Web Services Orchestration and Composition

Case Study of Web services Composition

WORKING PAPER

ABDALADHEM ALBRESHNE, PATRIK FUHRER, JACQUES PASQUIER

September 2009

Abstract

Web services technologies are becoming a de facto standard to integrate distributed applications and systems using XML-based standards. Developing applications that support web services interfaces will not be enough to provide complete and coordinated business processes. Thus, we need a new approach to compose these web services together in order to form web services orchestration and processes definition. Many new standards have been defined to solve this problem, for example BPEL4WS, and WSCI. This report aims to familiarize the reader with these different emerging standards and help understanding how web services orchestration can be achieved today.

Keywords: Web Services, SOA, WSDL, BPEL4WS, Services Composition, Services Orchestration.

Table of Contents

1 Introduction

1.1 Motivation

Web services and SOA (Service Oriented Architecture), viewed in a process-oriented perspective, need a language in order to define how services can be composed into business processes. Such definitions would allow describing abstract process definitions as well as executable processes [Erl06].

"*A process is an ordering of activities with a beginning and end: it has inputs (in terms of resources, materials, and information) and a specified output (the results it produces)*" [Pap08].

"*A workflow system automates a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules*" [Pap08].

In this paper we shall discuss the technologies used to orchestrate web services, give a detailed description of BPEL4WS (Business Process Execution Language for Web Services) [*OAS07*], explain how the participating web services are controlled by a central process and how various operations of web services are invoked and executed by this process.

1.2 Orchestration versus Choreography

Web services are used to distribute services over the Internet. They make operations of applications available or enable information systems to be invoked over the network [*Mat09*]. There are two ways to combine such services: either through orchestration or choreography.

In orchestration, the involved web services are under control of a single endpoint central process (another web service). This process coordinates the execution of different operations on the Web services participating in the process. The invoked Web services neither know and nor need to know that they are involved in a composition process and that they are playing a role in a business process definition. Only the central process (coordinator of the orchestration) is conscious of this aim, thus, the orchestration is centralized through explicit definitions of operations and the invocation order of Web services, (see Figure 1.1) [*Mat09*].

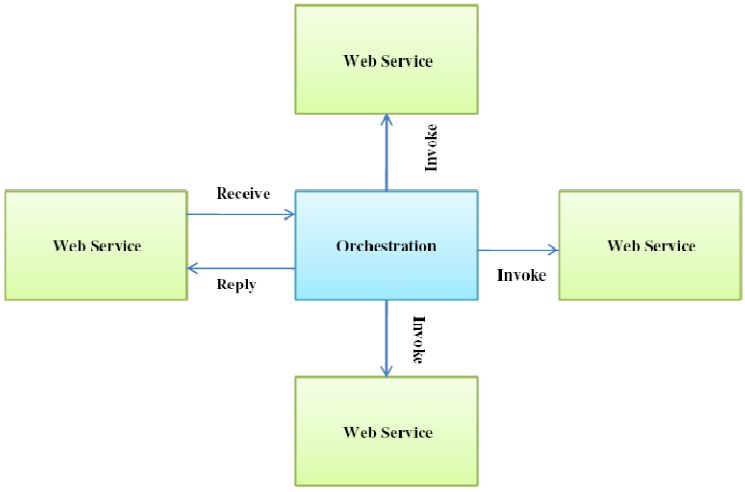

Figure 1.1 ***Composition of Web services with Orchestration (inspired from*** **[Pel03])**

Choreography, in contrast, does not depend on a central orchestrator. Each Web service that participates in the choreography has to know exactly when to become active and with whom to interoperate. Choreography is based on collaboration and is mainly used to exchange messages in public business processes. All Web services which take part in the choreography must be conscious of the business process, operations to execute, messages to exchange as well as the timing of message exchanges, (see Figure 1.2) [*Mat09*].

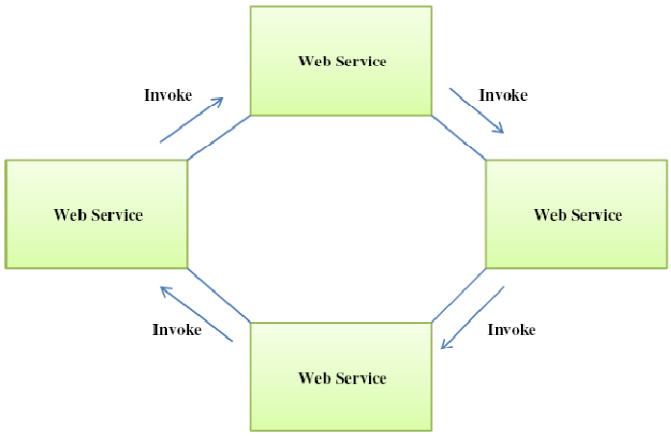

Figure 1.2 ***Composition of Web services with Choreography (inspired from*** **[Pel03])**

In comparison with choreography, orchestration is obviously more efficient and flexible when it comes to compose Web services to execute business processes. In fact, it has the following advantages over choreography [*Mat09*]:

1. The coordination of a process' components is centrally managed by a specific coordinator.
2. Web services can be incorporated without being aware that they are taking part in a larger business process.
3. Alternative scenarios can be put in place in case faults occur.

1.3 Early work

Recently many languages have emerged and proposed in the literature for composition and execution of web services, including: WSCL [*W3C102*], XLALNG [*Msd09*], WSFL, BPMN [*Wik05*], WSCI [*W3C02*], BPML [*Wik091*] and BPEL4WS [*OAS07*].

1. Web Services Conversation Language (WSCL)
   WSCL is a simple conversation language standard based on modeling the order of interactions among web services. Its function is similar to that of web services choreography [Pel03]. WSCL enables to define the abstract interfaces of Web services. Because WSCL conversation definitions are themselves XML documents, Web services infrastructures and development tools are able to understand them [*W3C102*]. There are four main elements to a WSCL specification as illustrated in Listing 1.1:

   - *Document type descriptions* specify the types (schemas) of the XML documents the service can accept and transmit in the course of a conversation.
   - *Interactions* model the actions of the conversation as document exchanges between two participants.
   - *Transitions* specify the ordering relationships between interactions.
   - *Conversations* list all the interactions and transitions that make up the conversation.

```
<?xml version="1.0" encoding="UTF-8"?>
<Conversation name="StoreFrontServiceConversation" version="1.01"
     xmlns="http://www.w3.org/2002/02/wscl10"
     initialInteraction="Start"  finalInteraction="End"
     targetNamespace="http://example.com/conversations/StoreFront101"
     hrefSchema="http://example.com/schema_files/StoreFront101.wscl"
     description="Conversation for a Store Front Service">
  <ConversationInteractions>
     list of all the interactions
  </ConversationInteractions>
  <ConversationTransitions>
         list of all the transitions
  </ConversationTransitions>
</Conversation>
```

Listing 1.1 ***Simplified WSCL example***

2. XLANG
   XLANG specification was developed by Microsoft. XLANG enables to construct business processes and provides the possibility to interact with web service providers. The specification supports sequential, parallel, and conditional process control flow. It deals also with exception handling and supports long-running transactions through compensation. XLANG can be exposed with a WSDL Interface [*Msd09*]. This language was merged with the WSFL language so as to create a BPEL4WS (Business Process Execution Language for Web Services).

