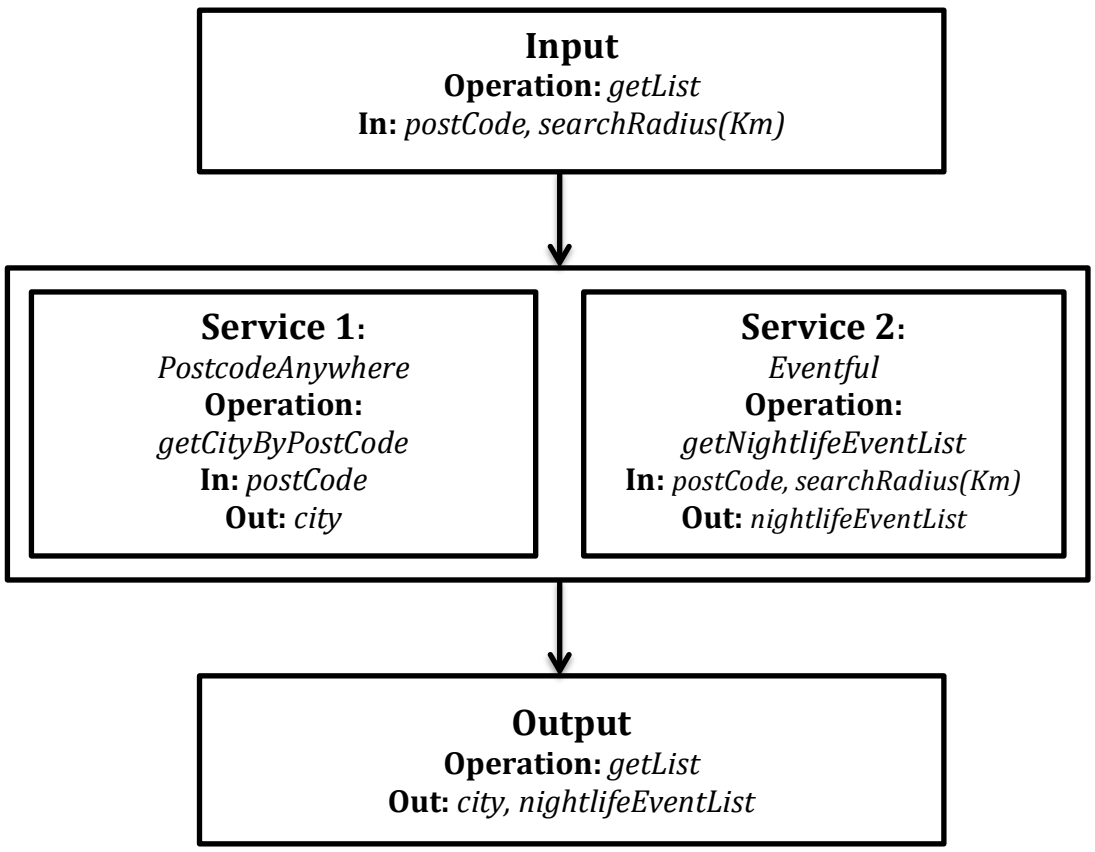
UK Nightlife Web Service

Patrik Rosini – MCSN – 499372

Index

1. Introduction

The UK Nightlife Web Service provides to a customer the list of currently week nightlife events in United Kingdom, geolocated from postcode, within a chosen search radius area (expressed in Km). The orchestration process is illustrated in the following figure:



The output of the offered web service is obtained from the orchestration of two different web services. Let us describe them in detail:

1. **PostcodeAnywhere** is a SOAP web service provided by "reallymoving.com". It exposes several operations but we used just "FindByPostcode" in the project. It returns geocoded information of a UK postcode, i.e. street address and place. In our case we used it just to retrieve the place name so the city name; WSDL URL: http://www.reallymoving.com/API/PostcodeAnywhere.asmx?WSDL
2. **Eventful** is a RESTful service provided by "Eventful.com" that offers a large collection of events i.e. nightlife, sport, music, etc.. In our project it is used to query for a list of UK nightlife events available in the current week within in a specified area. URL: http://api.eventful.com/rest/events/search?app_key=Qt5pBPBwKZrK7gdR&location=POSTCODE&within=RADIUS&units=km&category=singles_social&date=this%20week

