# Using BPMN to Model a BPEL Process

**Stephen A. White, IBM Corp., United States**

## ABSTRACT

The Business Process Modeling Notation (BPMN) has been developed to enable business user to develop readily understandable graphical representations of business processes. BPMN is also supported with appropriate graphical object properties that will enable the generation of executable BPEL. Thus, BPMN creates a standardized bridge for the gap between the business process design and process implementation. This paper presents a simple, yet instructive example of how a BPMN diagram can be used to generate a BPEL process.

## INTRODUCTION

The Business Process Modeling Notation (BPMN) has been developed to enable business user to develop readily understandable graphical representations of business processes. BPMN is also supported with appropriate graphical object properties that will enable the generation of executable BPEL. This paper presents an example of an order shipping process, yet instructive example of how a BPMN diagram can be used to generate a BPEL process.

When mapping a BPMN diagram to BPEL (version 1.1)[1], a decision must be made as to the basic structure of the BPEL document. That is, will the BPEL format be based on the BPEL graph structure (the flow element) or the BPEL block structure (the sequence element)? This choice will affect how much of the BPMN Sequence Flow will map to BPEL link elements. Using a block structure as the foundation for mapping, link elements only are used when there is a specific section of the Process where parallel activities occur. Using the graph structure as the foundation for mapping, most Sequence Flow will map to link elements, as the entire BPEL process is contained within a flow element. Since the BPMN 1.0 specification[2] takes the approach of defining the mapping of BPMN elements to BPEL elements mainly through the use of block structures, this paper will take the approach of using the graph structure for the mapping.

## THE EXAMPLE: TRAVEL BOOKING PROCESS

The example that will be used in this paper is a basic version of a travel booking process. This example will illustrate a few situations that occur within BPMN diagrams and how they map to BPEL, such as parallel flow, and loops. Figure 1 shows the original representation of the BPEL Process.[3]

[1] http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/

[2] http://www.bpmi.org/bpmn-spec.htm

[3] http://publib.boulder.ibm.com/infocenter/adiehelp/index.jsp?topic=/com.ibm.etools.ctc.bpel.doc/samples/travelbooking/travelBooking.html

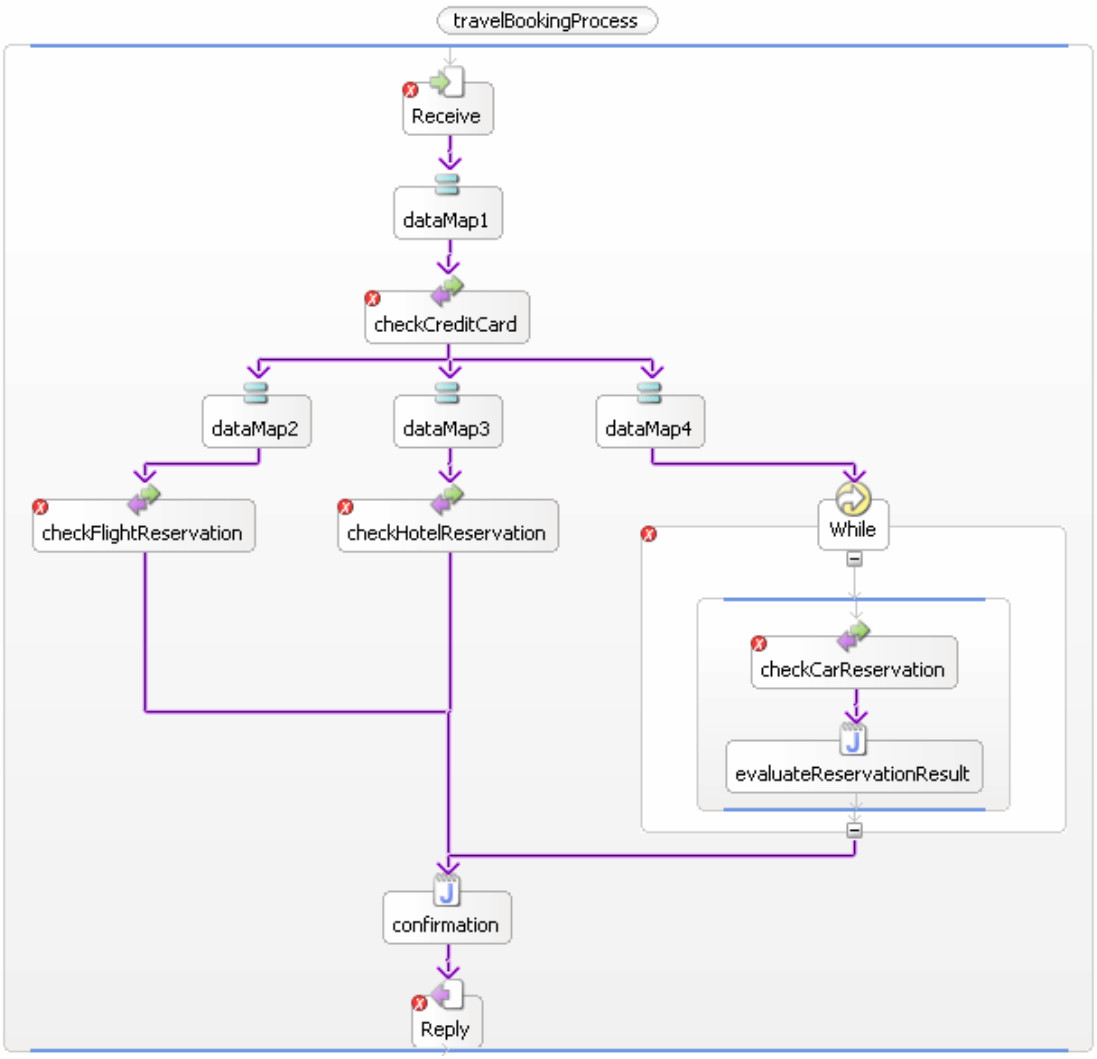


***Figure 1. Travel Booking Process with WebSphere Studio***

Figure 2 shows how the same process can be modeled with BPMN. It could be noted that Figure 2 shows the process model laid out in a horizontal direction, going left to right, while the original diagram in Figure 1 shows the process laid out in a vertical direction, going top to bottom. Process tools that create strictly BPEL process tend to lay out diagrams in a vertical direction. Although not universal, the difference between the two modeling techniques tends to separate business analysts, who tend to opt for horizontal diagrams, and IT specialist or software developers, who tend to opt for vertical diagrams. While BPMN does require any specific directionality, most BPMN diagrams flow horizontally.