3. Web Services Flow Language (WSFL)
   WSFL was developed by IBM to specify public as well as private processes. WSFL defines the execution order of activities and data exchanges for a process. A WSFL definition can also be exposed with a WSDL interface. WSFL provides support for the handling of exceptions while it does not directly support transactions [Pel03]. As mentioned above, this language was integrated with XLANG language to create a BPEL4WS (Business Process Execution Language for Web Services).

4. Business Process Modeling Notation (BPMN)
   BPMN is a graphical representation to describe business processes within a workflow. BPMN was realised by Business Process Management Initiative (BPMI) [*Wik05*]. The Business Process Modeling Notation (BPMN) constitutes a standard to model business process. The main goal of BPMN is to provide a solution form technical as well as business users to define and manage their processes by providing a standard notation which is intuitive yet capable of representing complex process semantics. The BPMN supports a mapping between the graphical notation to the implemented and execution languages, such as Business Process Execution Language (BPEL) [*Wik05*].
   The modeling in BPMN is done by simple diagrams with a small set of graphical elements. It should make it easy for business users as well as developers to understand the flow and the process. [*Wik05*] defines four basic categories of elements (some of them are illustrated in Figure 1.3). they are as follows:

   - Flow Objects: Events, Activities, and Gateways.
   - Connecting Objects: Sequence Flow, Message Flow, and Association.
   - Swimlanes: Pool and Lanel.
   - Artifacts (Artefacts): Data Object, Group and Annotation.

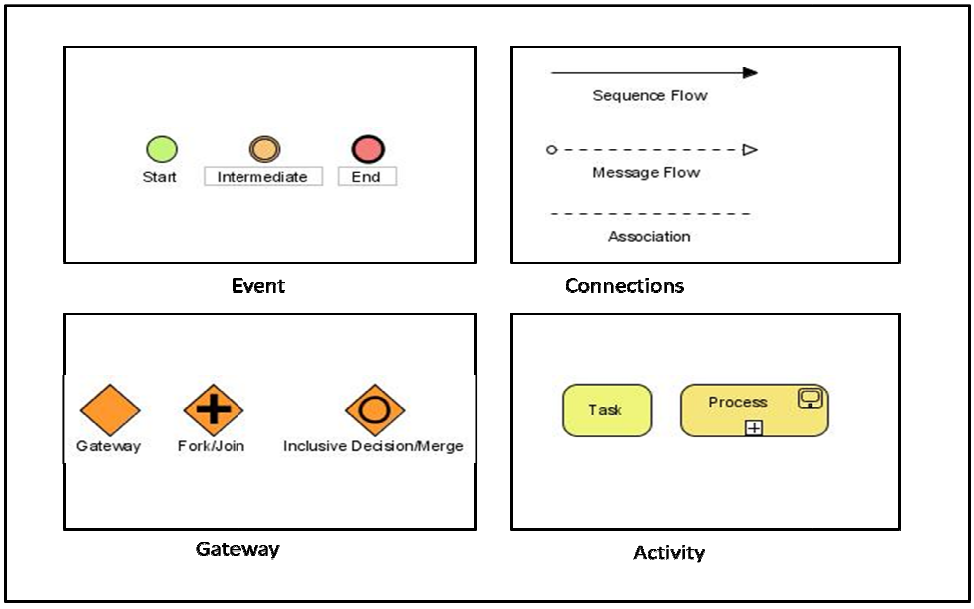

Figure 1.3 ***BPMN categories and elements (taken from* [*Wik05*])**

5. Web Services Choreography Interface (WSCI)
   WSCI is an XML-based interface description language that describes the messages among web services that are involved in a collaborative exchange [*W3C02*]. WSCI specification was realised by Sun, SAP, BEA and Intalio. WSCI provides message correlation, sequencing rules, exception handling, transactions as well as dynamic collaboration [Pel03]. WSCI does not define executable business processes, in contrast with BPEL4WS. Moreover, a single WSCI document only specifies one partner's contribution to a message exchange. WSCI has no single coordinating process orchestrating the interaction [Pel03].

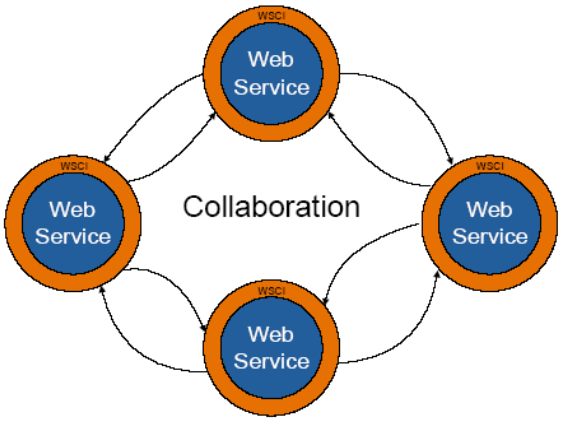

Figure 1.4 ***Web Services Choreography Interface (WSCI) (taken from*** **[Pel03])**

The following listing illustrates a simple example of WSCI. A Travel Agent process is created containing sequential activities. `ReceiveConfrimation` is one of them. There are also WSCI files for the buyer and the supplier in the process.

```
<? xml version = "1.0" ?>
  <wsdl:definitions name = "Travel Agent Dynamic Interface"
   targetNamespace = "http://example.com/consumer/TravelAgent"
   xmlns:tns = "http://example.com/consumer/TravelAgent"
   xmlns = "http://www.w3.org/2002/07/wsci10">
   <correlation name = "itineraryCorrelation"
            property = "tns:itineraryID">
   </correlation>
   <interface name = "TravelAgent">
     <process name = "PlanAndBookTrip"
          instantiation = "message">
        <sequence>
        <action name = "ReceiveConfirmation"
            role = "tns:TravelAgent"
            operation = "tns:TAtoTraveler/bookTickets">
           <correlate correlation="tns:itineraryCorrelation"/>
          <call process = "tns:BookSeats" />
        </action>
        <action name = "SendStatement"
            role = "tns:TravelAgent"
            operation = "tns:TAtoTraveler/SendStatement"/>
        </action>
        </sequence>
     </process>
    </interface>
</wsdl:definitions>
```