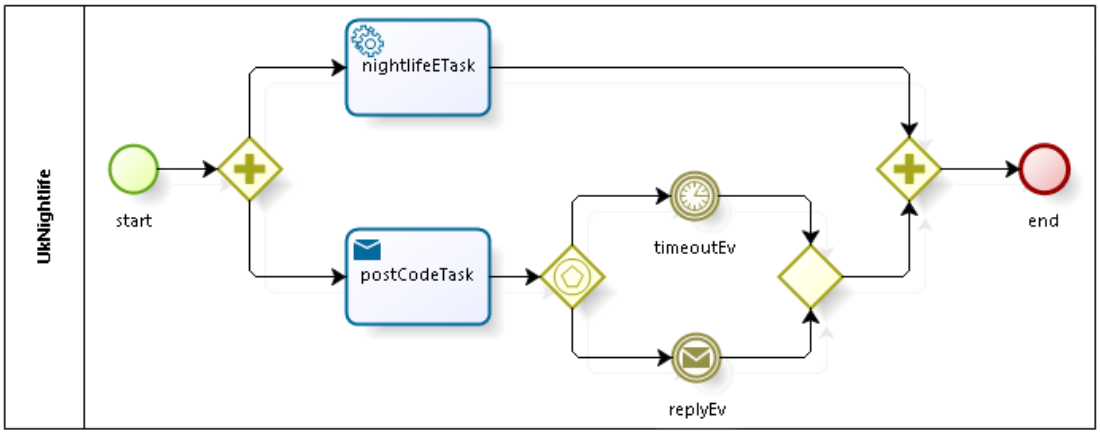
After described the adopted web services let us consider the operations invoked in the orchestration:

- **getCityByPostCode:** it is invoked by the orchestrator simultaneously to the getNightlifeEventList operation. It asks for a postcode (string) that is passed to the orchestrator as input from the customer. The extracted output consists of city name (string) that is used in the final output of the orchestration. Moreover, this operation is performed by a proxy that it has implemented to allow an asynchronous communication between the orchestrator and the PostcodeAnywhere web service;
- **getNightlifeEventList:** it is invoked concurrently to the getCityByPostCode. It requires a postcode (string) and search radius (int) as input coming from the inputs of the customer. Its output, together with the output of the getCityByPostCode, takes part of the final result to build the expected output of the whole orchestration.

As regard the operation provided by UK Nightlife Web Service it needs two inputs (postCode, searchRadius). The resulting output is formed of two strings obtained thanks to the composition of the outputs of invoked web services.

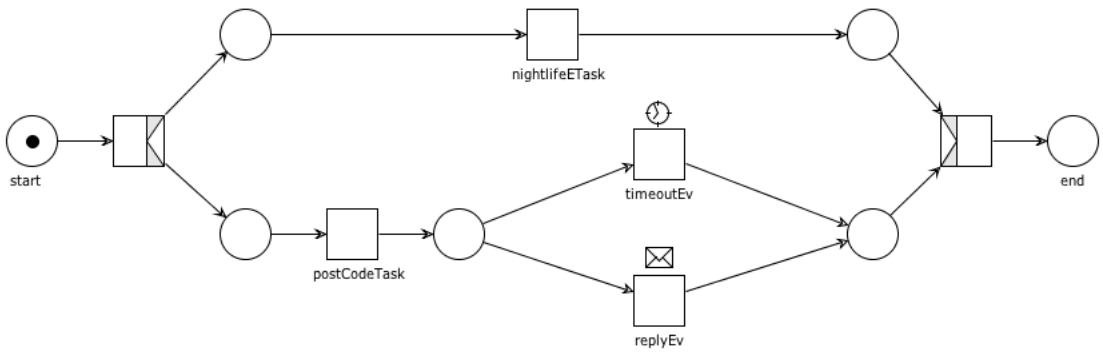
2. BPMN specification

In this section will be presented a fault-free version of the orchestration process modeled with a BPMN 2.0 diagram.
The model begins with a start event followed by a parallel gateway that it represents the parallel execution concerning the service task *nightlifeETask* modeling the synchronous invocation of the operation getNightlifeEventList and the send task *postCodeTask* modeling the asynchronous invocation of the operation getCityByPostCode. The proxy reply management is represented by an event-based gateway with a message and a timer event without others flow objects i.e. in timer event branch, since a fault-free version of the orchestration process has been designed. The termination of the orchestration process is simply modeled with an end event. The full BPMN 2.0 diagram is shown below.