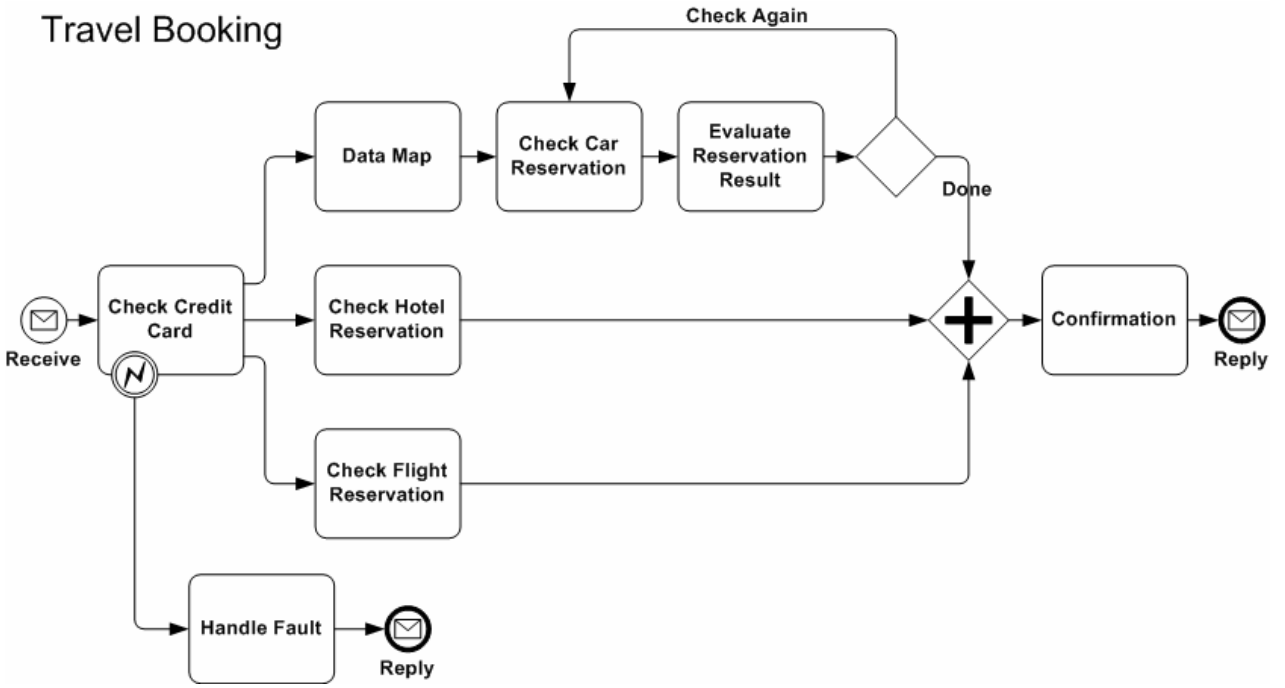


***Figure 2. Travel Booking Process with BPMN***

The Process begins with the receipt of a request for a travel booking. After a check on the credit card, reservations are made for a flight, a hotel, and a car. The car reservation may take more than one attempt before it is successful. After all three reservations are confirmed, a reply is sent.

### *Setting up the BPEL Information*

BPMN Diagrams, such as the one seen in Figure 2, can be used within many methodologies and for many purposes, from high-level descriptive modeling to detailed modeling intended for process execution. When one of the purposes of the process model is to define process execution and create the BPEL file for this purpose, then the process model will have to be developed with a modeling tool designed for this purpose. The diagram itself will not display all the information required to create a valid BPEL file. A diagram with all that information would be too cluttered to be readable. A BPMN diagram is intended to display the basic structure and flow of activities and data within a business process. Therefore, a modeling tool is necessary to capture the additional information about the the process that is necessary to create an executable BPEL file. In the sections that follow, the details behind the diagram objects will be highlighted, and it will be shown how they provide the content for the mapping to BPEL.

The modeling tool will need to define some basic types of information about the Process itself to fill in attributes of a BPEL process element of the BPEL document.

Example 1 displays how basic information collected about the BPMN Diagram and the *Travel Booking Process* within the diagram will be mapped to set up the preliminary BPEL information.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Business Process Diagram** | See next row… mapped to attributes of a process element |
| **ExpressionLanguage** = “Java” | expressionLanguage="Java" |
| **Business Process** | The process element |
| **Name** = “Travel Booking Process” | name="travelBookingProcess" |
| **ProcessType** = “Private” | abstractProcess="no" or not included |
| **SurpressJoinFailure** = “Yes” | suppressJoinFailure="yes" |

***Example 1. Mapping Basic attributes of the Business Process***

The tool that creates the BPEL file will need to define parameters, such as targetNamespace, the location of the supporting WSDL files, and other parameters, including those specific to the operating environment, to enable the BPEL file to function properly. This information will be based on the configuration of the environment for that modeling tool

The partnerLink elements are defined prior to the definition of the process, but the information about these elements will be found in the Properties of the Tasks of the BPMN Process. The Tasks of the Process that are of type Service and implemented as a Web service will define the Participant of the Web Service. The Participants and their Properties will map to the partnerLink elements.

Example 2 displays two examples of how the Properties defined for the implementation of a Web service for a Task will map to the partnerLink elements that will be defined at the head of the BPEL document.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Implementation** = "Web service" | Invoke, but some properties, below, will map to a partnerLink |
| **Participant** = "ProcessStarter" | name="ProcessStarter"<br>partnerLinkType="ProcessStarterPLT" |
| **BusinessRole** = "TravelProcessRole" | myRole="TravelProcessRole" |
| **Participant** = "HotelReservationService" | name="HotelReservationService"<br>partnerLinkType="HotelReservationServicePLT" |
| **BusinessRole** = "HotelReservationRole" | myRole="HotelReservationRole" |

***Example 2. Mapping Web Service Properties to* partnerLink**

The BPEL code for the partnerLink definitons can be seen in Example 3.

```
<partnerLinks>
   <partnerLink myRole="travelProcessRole" name="ProcessStarter" partnerLinkType="wsdl5:travelProcess"/>
   <partnerLink name="HotelReservationService" partnerLinkType="wsdl5:HotelReservationPartnerPLT"
                partnerRole="HotelReservationRole"/>
    <!-- Another 3 partnerLinks are defined -->
</partnerLinks>
```

***Example 3. Setting up the* partnerLink *Elements for the* process.**

The variable elements are also defined prior to the definition of the process. These will be created from Properties associated with a Process within a BPMN diagram. A modeling tool will allow the modeler to define these Properties. Properties of type *structure* will be used to group sets of Properties into packages that will map to the BPEL message elements. The message elements will actually be defined in a WSDL document that supports the BPEL document. The variable elements will be defined in the BPEL document and will reference the message elements.

