoke>
        </sequence>
        </onMessage>
        <onAlarm>
          <for>"'Pt2M'"</for>
          <sequence name="sequence1">
            <opaqueActivity name="empty"/>
          </sequence>
        </onAlarm>
     </pick>
  </sequence>
</process>
```

7 References

[BPEL 2.0] Web Services Business Process Execution Language Version 2.0 OASIS Standard, 2007

[WSDL 1.1] Web Services Description Language (WSDL) 1.1, 2001

[WS-Addressing 1.0] Web Services Addressing 1.0, 2006

[Sha11] Sharma, Shiv: Correlation Sets, link to the blog, 2011

[Sad13] Saddala, Manohar: Correlation Sets, link to the blog, 2013

[BB09] Beecher, Virginia; Bradshaw, Deanna; Das, Tulika; Kennedy, Mark; Prazma, Alex and Purich, Peter: Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite, 11g Release 1 (11.1.1), 2009

[Rei07] P. Reimann. Generating BPEL Processes from a BPEL4Chor Description. Studienarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, 2007.

[DKLW07] Decker, Gero ;Kopp, Oliver ;Leymann, Frank ; Weske, Mathias: BPEL4Chor: Extending BPEL for Modeling Choreographies Technical Report. Unpublished Technical Report of the Institute of Architecture of Application Systems at the University of Stuttgart and the Hasso-Plattner-Institute at the University of Potsdam, revision of August 18th, 2007

[DKLW09] Decker, Gero ;Kopp, Oliver ;Leymann, Frank ; Weske, Mathias: Interacting services: From specification to execution, 2009

[Son13] O. Sonnauer. Modellierung von Scientific Workflows mit Choreographien. Diplomarbeit, Universität Stuttgart, Fakultät Informatik, Elektrotechnik und Informationstechnik, Germany, 2013.

[JAXB] Introduction to JAXB, http://docs.oracle.com/javase/tutorial/jaxb/intro/index.html

[JAVA6] http://docs.oracle.com/javaee/6/api/

[AcOS09] Using Arrays in a BPEL Process, ActiveVOS Education Center for Developers, http://www.activevos.com/content/developers/patterns/data/arrays/doc/arrays.html, 2009

[Kai11] Stapel Kai. BPEL-Tutorial - BPEL-Prozess mit dynamischen Invoke auf Apache ODE, http://www.se.uni-hannover.de/pages/de:tutorials_bpel_ode_dynamicinvoke, 2011

Declaration

I hereby declare that the work presented in this thesis is entirely my own. I did not use any other sources and references that the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Stuttgart, 05. March 2014 _____