Listing 1.2 ***Simplified WSCI example (taken from*** **[*W3C02*])**

6. Business Process Management Language (BPML)
   BPML is used to describe business processes. The specification was realised by

the Business Process Management Initiative (BPMI.org). The specification is comparable to BPEL4WS because it provides similar process flow constructs and activities. It allows to define basic activities for receiving and invoking services, structured activities that deal with conditional choices, sequential and parallel activities, joins, and looping. Other advantages included in BPML are persistence, roles and instance correlation. It is capable of managing long-lived processes. XML messages are used to enable interaction between several participants, by defining roles and partner components similar to the BPEL constructs. BPML is also used to compose sub-processes into a larger business process. BPML supports transactional support as well as exception handling mechanisms [Pel03].

```
<bpml:process name="example">
<bpml:event activity="receiveRequest"/>
<bpml:context>
  <bpml:property name="orderID" type="type:identifier"/>
  <bpml:property name="customerService" type="inst:service"/>
  <bpml:property name="orderDetails" element="type:orderDetails"/>
  <bpml:property name="invoiceDetails" element="type:invoiceDetails"/>
</bpml:context>

<bpml:action name="receiveRequest" portType="srv:exampleServiceType" operation="request">
<bpml:input element="type:orderID" property="bp:orderID"/>
</bpml:action>

<bpml:sequence>
  <bpml:context>
    <bpml:exception name="cancel">
        <bpml:event activity="receiveCancelRequest"/>
        <bpml:action name="receiveCancelRequest" portType="srv:exampleServiceType"
                     operation="cancel" correlate="bp:orderID">
           <bpml:input element="type:orderID" property="bp:orderID"/>
       </bpml:action>
     </bpml:exception>
  </bpml:context>
</bpml:sequence>

</bpml:process>
```

Listing 1.3 ***Simplified BPML process example***

7. Business Process Execution Language for Web Services (BPEL4WS)
   A new web services workflow specification from IBM, Microsoft and BEA called BPEL4WS (Business Process Execution Language for Web services) has recently been developed. This specification is a result of integrating both XLANG and WSFL. BPEL4WS allows to model the behaviour of web services in a business process interaction. The specification has an XML-based grammar to implement and define the control logic needed to orchestrate web services involved in a process flow. This implemented language can then be interpreted and executed by an orchestration engine such as activeVOS [*Act091*], apache ODE [*The09*], or jBPM JBOSS [*Com09*]. The engine deals with the coordination of the various activities in the process and compensates the system when errors occur [Pel03]. We will discuss this aspect in further detail in the next Chapter.

1.4 Summary

We have discussed the different standards in the field of web services orchestration and choreography. WSCI is related to the choreography approach, it focuses on public message exchanges between web services while BPEL4WS defines how to create executable business processes. With BPEL4WS, only one of the partners orchestrates the process and takes control over the others. BPML is similar to BPEL4WS, both providing capabilities to define a business process.

2 Business Process Execution Language for Web Services (BPEL4WS)

2.1 Introduction

The Business Process Execution Language for Web Services (BPEL4WS), which is also referred to as BPEL, is currently a de facto standard for building, specifying and executing business processes for web services composition and orchestration.

BPEL composes web services to get a specific result. The composition result is named a *process*, involved services are called *partners*, and message exchange is referred to as an *activity*. In other words, a process contains a set of activities and it invokes external partner services using a WSDL interface [Mil09].

A BPEL process defines the order in which involved Web services are composed, either in sequence or in parallel. BPEL allows describing conditional activities. An invocation of a Web service can for example rely on the result of another web service's invocation. With BPEL, it is possible to create loops, declare variables, copy and assign values as well as to use fault handlers. Complex business processes can be built algorithmically by using all these constructs. It can be helpful to describe business processes graphically through UML [*Wik09*] (Unified Modelling Language) activity diagrams [*Mat09*].

BPEL supports two different ways of describing business processes that support orchestration and choreography [*Mat09*]:

1. Executable processes allow for specifying the details of business processes. They follow the orchestration paradigm and can be executed by an orchestration engine.
2. Abstract business protocols allow specification of the public message exchange between parties only. They do not include the internal details of process flows and are not executable. They follow the choreography paradigm.

2.2 The Structure of a Business Process

The role of BPEL is to define a new web service by composing a set of existing services through a process-integration type mechanism with control language constructs.

To build a BPEL process, the following elements are required:

1. Business partners which will interact with the process.
2. Information about the data exchange type between the process and the business partners.

3. A workflow that defines the order of process execution (receive and invoke web services, map data and reply to business partners).
4. A BPEL process needs a WSDL file in order to create an executable BPEL definition. The WSDL file consists of the namespace, partner link types, operations, and messages which are required to define process activities (for further details, see Section 2.4).
5. A BPEL must have namespaces which point to associated WSDL schema locations and other resources such as XSL style sheets and xml files used in resource catalogue custom functions.

2.3 How to build a BPEL process?

A process consists of a set of activities. It interacts with external partner services through a WSDL interface. To define a BPEL process, we use

1. A BPEL source file (.bpel) which describes the execution order, activities and conditional behaviours.
2. A process interface (.wsdl) that defines the ports, the namespace, partner link types, operations, and messages which are required to determine process activities, and WSDL files are needed in order to create a valid, executable BPEL definition.

2.4 BPEL Process Elements and Properties

The main elements of a Business Process Execution Language (BPEL) definition are shown in Listing 2.1. A BPEL process contains several structural parts:

```
<process name="ProcessName">
  <!-- Definition of roles of process participants -->
  <partnerLinks> ... </partnerLinks>
  <!-- Data and state variables used within the process -->
  <variables> ... </variables>
  <!-- Correlation comment -->
  <correlationSets> ... </correlationSets>
  <!-- Exception management -->
  <faultHandlers> ... </faultHandlers>
  <!-- Message and timeout event handler -->
  <eventHandlers> ... </eventHandlers>
  <!-- Processing steps -->
  <sequence>
  </sequenece>
  activities*
</process>
```

Listing 2.1 ***BPEL process elements and properties***

- The process element
  It is the root element of BPEL process definition. It has a name attribute and it is used to specify the definition related namespaces.