Example 4 displays two examples of how the Properties defined for a Process will map to the variable and message elements that will be defined at the head of the BPEL document and in supporting WSDL documents.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Property** | BPEL variable and WSDL message |
| **Name** = "input" | For BPEL variable:<br>name="input" messageType="input"<br>For WSDL message:<br>name="input" |
| **Type** = "structure" | The sub-Properties of the structure will map to the WSDL message elements |
| **Property** | For WSDL message, in the part element: |
| **Name** = "airline" | name="airline " |
| **Type** = "string" | type="xsd:string" |
| **Property** | For WSDL message, in the part element: |
| **Name** = "arrival" | name="arrival" |
| **Type** = "string" | type="xsd:string" |
| Eleven more sub-Properties are included | Eleven more part elements are included |
| Ten more structure Properties are included | Ten more BPEL variable elements and WSDL message elements will be included |

***Example 4. Mapping Process Properties to BPEL variable and message***

The BPEL code for the variable definitons can be seen in Example 5.

```
<variables>
    <variable messageType="wsdl0:input" name="input"/>
    <variable messageType="wsdl4:doCreditCardCheckingRequest" name="checkCreditCardRequest"/>
    <variable messageType="wsdl4:doCreditCardCheckingResponse" name="checkCreditCardResponse"/>
    <variable messageType="wsdl4:Exception" name="creditCardFault"/>
    <variable messageType="wsdl1:doCarReservationRequest" name="carReservationRequest"/>
    <!-- Another 6  variables are defined -->
</variables>
```

***Example 5. Setting up the variable Elements for the process.***

The WSDL code for the message definitons can be seen in Example 6.

```
<message name="input">
    <part name="airline" type="xsd:string"/>
    <part name="arrival" type="xsd:string"/>
    <part name="departure" type="xsd:string"/>
    <!-- Another 10 parts are defined -->
</message>
<message name="doFlightReservationRequest">
    <part name="airline" type="xsd:string"/>
    <part name="arrival" type="xsd:string"/>
    <part name="departure" type="xsd:string"/>
    <!-- Another four parts are defined -->
</message>
<!-- Another nine messages are defined -->
```

***Example 6. Setting up the message Elements for the WSDL Document***

All the Sequence Flow that are seen in Figure 2, except four, will map to BPEL link elements. In addition, the process will require three link elements that are not represented by Sequence Flow within Figure 2. These exceptions will be explained below. When the flow element for the process is defined, the link definitions will precede the definitions of the process activities. The BPEL code for

the link definitions can be seen in Example 7. The name for each link will be automatically generated by the modeling tool.

```
<flow name="Flow" wpc:id="1"/>
    <links>
        <link name="link1"/>
        <link name="link2"/>
        <link name="link3"/>
        <link name="link4"/>
        <link name="link5"/>
        <link name="link6"/>
        <link name="link7"/>
        <link name="link8"/>
        <link name="link9"/>
        <link name="link10"/>
        <link name="link11"/>
        <link name="link12"/>
    </links>
    <!-- The Main Process will be place here -->
</flow>
```

***Example 7. Setting up the link Elements for the flow.***

### *The Start of the Process*

The Process is started with the receipt of a message request for the booking of a travel itinary through a Message Start Event (see Figure 3). After the request has been received, a check for the validity of the submitted Credit Card information is performed. As can be seen above in Figure 2, the *Check Credit Card* Task has an Error Intermediate Event attached to its boundary. This Event will be used for handling an incorrect credit card number. The mapping for this type of fault handling will be shown in the section entitled "***Error (Fault) Handling***" below.

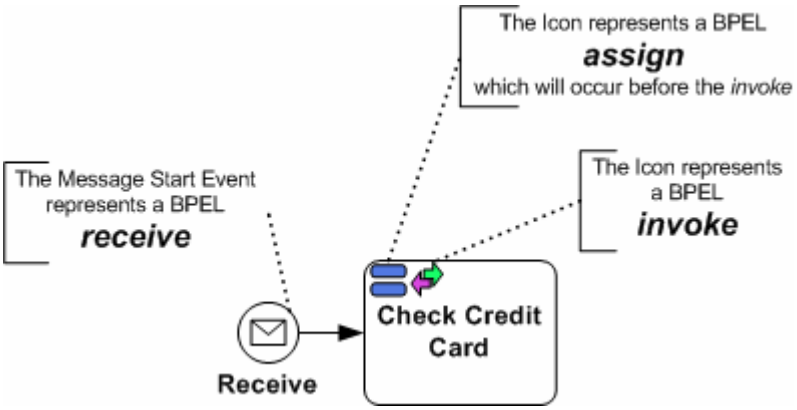


***Figure 3. The Beginning of the Travel Booking Process***

The *Receive* Message Start Event is the mechanism that initiates the Process through the receipt of a message. This will map to a BPEL receive element.

Example 8 displays the properties of the *Receive* Message Start Event and how these properties map to the attributes of the receive element.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Start Event** (EventType: Message) | receive |
| **Name** = "Receive" | name="Receive" |
| **Instantiate** = "True" | createInstance="yes" |
| **Message** = "input" | variable="input" |
| **Implementation** = "Web service" | See next three rows… |
| **Participant** = "ProcessStarter" | partnerLink="ProcessStarter" |
| **Interface** = "travelPort" | portType="wsdl0:travelPort" |
| **Operation** = "book" | operation="book" |

***Example 8. Mapping of the Message Start Event***

Example 9 displays the resulting BPEL code that will be generated for the *Receive* Message Start Event.

```
<receive createInstance="yes" operation="book" name="Receive" wpc:displayName="Receive"
         portType="wsdl0:travelPort" variable="input" wpc:id="2">
    <source linkName="link1" />
</receive>
```

***Example 9. BPEL code for the "Receive" Start Event***

Note that the *Check Credit Card* Task in Figure 3 contains small icons in the upper right corner of its shape. These icons are used to provide a graphical indication as to the type of Task that will be performed. Figure 3 and the figures below will display these type of icons. The icons are not part of the standard BPMN notation, but are part of extensibility of BPMN. It is expected that process modeling tools will utilize such icons as fit the capabilities and expected usage of the tool. The location of the icons within the Task shape is at the discretion of the modeler or modeling tool. For the purposes of this paper, these icons will aid in showing how the diagram maps to BPEL.