3. Analysis of the BPMN specification

Let us describe the workflow net which models the control flow of the presented BPMN diagram.



The main design choice of the previous section are correctly represented namely the parallel execution is properly modeled with a pair of AND-split/AND-join and the proxy reply management where a message and a timer transition triggers are adopted.
In conclusion the obtained workflow net is free-choice, well-structured and sound.

4. WS-BPEL implementation

According with the project specification the processes implemented are:

- UKNightlife, the orchestrator process;
- GetCityByPostCodeProxy, the proxy process;
- GetCityByPostCodeLocal (WS-BPEL process), the local dummy version of PostcodAnywhere WS.

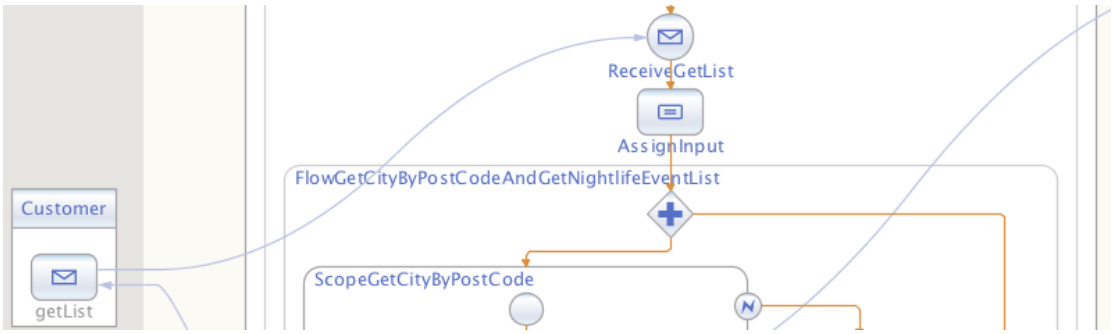
Let us concentrate to the WS-BPEL implementation of the orchestrator.

4.1 WS-BPEL process

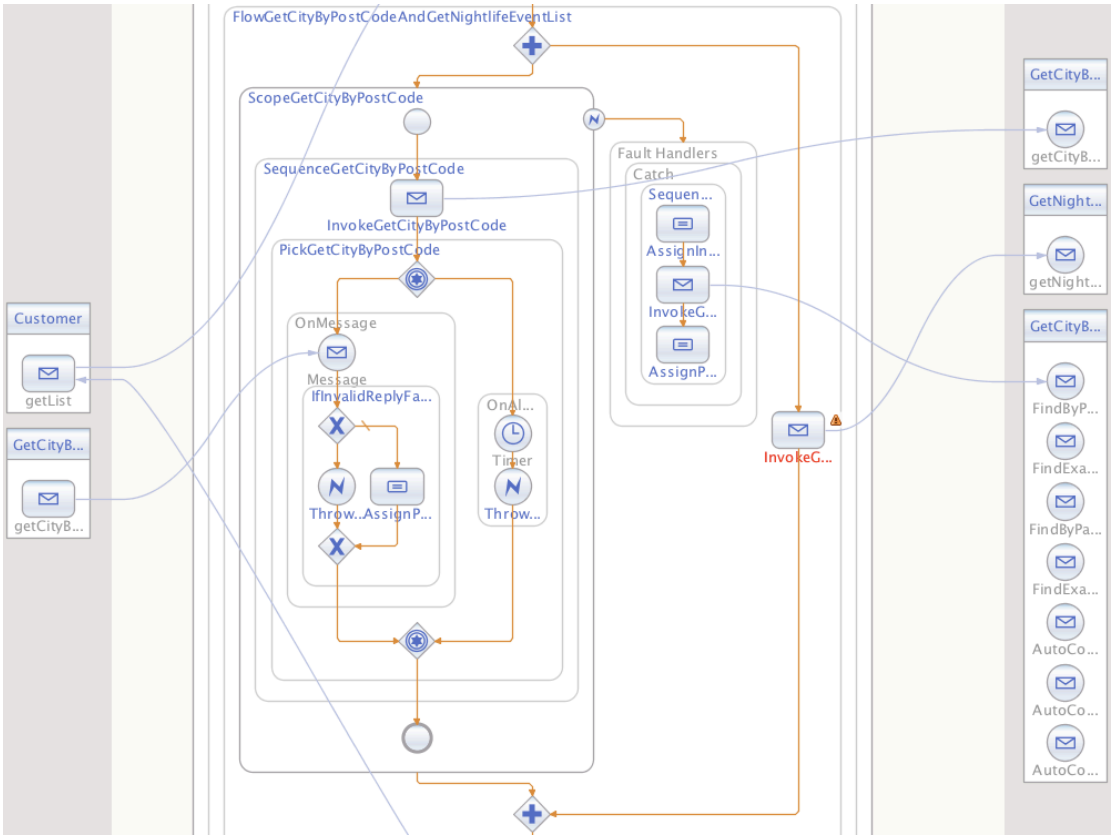
The description of the UK Nightlife SOAP service is contained in its WSDL. The following figure is a part of it just to illustrate the data that can be exchanged by incoming and outgoing messages.

```
<message name="ukNightlifeOperationRequest">
    <part name="postCode" type="xsd:string"/>
    <part name="searchRadius" type="xsd:int"/>
</message>
<message name="ukNightlifeOperationResponse">
    <part name="city" type="xsd:string"/>
    <part name="nightlifeEventList" type="xsd:string"/>
</message>
<message name="ukNightlifeOperationFault">
    <part name="ukNightlifeFault" type="xsd:string"/>
</message>
```