- Partner Links elements
  These elements in a BPEL process define the interaction of participating services with the process. They describe what are the processes and services' roles at that

step in the flow, and they define the kind of data which can be handled by the parties in those roles.

- Variables elements
  A BPEL process allows to declare variables in order to receive, manipulate, and send data. For example, the process receives a reservation order from a client and assigns the message to an input XML variable, and this variable can be copied to another operation. Variables are defined in either WSDL message types, or XML schema types, or XML schema elements.

- Fault Handlers element
  A fault handler determines the activity which the process has to perform when an error occurs.

- Correlation Sets element
  Message correlation is the BPEL mechanism which enables several processes to interact in Stateful conversation. Because there can be many instances of the same process running at the same time, message correlation element provides a way to decide which process instance a specific message is sent for [Pap08].

- Event handling element
  An event handler allows the scope to response to events, or the expiration of timers, at any time during the execution of a scope. `<EventHandlers>` supports two types of events: message events and alarm events. A message event has a request/response or one-way operation implemented by a business process. An alarm event implements temporal semantics in which it may occur at a defined moment in time. An event handler resembles to pick activity [Pap08].

- Activities elements
  Activities are the processing steps, performed in a BPEL body. BPEL supports primitive as well as structure activities [*Mat09*].

  Primitive Activities

  - `<invoke>`: invoking other Web services' operations.
  - `<receive>`: waiting for the client to invoke the business process by sending a message.
  - `<reply>`: generating a response for synchronous operations.
  - `<assign>`: manipulating data variables.
  - `<throw>`: indicating faults and exceptions.
  - `<wait>`: waiting for some time.
  - `<terminate>`: terminating the entire process.

  Structure and control activities
  The combination of the primitive activities described above enables to build complex algorithms which define exactly the steps of business processes. BPEL

provides several structure activities that allow to combine those primitive activities. The main structure activities are the following:

- <sequence>: defining a set of activities that will be executed in an ordered sequence.
- <flow>: defining a group of activities which will be invoked in parallel.
- <switch>: Case-switch construct for implementing branches.
- <while>: defining loops.
- <pick>: selecting one of several alternative paths.

2.5 A simple example in BPEL

In this section we will consider a simple example to show a global view to how a BPEL process is designed. Travel Reservation application, shown in Figure 2.1 explains the BPEL constructs that we introduced in the previous section. In this example, we show a client making a request in order to buy a flight ticket, rent a car and reserve a room for his travel. The Travel Agent (central process) will communicate with a hotel service, a car rent service and an airline service to fulfill the client's wishes. Once a reservation is made, it is sent back to the client. We will begin developing this example on Chapter 3.

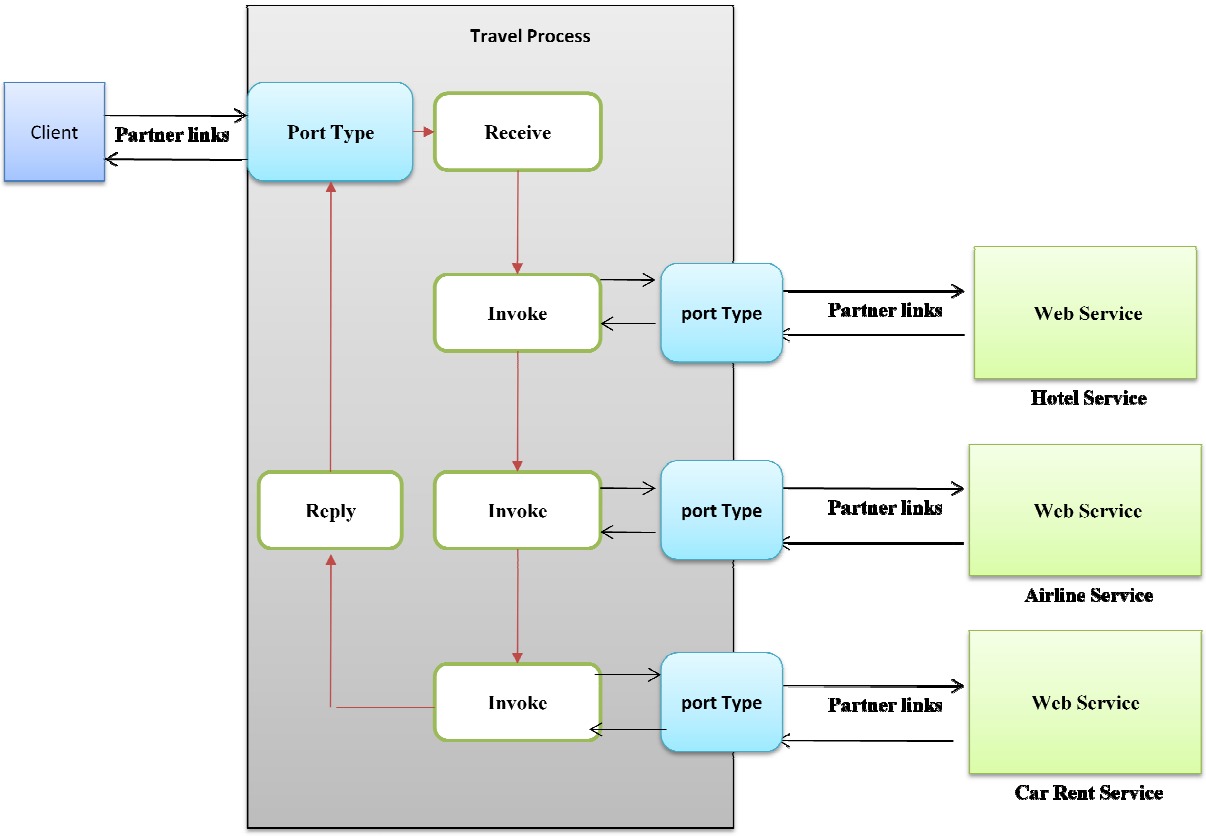

Figure 2.1 *Travel Reservation Agent Process*

2.6 BPEL features

As mentioned in [Pap08], using BPEL offers many interesting benefits such as:

- Being based on XML, so it is portable across platforms and vendors.
- Supporting a reliable, asynchronous communication, endpoint management and process persistence.
- Providing the ability to manage both atomic transactions as well as long-running business transactions.
- Being rapidly prototyped, developed and modified - even by individuals with limited development experience.
- Enabling to model business process collaboration through `<partnerLink>`.
- Enabling to model the execution control of business processes (through the use of a self-contained block and transition structured language that supports representation of directed graphs).
- Separating abstract definition from concrete binding (static and dynamic selection of partner services via endpoint references).
- Representing participants' role relationships (through `<partnerLinkType>`).
- Supporting compensation (through fault handlers and compensation).
- Supporting context (through the `<scope>` mechanism).
- Offering service composability (structured activities can be nested and combined arbitrarily).