The *Check Credit Card* Task follows the Start Event through a connection of a Sequence Flow. The Sequence Flow indicates a dependency relationship between the BPEL elements mapped from the Start Event and then the Task. Since all the BPEL activities are contained within a flow, the dependency will take the form of a BPEL link element ("link1") seen in Example 7). The link element will be connected to the activities with the inclusion of a source element added to the receive element (Example 9) and a target element added to the first element mapped from *Check Credit Card* Task (an assign activity in Example 11 below).

The "*Check Credit Card*" Task will map to a BPEL invoke element. But this Task, as seen in Figure 3, has two icons in the upper right corner. The first icon (two blue bars stacked) indicates that a data mapping is required for the main Task to be performed. Some of the data that has been received from the input to the Process (the Message Start Event) will be mapped to the data structure of the message for the credit card checking service. As can be seen in Figure 1, the original process included a separate activity each time that a data mapping was required. However, many business process methodologies do not consider such data mapping as a separate Task that would warrant its own shape and space on the diagram. Such data mapping is usually included as pre- or post-activity functions for a business Task, although it may be used as a stand-alone Task in some situations, as we shall see later in the process. As a group, this data mapping will be defined as properties of the Task and will be mapped to a BPEL assign element. The individual property mappings will be combined to be sets of copy elements within the assign element.

The second icon (purple and green arrows pointing in different directions) indicates that the main function of the Task will be a Service type of Task, implemented through a Web servce, which maps to a BPEL invoke element.

Example 10 displays the properties of the "*Check Credit Card*" Task and how these properties map to the attributes of the assign and the invoke.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Task** (TaskType: Service) | invoke |
| **Name** = "Check Credit Card" | name="checkCreditCardRequest" |
| **InMessage** | inputVariable="checkCreditCardRequest" |
| **OutMessage** | outputVariable="checkCreditCardResponse" |
| **Implementation** = Web service | See next three rows… |
| **Participant** | partnerLink="creditCardCheckingService" |
| **Interface** | portType="wsdl4:creditCardCheckingServiceImpl" |
| **Operation** | operation="doCreditCardChecking" |
| **Assignment** | assign. The name attribute is automatically generated by the tool creating the BPEL document. |
| **From** = input.cardNumber | within a copy element, paired with the next row<br>from part="cardNumber" variable="input" |
| **To** = checkCreditCardRequest.<br>cardNumber | within a copy element, paired with the previous row<br>to part="cardNumber" variable="checkCreditCardRequest" |
| **AssignTime** = Start | This means that the assign element will precede the invoke |
| **From** = input.cardType | within a copy element, paired the next row:<br>from part="cardType" variable="input" |
| **To** = checkCreditCardRequest.<br>cardType | within a copy element, paired the previous row<br>to part="cardType" variable="checkCreditCardRequest" |
| **AssignTime** = Start | This means that the assign element will precede the invoke. All Assignments with an AssignTime of "Start" will be combined for one assign element. |

***Example 10. Mapping of the "Check Credit Card" Task***

Example 11 displays the resulting BPEL code that will be generated for the *Check Credit Card* Task.

```
<assign name="DataMap1" wpc:displayName="DataMap1" wpc:id="20">
    <target linkName="link1"/>
    <source linkName="link2"/>
    <copy>
        <from part="cardNumber" variable="input"/>
        <to part="cardNumber" variable="checkCreditCardRequest"/>
    </copy>
    <copy>
        <from part="cardType" variable="input"/>
        <to part="cardType" variable="checkCreditCardRequest"/>
    </copy>
</assign>
<invoke inputVariable="checkCreditCardRequest" name="checkCreditCard" operation="doCreditCardChecking"
          outputVariable="checkCreditCardResponse" partnerLink="CreditCardCheckingService"
          portType="wsdl4:CreditCardCheckingServiceImpl" wpc:displayName="Check Credit Card" wpc:id="5">
    <target linkName="link2"/>
    <source linkName="link3"/>
    <source linkName="link6"/>
    <source linkName="link9"/>
</invoke>
```

***Example 11: BPEL code for the "Check Credit Card" Task***

There is a sequential relationship where the assign element must precede the invoke element as determined by the assignment properties of the Task. This relationship will result in a BPEL link

("link2") element. In this case, there is no corresponding Sequence Flow in the BPMN diagram as there was for the "link1" link.

### *Creating Parallel Flow*

After the *Check Credit Card* Task, three main activities will occur. They will involve the checking of car, hotel, and flight reservations (see Figure 4). These activities are not dependent on each other so they can be performed at the same time, in parallel. The checking of the car reservation is more complicated and will be dealt with in the next section; only a data mapping activity, which precedes the check for the car, will be seen in this section.

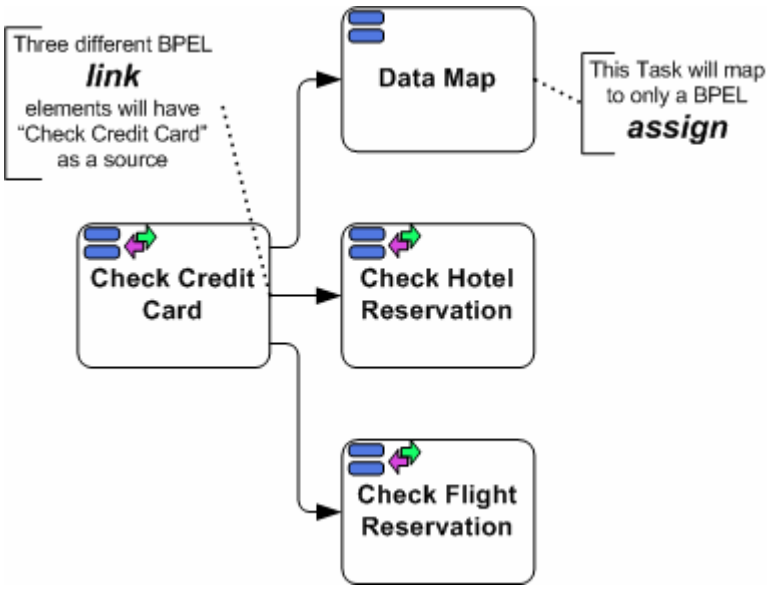


***Figure 4. Parallel Flow within the Process***

The parallelism in the Process is indicated by the three outgoing Sequence Flow from the *Check Credit Card* Task. The three target activities for these Sequence Flow are available to be performed at the same time. Each of the three Sequence Flow will result in three BPEL link elements ("link3," "link6," and "link9"). The three link elements will be included within source elements in the "checkCreditCard" invoke element (see Example 11). There will be a corresponding target element in each of the three assign elements defined below in this section (see Example 12, Example 13, and Example 15).