Concerning the BPEL implementation of the orchestration process, the first phase is about a receive activity that acts as start activity waiting an external message from the customer followed by an assign activity for the assignment of inputs received from the customer.

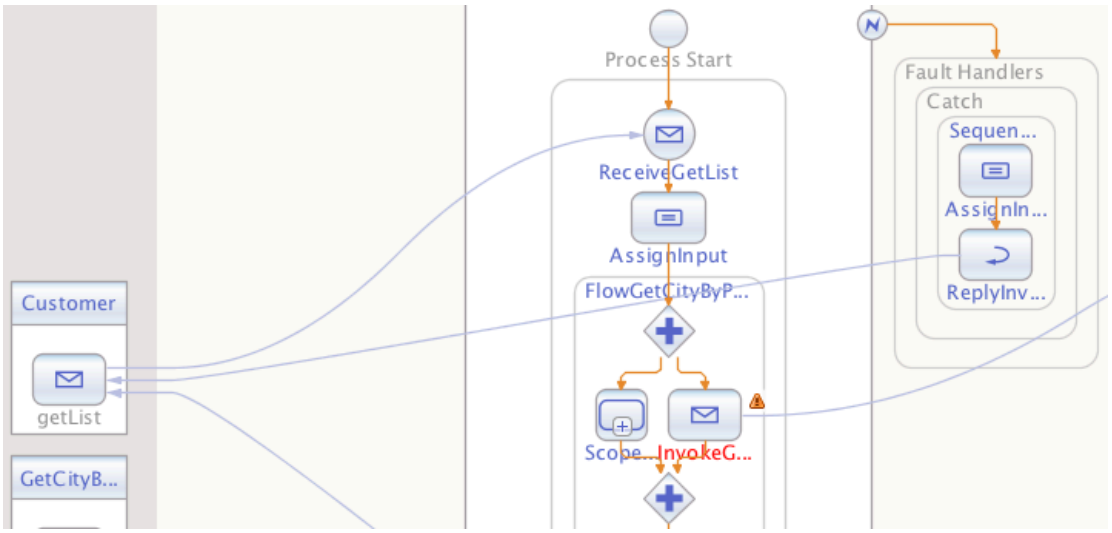


Next phase of the orchestration process is the invocation of the Postcode Anywhere and the Eventful web services in a flow activity.