3 Web Services Composition Case Study (BPEL4WS)

3.1 Building a Business Process

This example demonstrates the basics of creating an executable BPEL business process; the code for it can be downloaded from [*Abd09*] and deployed to the Glassfish Server.

In a typical scenario, the BPEL business process receives a request. To fulfil it, the process invokes the involved Web services and then responds to the original caller. Because the BPEL process communicates with other Web services, it depends on the WSDL descriptions of the Web services invoked by the composite Web service.

To understand how business processes are described with BPEL, we will define a simplified business process for a travel reservation agent: The client invokes the business process, specifying his name, the destination, the departure date and the return date. The BPEL process checks if it is possible to book the flight ticket with Airline A. We assume that Airline Company A provides a Web service through which such bookings can be made. Then the BPEL checks if it is possible to rent a car from CAR-Rent Company. We assume as well that the CAR rent Company provides a web services through which such renting can be made. Finally, the BPEL process reserves a room from a hotel and returns the travel reservation summary to the client. We build a synchronous BPEL process. We assume that the Web services for booking the air plane ticket and the hotel as well as renting the car are synchronous because such data can be obtained immediately and returned to the requestor.

Defining a business process in BPEL means essentially to create a new Web service that is a composite of existing services. The interface of the new BPEL composite Web service uses a set of port types through which it provides operations like any other Web service. Figure 3.1 shows a view of the process which we are using as an example.

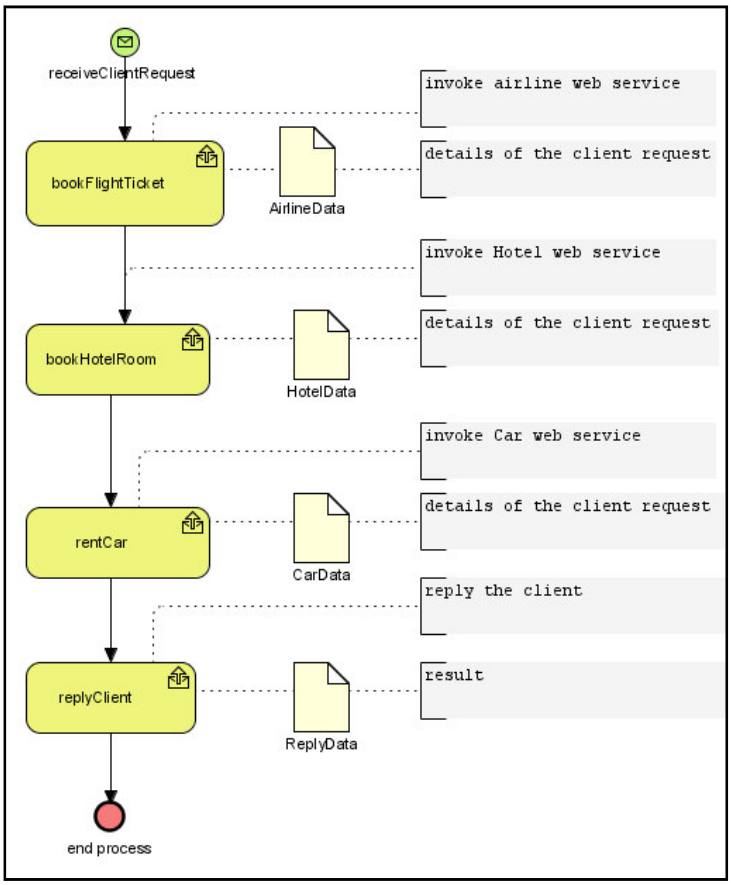

Figure 3.1 ***Travel Reservation Process***

To develop a BPEL process, we go through the following steps:

1. Get familiar with the involved Web services.
2. Define the WSDL for the BPEL process.
3. Define partner link types.
4. Define the BPEL process:
   - Define the namespaces.
   - Define the partner links.
   - Define variables.
   - Define the process logic.
5. Build the application.
6. Deploy the application.

Step 1: Get familiar with the involved Web services

Before we start writing the BPEL process definition, we need to get some information about the Web services involved in the process. These services are called *partners*. This example involves the A Airlines web service, the Hotel web service and the Car rent web service.

Airline Web Service: The A Airline Web service is synchronous. It specifies one port type: `AirlineReservation` which is used to book flight tickets through the `bookTicket` operation. This operation returns the reservation result to the process (see Figure 3.2).

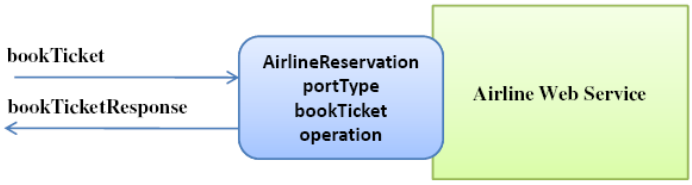

Figure 3.2 ***Airline Web service***

Hotel Web Service: The Hotel web service provides the `HotelReservation` portType through which the room can be reserved by calling the `reserveRoom` operation. This operation will have an input message called `reserveRoom` and an output message (return message) `reserveRoomResponse` which contains a confirmation and the room number (see Figure 3.3). How to build Hotel web service is outside the scope of this paper.

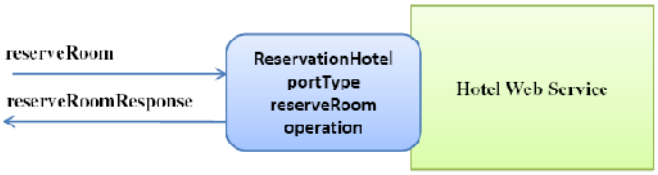

Figure 3.3 ***Hotel Web service***

Car Web Service: The Car Web service provides the `CarReservation` portType through which the car can be rent by the client by calling the `rentCar` operation. This returns a confirmation and the car type attributed to the client (see Figure 3.4).