Starting from the bottom of Figure 4, the mapping of the *Check Flight Reservation* Task and its Propeties is very similar to the mapping of the *Check Credit Card* Task (see Example 10). In this case too, the mapping results in an assign element that will precede an invoke element. A link element ("link4") that does not have a correspoding Sequence Flow must be added to create the sequential dependency between the assign and invoke.

Example 12 displays the resulting BPEL code that will be generated for the *Check Flight Reservations* Task.

```
<assign name="DataMap2" wpc:displayName="DataMap2" wpc:id="21">
    <target linkName="link3"/>
    <source linkName="link4"/>
    <copy>
        <from part="airline" variable="input"/>
        <to part="airline" variable="flightReservationRequest"/>
    </copy>
    <!-- Six additional copy elements are not shown -->
</assign>
<invoke inputVariable="flightReservationRequest" name="checkFlightReservation"
          operation="doFlightReservation" outputVariable="flightReservationResponse"
          partnerLink="FlightReservationService" portType="wsdl3:FlightReservationServiceImpl"
          wpc:displayName="Check Flight Reservation" wpc:id="10">
    <target linkName="link4"/>
    <source linkName="link5"/>
</invoke>
```

***Example 12. BPEL code for the "Check Flight Reservations" Task***

Moving up the stack of Tasks in Figure 4, the mapping of the *Check Hotel Reservation* Task and its Properties is also very similar to the mapping of the *Check Flight Reservation* Task in that an assign element will also precede an invoke element. Again, a link element ("link7") must also be created to create the sequential dependency between the assign and invoke.

Example 13 displays the resulting BPEL code that will be generated for the *Check Hotel Reservations* Task.

```
<assign name="DataMap3" wpc:displayName="DataMap3" wpc:id="22">
    <target linkName="link6"/>
    <source linkName="link7"/>
    <copy>
        <from part="hotelCompany" variable="input"/>
        <to part="name" variable="hotelReservationRequest"/>
    </copy>
    <!-- Six additional copy elements are not shown -->
</assign>
<invoke inputVariable="hotelReservationRequest" name="checkHotelReservation"
          operation="doHotelReservation" outputVariable="hotelReservationResponse"
          partnerLink="HotelReservationService" portType="wsdl2:HotelReservationServiceImpl"
          wpc:displayName="Check Hotel Reservation" wpc:id="9">
    <target linkName="link7"/>
    <source linkName="link8"/>
</invoke>
```

***Example 13. BPEL code for the "Check Hotel Reservations" Task***

The Task at the top of Figure 4 is there to prepare data for the *Check Car Reservation* Task (see Figure 5). The other Tasks in Figure 4 have hidden the data mapping, as indicated by the icon in the upper right of the shape. This is not possible for the car reservation, since the Task for checking the reservation is included within a loop. The data mapping is only needed once, while the checking of the reservation may happen many times. Thus, the data mapping is placed in a separate Task that does nothing except the data mapping.

Example 14 displays the properties of the *Data Map* Task and how these properties map to the attributes of the assign element.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Task** (TaskType: None) | None, but assignment properties will create a mapping to an assign. Otherwise a BPEL empty element would have been created. |
| **Name** = "Data Map" | None. |
| **Assignment** | assign. The name attribute is automatically generated by the tool creating the BPEL document. |
| **From** = input.carCompany | within a copy element, paired the next row<br>from part="carCompany " variable="input" |
| **To** = carReservationRequest.<br>company | within a copy element, paired the previous row<br>to part="company" variable="carReservationRequest" |
| **AssignTime** = Start | This doesn't have any direct effect since the Task is of type "None." All Assignments with an AssignTime of "Start" will be combined for one assign element. |
| There are four other From/To Assignments that are not shown | These will map to additional from and to elements within a copy. |

***Example 14. Mapping of the "Data Map" Task***

Example 15 displays the resulting BPEL code that will be generated for the *Data Map* Task.

```
<assign name="DataMap4" wpc:displayName="Data Map" wpc:id="23">
    <target linkName="link9"/>
    <source linkName="link10"/>
    <copy>
        <from part="carCompany" variable="input"/>
        <to part="company" variable="carReservationRequest"/>
    </copy>
    <!-- Four additional copy elements are not shown -->
</assign>
```

***Example 15. BPEL code for the "Data Map" Task***

### *Mapping a Loop*

A loop occurs in the part of the Process where the car reservation is checked and then evaluated (see Figure 5).