As shown in the figure the right branch of flow activity it merely consists in the call of the GetNightlifeEventList partner provider. On the left branch the invoke activity calls the GetCityByPostCode partner provider that routes the calling to the GetCityByPostCodeProxy process performing an asynchronous invocation of the remote web service. In the next step the pick activity waits for the answer of proxy invocation with an if activity to check the correctness of the result and with a wait activity in the case of no response. It's considerable to note that an exception fault can be thrown in the cases of invalid reply or timeout in the waiting of answer from the proxy. To provide a specific fault handling on this

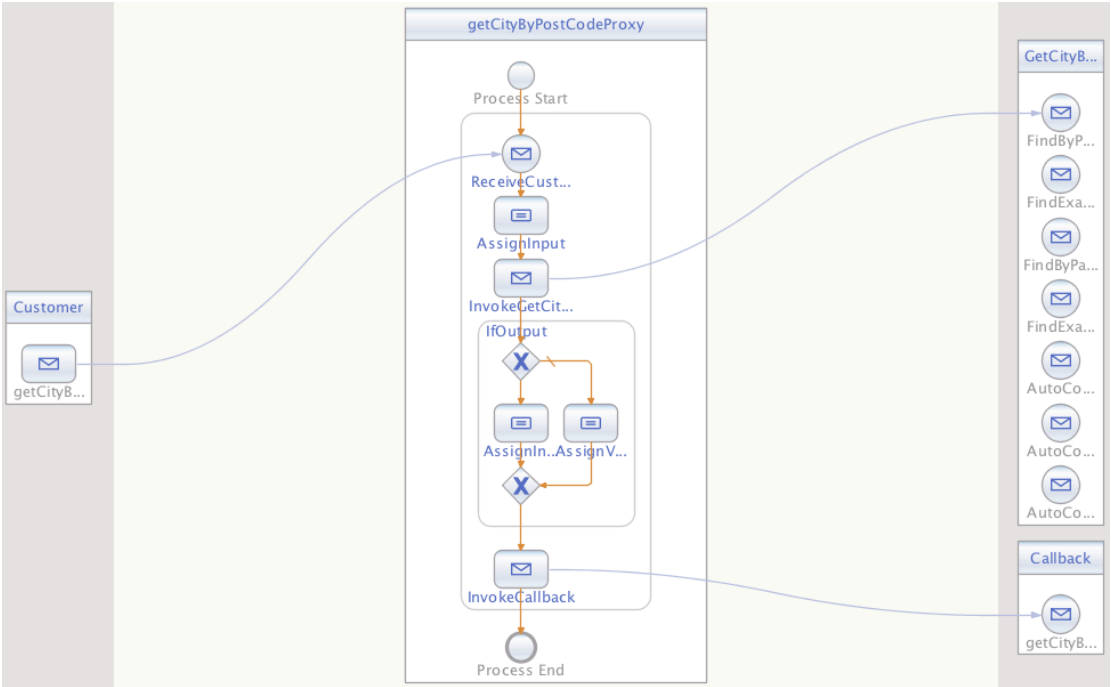
eventuality two fault handlers are adopted that is one associated to the process scope and the other one related to the inner scope.
In detail, the external fault handler is capable to handle an *invalid_reply_fault* that is thrown if the result received from the proxy is not valid instead the internal fault handler is capable to handle a *timeout_fault* that is thrown if the proxy doesn't reply within 5 seconds. In case of *invalid_reply_fault* the fault handler assigns a specific message to the ukNighlifeFault part of the fault variable. Moreover, the orchestration process is terminated and no more activities are executed so it implies that the invocation of GetNightlifeEventList partner provider is terminated if it has not been already executed and a reply is sent to the customer with a description of the occurred fault as shown in the following figure:



In case of timeout fault since it must not terminate the orchestration process, thus allowing the parallel invocation of the two web services, the fault handler consist in the synchronous invocation of the local version of GetCityByPostCode through an assign activity of an XML literal to the partner link used to interact with the GetCityByPostCodelLocal process that is the endpoint reference used to perform the dynamic binding with the endpoint of the local version of the GetCityByPostCode by reusing the same WSDL interface of the original PostcodeAnywhere web service for the GetCityByPostCodelLocal partner provider. The invocation of the GetCityByPostCodelLocal process returns a predefined value (in our case the fault description) of city name that is assigned to the city part of the output variable in the final result. The XML literal adopted is shown in the next figure:

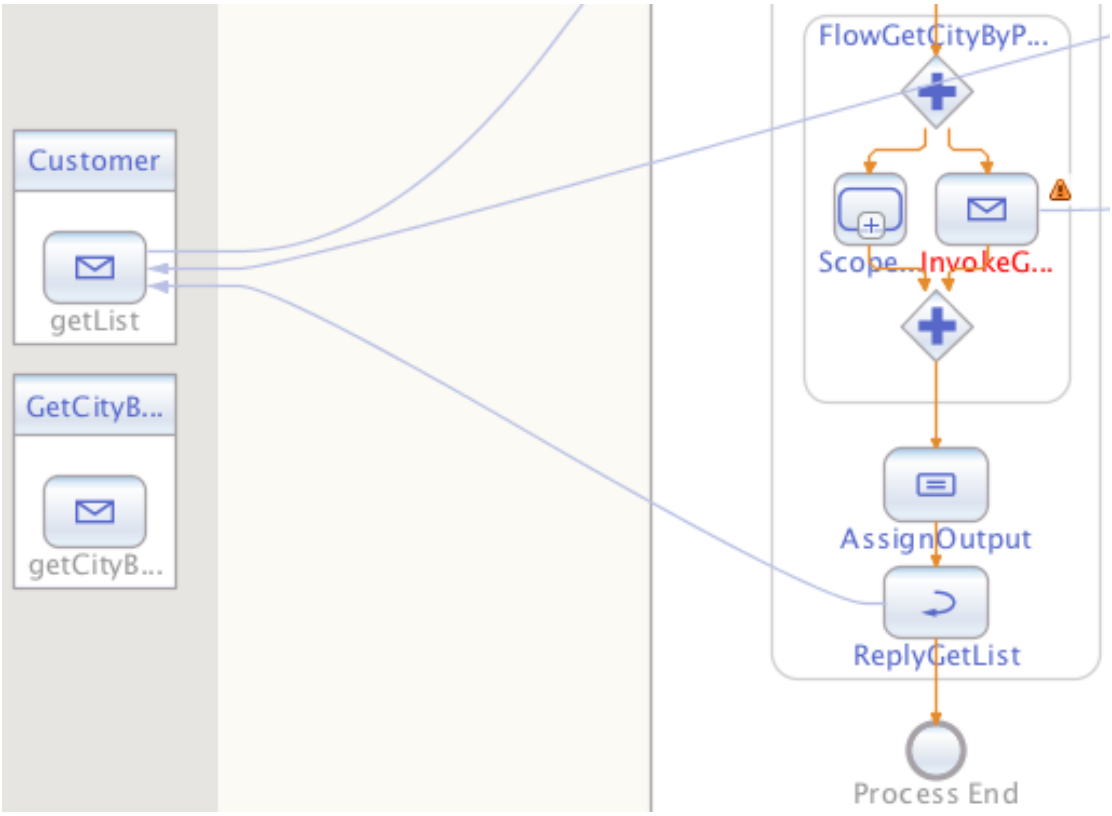
```
<literal>
  <sref:service-ref>
    <wsa:EndpointReference>
            <wsa:Address>
                http://localhost:9080/GetCityByPostCodeLocalService/GetCityB
            </wsa:Address>
            <wsa:ServiceName PortName="GetCityByPostCodeLocalPort"
                             xmlns:location="http://www.reallymoving.com/">
                location:GetCityByPostCodeLocalService
            </wsa:ServiceName>
    </wsa:EndpointReference>
  </sref:service-ref>
</literal>
```

The implementation of the GetCityByPostCodeProxy is trivial as shown in the figure. It receives as inputs from the orchestrator the postcode value and it requires an if activity to perform a first check of the response message from the remote web service and it assigns to the city part of the getCityByPostCodeCallback operation the response result or a specific value indicating an invalid response. This is required because the PostcodeAnywhere web service provides only an empty response message in case of invalid result. The correlation between the request to the proxy and the reply from the callback is obtained through the ID part variable corresponding to a BPEL Id process instance sent by the orchestrator and get back from the callback.