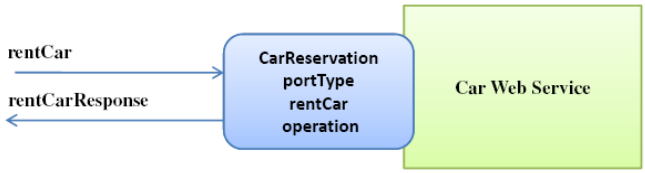

Figure 3.4 ***Car Web service***

Step 2: Define the WSDL for the BPEL Process

We have to expose the business travel reservation process as a Web service. The second step is to define the WSDL for this process. The WSDL specification of the process (see Listing 3.1) defines the available port types for its clients as well as operations, messages, partner link types, and properties of interest to the process. The process receives messages from its clients and returns results. It has to expose

the `TravelReservation` port type (see Figure 3.5) which specifies an input message used to return the result to the client.

We use the Netbeans tool to build this example. In [*She09*], there is a tutorial which provides an overview of the sample project, `SynchronousSample`, and illustrates deploying, executing and testing a synchronous BPEL process using the NetBeans IDE 6.5 bundle with all the necessary runtimes. We assume that you prepared the same environment as recommended in the tutorial mentioned above.

Figure 3.5 ***Process Endpoint***

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="TravelAgent"
             targetNamespace="http://localhost/TravelAgent/TravelAgent"
    xmlns:tns="http://localhost/TravelAgent/TravelAgent"
    xmlns:ns="http://xml.netbeans.org/schema/TravelAgent"
    <types>
        <xsd:schema targetNamespace="http://localhost/TravelAgent/TravelAgent">
            <xsd:import namespace="http://xml.netbeans.org/schema/TravelAgent"
            schemaLocation="TravelAgent.xsd"/>
        </xsd:schema>
    </types>
    <message name="requestMessage">
       <part name="inputType" element="ns:typeA"/>
    </message>
    <message name="responseMessage">
        <part name="resultType" element="ns:typeA"/>
    </message>
    <portType name="portTypeClient">
        <operation name="getTravelOffer">
            <input name="input1" message="tns:requestMessage"/>
            <output name="output1" message="tns:responseMessage"/>
        </operation>
    </portType>
    <binding name="binding1" type="tns:portTypeClient">
        <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document"/>
        <operation name="getTravelOffer">
            <input name="input1">
                <soap:body use="literal"/>
            </input>
            <output name="output1">
                <soap:body use="literal"/>
            </output>
        </operation>
    </binding>
    <service name="service1">
        <port name="port1" binding="tns:binding1">
            <soap:address location="http://localhost:18181/TravelAgent"/>
        </port>
    </service>
    <plink:partnerLinkType name="partnerlinktypeforclient">
        <plink:role name="partnerlinktyperole1" portType="tns:portTypeClient"/>
    </plink:partnerLinkType>
</definitions>
```

Listing 3.1 ***The simplified WSDL for the BPEL process***

Step 3: Define Partner Link Types

The third step is to define the partner link types. Partner link types represent the interaction between a BPEL process and the involved web services (*Partners*) which the BPEL process invokes.

In this example, there are four different partners: the client who requests the reservation and invokes the Process, the airline web service to book the flight ticket, the hotel web service to book a room and the car rent web service. The WSDL of the process imports the WSDL of other web services providers and defines the partner link types for them.

The four partner link types are as following:

- ClientPartnerLink: used to describe the interaction between the client and the BPEL process itself. This interaction is synchronous. This partner link type is declared in the WSDL of the BPEL process.

- AirlinePartnerLink: used to describe the interaction between the Airline service provider and the BPEL process itself. This interaction is synchronous. This partner link type is defined in the BPEL process.

- HotelPartnerLink: used to describe the interaction between the BPEL process and the Hotel Reservation Web service. This interaction is synchronous. This partner link type is defined in the BPEL process.

- CarPartnerLink: describes the interaction between the BPEL process and the Car rent Web service. This interaction is synchronous. This partner link type is defined in the BPEL process.

Step 4: Define the Business Process

Now, we start writing the BPEL process definition (see Listing 3.2). The BPEL process waits for an incoming message from the client which starts the execution of the business process. In our example, the client initiates the BPEL process through sending an input message.

In the following, we briefly describe the main elements of the BPEL process construction:

- Namespaces: here we define the target namespace and the namespaces to access the Airline WSDL, Hotel WSDL, Car WSDL, and the BPEL process WSDL description files.

- Partner Links: we specify the partner links which define different partners that interact with the BPEL process. Each partner link is related to a specific `partnerLinkType` that characterizes it.

- Variables: they are used to store, reformat and transform messages. We often need a variable for every message sent to the partners and received from them. Each variable has a specific type. We can use a WSDL message type, an XML

Schema simple type, or an XML Schema element. In this example, we use XML Schema element types for all variables.

- Process logic definition: the process main body specifies the order in which the partner Web services are invoked. In our example, we start with a `<sequence>` that allows defining several activities that will be performed sequentially. Within the sequence, we first specify the input message that starts the business process. We do that with the `<receive>` construct, which waits for the matching message - in our case the `TravelRequest` message. Then, we link the message reception with the client partner, and wait for the `bookTicket` operation to be invoked on the port type `AilrlineReservation`. We store the received message into the `TravelRequest` variable. In a similar way, we continue invoking the other web services provider (Hotel and Car web services). After have finished the services invocation, the process replies to the client with the reservation details.

Step 5: Build the complete application

- Preparing the environment
  Before starting building our example, we need the following software resources to be installed:

Software or Resource	Version Required
Java Development Kit (JDK) [*Sun09*]	Version 5
Glassfish Application server [*Net09*]	(it is bundled with NetBeans IDE)
NetBeans IDE [*Net09*]	Version 6.5

Table 3.1 ***Software Resources***

- Build the project
  Netbeans tools enable us to build a complete project. With the same steps explained in [*She09*], we create a project named `TravelAgent`. The process files node contains the below-mentioned items:

  - `TravelProcess.bpel`
  - `Travel.wsdl`
  - `TravelSchema.xsd`

- Import external WSDL files
  The three services providers Airline, Hotel and Car rent Web services have been exposed over the network and their WSDL file can be accessed by URLs as following:

Web service	URL
Airline	http://localhost:8080/AirlineReservation/AilrlineReservationService?WSDL
Hotel	http://localhost:8080/HotelReservation/HotelReservationService?WSDL
Car Rent	http://localhost:8080/CarReservation/CarReservationService?WSDL

Table 3.2 ***The WSDL URLs for the three Web services providers***

Netbeans tool enables also to import the WSDL files for any external web service provider. Thus, we import the three WSDL files for the three service providers with their xml Schema.

- Declare variables
  We declare variables in the travelschemal.xsd file to use them inside our process such as client name, his departure date, his arrival date, hotel name, company airline name, etc.

- Develop the process
  When we created the travel process, it was just an empty process with a start and end element. Now, we add the logic and implement the scenario that we have designed before (see Listing 3.2). During this step, we create the below-mentioned order:

  - Starting with defining the name spaces.
  - Importing the `wsdl` files for the three providers and the process.
  - Defining the partner links.
  - Declaring the required variables.
  - Starting the process logic with receive activity to wait for the client's request.
  - Assigning the incoming message as input to the airline service.
  - Invoking the airline services to book the ticket and get a return result.
  - Assigning the client's name and his departure and arrival dates to hotel service's input.
  - Invoking the hotel web service and get a return result.
  - Replying to the client with the final result.