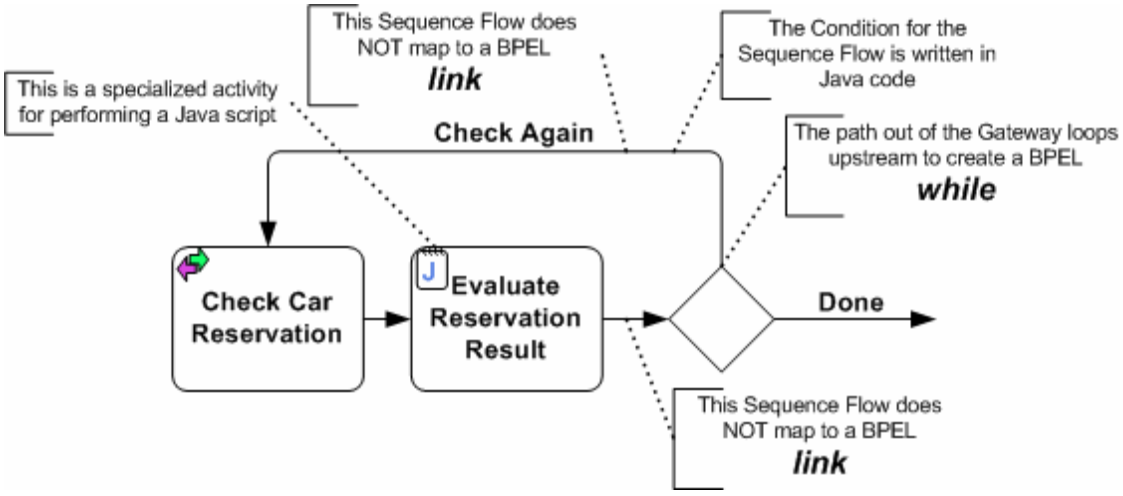


***Figure 5. A Loop within the Process***

The two Tasks can be done mutliple times if the evaluation determines that the reservation does not meet the specified critera. The loop is constructed by a decision Gateway that splits the flow based on the evaluation results. The *Check Again* Sequence Flow starts at the Gateway and then connects to an *upstream* object, creating the loop. It is the configuration of this section of the Process, its connections throght Sequence Flow, which determines how it is mapped to BPEL, rather than the strict dissection of the objects and mapping them, as has been shown above.

For Gateways that are not involved in a loop – and this Process does not have such as example – the mapping to BPEL will depend on whether the mapping is based on a graph structure (flow) or block structure (sequence).

For a block structure mapping, the Gateway would be mapped to a switch. Each of the outgoing Sequence Flow from the Gateway would map to a case element within the switch and the Expression for the Sequence Flow would map to the condition of the case.

For a graph structure mapping, each of the outgoing Sequence Flow from the Gateway will map to separate link elements (the incoming Sequence Flow to the Gateway does not map to a BPEL element). The Condition for each of these Sequence Flow will be the transitionCondition of the source element in the activity that precedes the Gateway. In this case, it would have been the mapped invoke from *Evaluate Reservation Result* Task.

However, since a loop is created by the Sequence Flow from the Gateway and, due to the acyclical nature of the flow, link elements cannot be used in a target element that is in an upstream activity within a flow, this means that a while element must be created to handle the loop. The contents of the while will be determined by the boundaries set by the Gateway and the target activity that is upstream from the Gateway. As can be seen in Figure 5, the *Check Car Reservation* and *Evaluated Reservation Result* Tasks are within the loop and will map to the contents of the BPEL while. Consistent with the decision for mapping the whole Process, the contents of the while will be mapped to the graph structured elements. This means that the main element of the while will be a flow, and the BPMN Task mappings will fit with that flow.

The BPMN *Check Again* Sequence Flow, which connects to the upstream *Check Car Reservation* Task, has a branching Condition. This Condition would typically map to as the condition attribute for the while. In this case, however, the condition is written with the Java programming language, and an extension in the form of a condition element is used to hold the Java code.

Example 16 displays the resulting BPEL code that will be generated for the Loop from the Gateway.

```
<while condition="DefinedByJavaCode" name="While" wpc:id="11">
    <wpc:condition>
        <wpc:javaCode><![CDATA[
            boolean condition = false;
            try {
                if (getCarReservationResponse().getBooleanPart("result")) {
                    condition = false;
                } else {
                    condition = true;
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
            return condition;
                                    ]]>
        </wpc:javaCode>
    </wpc:condition>
    <target linkName="link10"/>
    <source linkName="link11"/>
    <flow wpc:id="16">
        <links>
            <link name="link13"/>
        </links>
     <!-- Two invoke elements are place here, and shown below -->
    </flow>
</while>
```

***Example 16. BPEL code for the Loop from the Gateway***

The *Check Car Reservation* Task and its Properties have a straightfoward mapping to an Invoke element. Example 17 displays the resulting BPEL code that will be generated for the *Check Car Reservation*s Task.

```
<invoke inputVariable="carReservationRequest" name="checkCarReservation" operation="doCarReservation"
        outputVariable="carReservationResponse" partnerLink="CarReservationService"
        portType="wsdl1:CarReservationServiceImpl" wpc:displayName="Check Car Reservation" wpc:id="13">
    <source linkName="link13"/>
</invoke>
```

***Example 17. BPEL code for the "Check Car Reservation" Task***

The *Evaluate Reservation Result* Task is different from the previous Tasks in this Process. For BPMN, it is a Task of type *Script*. This means that when the Task is reached in the Process, a service will not be called, but the engine that is executing the Process will perform a script that has been defined for the Task. In this case, the script is written in the Java programming language. The script will check the results of the three reservation checks (flight, hotel, and car) and determine whether all three were successful or whether the trip could not be booked as planned.

To enable the performance of the script, the BPEL invoke activity will be extended to hold the Java code that the Process engine will perform. The extension will be the addition of a script element that contains a javaCode element that holds the script code.

Example 18 displays the properties of the *Evaluate Reservation Result* Task and how these properties map to the attributes of the invoke element.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **Task** (TaskType: Script) | Invoke<br>Since the TaskType is "Script" these attributes are automatically set:<br>partnerLink="null"<br>portType="wpc:null"<br>operation="null"<br>inputVariable is not used<br>outputVariable is not used |
| **Name** = "Evaluate Reservation Result" | Name="evaluateReservationRequest" |
| **Script** = [Java Script] | The Invoke element is extended to add the wpc:script element, which contains a wpc:javaCode element. The Java code is included within the wpc:javaCode element. |

***Example 18. Mapping of the "Evaluate Reservation Result" Task***

Example 19 displays the resulting BPEL code that will be generated for the *Evaluate Reservation Result* Task.

```
<invoke name="evaluateReservationRequest" operation="null" partnerLink="null" portType="wpc:null"
        wpc:displayName="Evaluate Reservation Request" wpc:id="14">
    <wpc:script>
        <wpc:javaCode><![CDATA[
            <!-- Java Code Inserted here -->
                                    ]]>
        </wpc:javaCode>
    </wpc:script>
    <target linkName="link13"/>
</invoke>
```

***Example 19. BPEL code for the "Evaluate Reservation Result" Task***

In Figure 5, there is a Sequence Flow between the *Check Car Reservation* Task and the *Evaluate Reservation Result* Task. This Sequence Flow will map to a link element ("link13"), which will be named in the source element of the "carReservationRequest" invoke and the target element in the "evaluateReservationRequest" invoke. The Sequence Flow between the *Evaluate Reservation Result* Task and the Gateway marks the end of the loop, thus the end of the while. Therefore, a link element is not required.

### *Synchronizing Parallel Flow*

The Process has three parallel paths that follow the *Check Credit Card* Task. These three paths converge and are synchronized before the *Confirmation* Task, as represented by the Parallel Gateway (see Figure 6). This synchronization means that all three paths must complete at that point before the flow of the Process can continue.

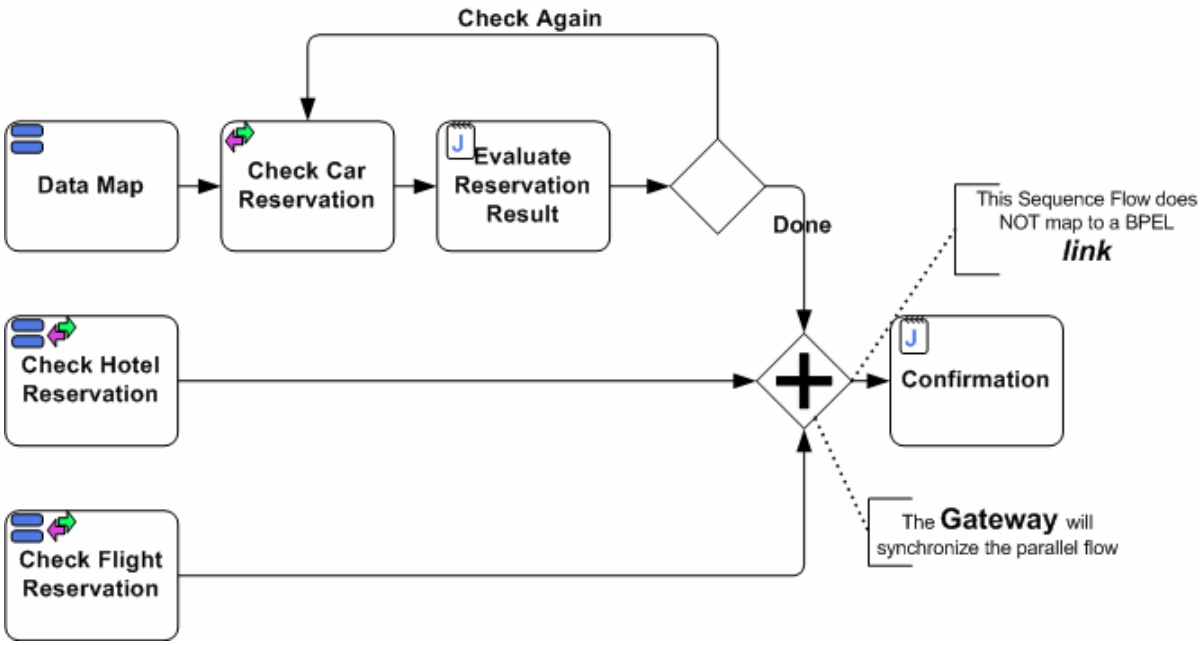


***Figure 6. Flow Synchronization within the Process***

The *Confirmation* Task is another Task that defines a script written in Java. Thus, the mapping to BPEL will be similar to the mapping of the *Evaluate Reservation Result* Task and its Properties (see Example 18).

Example 20 displays the resulting BPEL code that will be generated for the *Confirmation* Task.

```
<invoke name="Confirmation" operation="null" partnerLink="null" portType="wpc:null"
        wpc:displayName="Confirmation" wpc:id="12">
    <wpc:script>
        <wpc:javaCode><![CDATA[
            <!-- Java Code Inserted here -->
                                    ]]>
    </wpc:javaCode>
    </wpc:script>
    <target linkName="link5"/>
    <target linkName="link8"/>
    <target linkName="link11"/>
    <source linkName="link12"/>
</invoke>
```

***Example 20. BPEL code for the "Confirmation" Task***

Within the BPEL code for the process, the actual synchronization of the paths will occur in the "Confirmation" invoke. There is not a separate synchronization element, such as the Parallel Gateway in BPMN. BPEL uses the link elements within a flow to create dependencies, including synchronization, between activities. Because the "Confirmation" invoke has three target elements (for "link9," "link10," and "link11"—see Example 20), this invoke must wait until it receives a signal from all three link elements before it can be performed. The source elements for these elements are within the "flightReservationRequest" invoke, the "hotelReservationRequest" invoke, and the while activity, respectively.

The lack of a joinCondition for the "Confirmation" invoke means that there must be at least one positive signal, but the fact that there must be all three signals, positive or negative, requires that all three preceding activities must have been completed, thus synchronizing the flow.

It should be noted that the *Done* Sequence Flow from the Gateway, which maps to the "link12" link element, has a Condition which is used for the branching from the Gateway. Thus, the source element that names "link12" could have a transitionCondition defined. This really is not needed, however, since the "link12" link will not be triggered until the while, its source, has completed. This

means that for any transitionCondition for that link will always be true when it is triggered. If there had been another outgoing Sequence Flow from the Gateway, then the Conditions for the Sequence Flow would have an effect on the flow of the Process.

### *The End of the Flow*

After the *Confirmation* Task, the Process ends with a reply being sent back to the initiator of the Process (see Figure 7). The reply is bundled into a Message End Event.

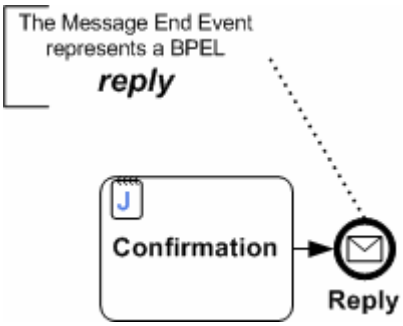


***Figure 7. The Conclusion of the Process***

The Message End Event will map to a reply element. Also, the Sequence Flow from the *Confirmation* Task to the *Reply* End Event will map to a link element ("link12"), with the source and target naming the link in the respective BPEL activities.