```
<?xml version="1.0" encoding="UTF-8"?>

<process name="TravelAgent"
         targetNamespace="http://enterprise.netbeans.org/bpel/TravelAgent/TravelAgent_1"
         xmlns:ns1="http://localhost/TravelAgent/TravelAgent"
         xmlns:ns2="http://xml.netbeans.org/schema/TravelAgent">

    <import namespace="http://xml.netbeans.org/schema/TravelAgent"
            location="TravelAgent.xsd"
            importType="http://www.w3.org/2001/XMLSchema"/>
    <import namespace="http://localhost/TravelAgent/TravelAgent"
            location="TravelAgent.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.airline/"
            location="localhost_8080/AirlineReservation/AilrlineReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.hotel/"
            location="localhost_8080/HotelReservation/HotelReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>
    <import namespace="http://ws.car/"
            location="localhost_8080/CarReservation/CarReservationService.wsdl"
            importType="http://schemas.xmlsoap.org/wsdl/"/>

<partnerLinks>
        <partnerLink
            name="ClientPortEntry"
            partnerLinkType="ns1:partnerlinktypeforclient"
            myRole="partnerlinktyperole1">
```

```
29        </partnerLink>
30        <partnerLink name="AirlinePartnerLink1"
31        xmlns:tns="http://enterprise.netbeans.org/bpel/AilrlineReservationServiceWrapper"
32        partnerLinkType="tns:AilrlineReservationLinkType"
33        partnerRole="AilrlineReservationRole"/>
34         ……….
35    </partnerLinks>
36
37    <variables>
38        <variable name="RentCarOut" xmlns:tns="http://ws.car/"
39                  messageType="tns:rentCarResponse"/>
40         ……..
41    </variables>
42
43    <sequence>
44
45        <receive name="start" partnerLink="ClientPortEntry"
46            operation="getTravelOffer" portType="ns1:portTypeClient"
47            variable="inputVar" createInstance="yes">
48        </receive>
49
50        <assign name="AssignAirlineInput">
51            <copy>
52                <from>$inputVar.inputType/ns2:name</from>
53                <to>$BookTicketIn.parameters/parameter/name</to>
54            </copy>
55        </assign>
56        <invoke name="InvokeAirline" partnerLink="AirlinePartnerLink1"
57                operation="bookTicket" xmlns:tns="http://ws.airline/"
58                portType="tns:AilrlineReservation" inputVariable="BookTicketIn"
59                outputVariable="BookTicketOut"/>
60
61       <assign name="AssignHotelInput">
62            <copy>
63                <from>$BookTicketOut.parameters/return/airline</from>
64                <to>$outputVar.resultType/ns2:airlineName</to>
65            </copy>
66
67        </assign>
68
69        <invoke name="InvokeHotel" partnerLink="HotelPartnerLink2"
70                operation="reserveRoom" xmlns:tns="http://ws.hotel/"
71                portType="tns:HotelReservation" inputVariable="ReserveRoomIn"
72                outputVariable="ReserveRoomOut"/>
73
74        <assign name="AssignCarInput">
75            <copy>
76                <from>$inputVar.inputType/ns2:name</from>
77                <to>$RentCarIn.parameters/parameter/name</to>
78            </copy>
79        </assign>
80        <invoke name="InvokeCar" partnerLink="CarPartnerLink3" operation="rentCar"
81                xmlns:tns="http://ws.car/" portType="tns:CarReservation"
82                inputVariable="RentCarIn" outputVariable="RentCarOut"/>
83
84        <assign name="AssignClientOutput">
85            <copy>
86                <from>$RentCarOut.parameters/return/carType</from>
87                <to>$outputVar.resultType/ns2:carType</to>
88            </copy>
89        </assign>
90
91        <reply
92            name="ClientReply" partnerLink="ClientPortEntry"
93            operation="getTravelOffer" portType="ns1:portTypeClient"
94            variable="outputVar">
95        </reply>
96    </sequence>
97</process>
```

Listing 3.2 Simplified BPEL process

Step 6: Deploy the application

Before we can deploy the application, the GlassFish Application Server must be configured correctly and be running. As a BPEL project is not directly deployable, we must add a BPEL project as a JBI (It defines an environment for plug-in components that interact using a services model based directly on Web Services Description Language) module to a Composite Application project. Then we can deploy the Composite Application project.

3.2 Consuming the Web Service

The web service client programming model for Java EE is about accessing a remote web service from a Java EE component. Remember that with web services, the client and the server are fundamentally disconnected. There are server side issues and client side issues. In other words, if you setup a web service endpoint, any client that adheres to the abstract contract of its WSDL can talk to that endpoint.

Create the Service Endpoint Interface

Now, we create a client that accesses the process that we have just deployed. A client invokes the process in the same way it invokes any web service or a method locally.

To get started with the client application, we create a new java standalone application using NetBeans tools. This application contains a main class where we call the process. Moreover, the tools add the JAX-WS APIs package which will is needed to compile the application.

NetBeans tool uses the `wsimport` command provided by the glassfish server library to generate the required artefacts from the process's WSDL file.

This command generates a group of java classes which are used by the client to access web service methods:

- `ClientPortType1.java`
- `Service1.java`
- `ObjectFactory.java`
- `SimpleProcess.java`
- `package-info.java`

Create the standalone client

Let us develop a standalone client that calls the `getTravelOffer` method of the Travel Process service (see Listing 3.3). It makes the call through a local object that acts as a client proxy to the remote service. It is called a static stub because the stub is generated before runtime by the `wsimport` tool.

```
package travelreservationclient;

/* @author Albreshne */
public class Main {
    public static void main(String[] args) {

        try { // Call Web Service Operation
            localhost.travelagent.travelagent.Service1 service = new
                        localhost.travelagent.travelagent.Service1();
            localhost.travelagent.travelagent.PortTypeClient port = service.getPort1();
            org.netbeans.xml.schema.travelagent.SimpleProcess inputType = new
                        org.netbeans.xml.schema.travelagent.SimpleProcess();

            inputType.setName("albreshne");
            inputType.setDepartDate("2009-07-17");
            inputType.setArriveDate("2009-07-12");

            org.netbeans.xml.schema.travelagent.SimpleProcess result =
                                        port.getTravelOffer(inputType);
            System.out.println("Client Name : "+result.getName());
            System.out.println("Arrive Date : "+result.getArriveDate());
            System.out.println("Depart Date : "+result.getDepartDate());

            System.out.println("Airline : "+result.getAirlineName());
            System.out.println("Flight Ticket Price : "+result.getFlightTicketPrice());

            System.out.println("Hotel : "+result.getHotelName());
            System.out.println("Room Number : "+result.getRoomNumber());
            System.out.println("Room Price: "+result.getRoomPrice());

            System.out.println("Car Type : "+result.getCarType());
            System.out.println("Car Rent Price : "+result.getCarRentPrice());
       } catch (Exception ex) { }
 }
}
```

Listing 3.3 ***Client implementation***

After having compiled successfully and made running the client application, the result should be the following:

```
Client Name : Albreshne Adhem
Arrive Date : 2009-07-12
Depart Date : 2009-07-17
Airline : A Airlines
Flight Ticket Price : 650.0
Hotel : Soleil Hotel
Room Number : 100
Room Price: 700.0
Car Type : BMW
Car Rent Price : 300.0
```

Listing 3.4 ***Client request result***

4 Conclusion

We have explored a few emerging concepts in the area of collaboration between web services, such as orchestration, choreography, business process management and workflow. The principal objective of orchestration and choreography is connecting web services together in a collaborative way in order to get the desired result. Orchestration of web services allows building dynamic and flexible processes. A set of open, standards-based protocols are available for designing and executing interactions between numerous web services. There are essential conditions which have to be satisfied in order to make orchestration and web services composition more efficient and useful. Asynchronous support is required to build reliability into a process. Moreover, strong transactional semantics and exception handling are needed to deal with internal as well as external errors. In addition, these specifications provide a set of programming constructs to describe workflow. Finally, a method allowing to correlate requests so as to build higher-level conversations must be available.

Several standards for web services orchestration have recently been developed, including BPEL4WS, WSCI and BPML. All of these standards meet the basic needs for orchestrating web services. A number of tools have already been released to support web services orchestration. But there is still need to develop additional use cases and scenarios to demonstrate the business value of web services orchestration.

Currently, research is focussing on integration and automatic composition of web services as well as their dynamic composition. Ontology-based web services composition is another interesting field where many searchers try to use ontology to describe, simulate, compose, test and verify compositions of web services.

References

[Erl06] **Erl, T.** *Service Oriented Architecture, Concepts, Technology, and Design.* s.l. : Prentice Hall, 2006.

[Pap08] **Papazoglou, Michael.** *Web Services: Principles and Technology.* s.l. : Prentice Hall, 2008.

[Pel03] **Peltz, Chris.** *Web Services Orchestration.* s.l. : Hewlett-Packard Company, 2003.

[Mil09] **Milanvoic, Nikola et Malek, Miroslaw.** *Current Solutions for Web Service composition.* s.l. : IEEE Computer Society, 2004.

Referenced Web Resources

[*OAS07*] **OASIS.** Web Services Business Process Execution Language (WSBPEL). 2007. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel [Accessed 07 20, 2009].

[*Mat09*] **Matjaz and B.** A Hands-on Introduction to BPEL. http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html [Accessed 07 12, 2009].

[*W3C102*] **W3C.** Web Services Conversation Language (WSCL) 1.0. 2002. http://www.w3.org/TR/wscl10/ [Accessed 06 28, 2009].

[*Msd09*] **msdn.** XLANG/s Language. *msdn.* http://msdn.microsoft.com/en-us/library/aa577463.aspx [Accessed 07 15, 2009].

[*Wik05*] **Wikipedia.** Business Process Modeling Notation. 2005. http://en.wikipedia.org/wiki/Business_Process_Modeling_Notation [Accessed 06 03, 2009].

[*W3C02*] **W3C.** Web Service Choreography Interface (WSCI) 1.0. 2002. http://www.w3.org/TR/wsci/ [Accessed 08 05, 2009].

[*Wik091*] **Wikipedia.** Business Process Modeling Language. http://en.wikipedia.org/wiki/Business_Process_Modeling_Language [Accessed 07 12, 2009].

[*Act091*] **Endpoints, active.** http://www.activevos.com/ [Accessed 07 28, 2009].

[*The09*] **Foundation, The Apache Software.** Apache ODE. http://ode.apache.org/ [Accessed 07 28, 2009].

[*Com09*] **Community, jBoss.** jBPM Overview. http://www.jboss.org/jbossjbpm/jbpm_overview/ [Accessed 07 28, 2009].

[*Wik09*] **Wikipedia.** Unified Modeling Language. http://en.wikipedia.org/wiki/Unified_Modeling_Language [Accessed 07 22, 2009].

[*Abd09*] **Albreshne, Abdaladhem.** http://diuf.unifr.ch/people/albreshn/ [Accessed 06 20, 2009].

[*She09*] **Barkodar, Sherry.** Developing a Simple Synchronous BPEL Process. *NetBeans.* [Accessed 03 10, 2009] http://www.netbeans.org/kb/61/soa/synchsample.html..

[*Sun09*] **Sun.** Java SE Downloads-JDK 5. http://java.sun.com/javase/downloads/index_jdk5.jsp [Accessed 06 30, 2009].

[*Net09*] **NetBeans.** NetBeans Tool. http://www.netbeans.org/ [Accessed 07 20, 2009].

[*IBM01*] **IBM.** WSFL-Guide. 2001. http://xml.coverpages.org/WSFL-Guide-200110.pdf [Accessed 07 01, 2009].