Example 21 displays the properties of the *Reply* Message Ene Event and how these properties map to the attributes of the reply element.

| BPMN Object/Attribute | BPEL Element/Attribute |
|---|---|
| **End Event** (EventType: Message) | reply |
| **Name** = "Reply" | name="Reply" |
| **Message** = "output" | variable="output" |
| **Implementation** = "Web service" | See next three rows… |
| **Participant** = "ProcessStarter" | partnerLink="ProcessStarter" |
| **Interface** = "travelPort" | portType="wsdl0:travelPort" |
| **Operation** = "book" | operation="book" |

***Example 21. Mapping of the "Reply" End Event***

Example 22 displays the resulting BPEL code that will be generated for the *Reply* Message End Event.

```
<reply name="Reply" operation="book" wpc:displayName="Reply" partnerLink="ProcessStarter"
        portType="wsdl0:travelPort" variable="output" wpc:id="3">
    <target linkName="Link12"/>
</reply>
```

***Example 22. BPEL code for the "Reply" Message End Event***

### *Error (Fault) Handling*

As can be seen in Figure 8, the *Check Credit Card* Task has an Error Intermediate Event attached to its boundary. The Intermediate Event will react to a specific error (fault) trigger, interrupt the Task, and direct the flow to its outgoing Sequence Flow. The error will be triggered if the entered credit card number is invalid.

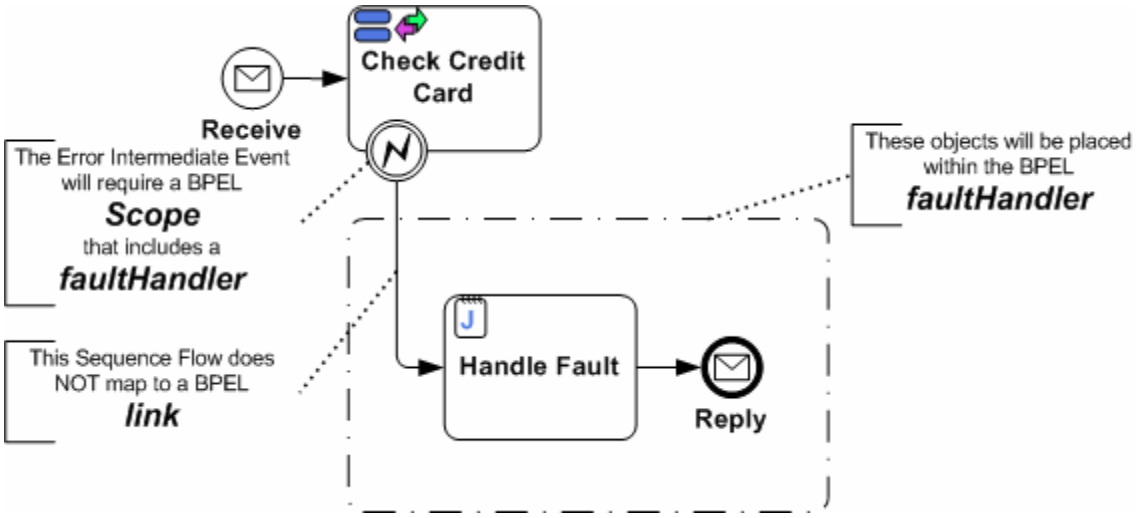


***Figure 8. Fault Handling for the Process***

If the error occurs, then the *Handle Fault* Task will process the information and prepare the message that will be sent back. The *Reply* Message End Event will send the message. This means that the Process will end and all the other activities in the Process will not occur. Thus, the Intermediate Event, as it leads to an End Event without merging back into the rest of the Process, interrupts the entire Process. The BPEL mechanism for interrupting the Process will be a faultHandlers element within a scope. This scope will envelope the entire contents of the process, which means that the main process flow (see Example 7) will actually be contained within the scope – along with the faultHandlers. The flow will run within the scope until it completes normally, unless the faultHandlers is triggered, thereby interrupting the flow.

The contents of the faultHandlers will be the activities that will be performed if the is faultHandlers triggered. This means that the *Handle Fault* Task and the *Reply* Message End Event mappings will be placed within the faultHandlers. The *Handle Fault* Task is a script Task and its mapping to BPEL will be similar to that of the *Evaluate Reservation Result* Task (see Example 18). The *Reply* End Event will be mapped the same way as the *Reply* End Event of the main part of the Process (see Example 22).

Consitent with the way that the main Process and the loop were mapped to BPEL, the fault handling section will be placed within a flow (within the faultHandlers). The Sequence Flow from the Error Intermediate Event to the *Handle Fault* Task will not be reflected with a link element since the mapping of the *Handle Fault* Task will be the first activity within the flow of the faultHandlers. However, the Sequence Flow from the *Handle Fault* Task to the *Reply* End Event will map to a link ("link14") with the source and target naming the link in the respective BPEL activities.

Example 23 displays the resulting BPEL code that will be generated for the Process Fault Handling, including the code for the *Handle Fault* Task and the *Reply* End Event.

```
<scope wpc:id="15">
    <faultHandlers>
        <catch faultName="wsdl4:Exception" faultVariable="creditCardFault">
            <flow wpc:id="17">
                <links>
                    <link name="Link14"/>
                </links>
                <invoke name="HandleFault" operation="null" wpc:displayName="handleFault"
                        partnerLink="null" portType="wpc:null" wpc:id="18">
                    <wpc:script>
                        <wpc:javaCode><![CDATA[
                            <!-- Java Code Inserted here -->
                                                    ]]>
                        </wpc:javaCode>
                    </wpc:script>
                    <source linkName="Link14"/>
                </invoke>
                <reply name="Reply" operation="book" wpc:displayName="Reply" partnerLink="ProcessStarter"
                       portType="wsdl0:travelPort" variable="output" wpc:id="19">
                    <target linkName="Link14"/>
                </reply>
            </flow>
        </catch>
    </faultHandlers>
    <flow wpc:id="15">
        <!-- The main process as shown above is placed here -->
    </flow>
</scope>
```

***Example 23. Mapping of the Fault Handling for the Process***

## CONCLUSION

This paper provides an example of how a BPMN Business Process Diagram can be used to represent an executable process. To create the executable process, the diagram objects and their properties are dissected and then mapped into the appropriate BPEL elements. Although this paper is not intended to cover all aspects of mapping BPMN diagrams to BPEL, it takes a step by step illustration of the specific objects of a travel booking process and their mapping to BPEL. Thus, this example shows how a BPMN diagram can serve the dual purpose of providing a business-level view of a business process and allow the generation of process executable code through BPEL.