Continuing the analysis of the orchestration process, if no faults are thrown in the pick activity the orchestrator executes the assignment in the if activity setting a partial final output result. In the last phase

the orchestrator builds the final output result with the last assign activity using the collected results of the two web services invocations and a reply is sent to the customer as shown in the next figure so, the WS-BPEL process is concluded.



4.2 Testing

The UK Nightlife Web Service has been tested for *getList* operation and additional tests regarding the fault handling have been executed. The first test assumes a customer request from Manchester, so the input could be:

```
<soapenv:Envelope xsi:schemaLocation="http://schemas.xmlsoap
  <soapenv:Body>
    <ukn:getList>
      <postCode>M2 4JN</postCode>
      <searchRadius>10</searchRadius>
    </ukn:getList>
  </soapenv:Body>
</soapenv:Envelope>
```

The relative output taking into account of the previous input request could be:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://sc
  <SOAP-ENV:Body>
    <m:getListResponse xmlns:m="http://j2ee.netbeans.org/wsdl/UkNightlife/s
      <city xmlns:m="http://j2ee.netbeans.org/wsdl/GetCityByPostCodeProxy/s
          Manchester
      </city>
      <nightlifeEventList xmlns:msgns="http://j2ee.netbeans.org/wsdl/UkNigh
        <search>  <total_items>127</total_items>  <page_size>10</page_size>
        </search>
      </nightlifeEventList>
    </m:getListResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As shown in the figure the output of the RESTful web services (Eventful) is not well formatted since OpenESB doesn't allow to parse an XML document in the form of a result list in an effective manner.

4.2.1 invalid_reply_fault

In order to simulate an exception fault during the orchestration, to guarantee that an invalid_reply_fault is thrown it is needed to use an incorrect postcode in the input request, i.e. non-existent postcode, etc., so the input request is:

```
<soapenv:Envelope xsi:schemaLocation="http://schemas.xmlsoap
  <soapenv:Body>
    <ukn:getList>
      <postCode>CR1 1TA</postCode>
      <searchRadius>10</searchRadius>
    </ukn:getList>
  </soapenv:Body>
</soapenv:Envelope>
```

As expected, the customer receives a reply with fault description:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://sc
  <SOAP-ENV:Body>
    <SOAP-ENV:Fault>
      <faultcode>SOAP-ENV:Client</faultcode>
      <faultstring>fault1</faultstring>
      <detail>
        <ukNightlifeFault>Invalid_reply_fault</ukNightlifeFault>
      </detail>
    </SOAP-ENV:Fault>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.2.2 timeout_fault

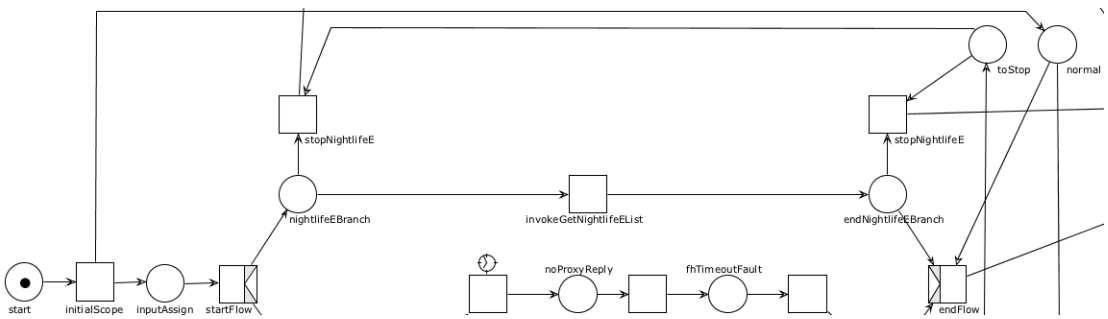
As regard of the timeout_fault, to guarantee that it is thrown, a wait activity is injected in the proxy implementation with a waiting time greater than 5 sec to artificially generate a delay in the reply from the proxy. So, assuming the first input request in paragraph 4.2 the output is shown in the following figure:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelop
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://schemas.xmlsoap.org/soap/envelope/ http://s
  <SOAP-ENV:Body>
    <m:getListResponse xmlns:m="http://j2ee.netbeans.org/wsdl/UkNightlife/
      <city xmlns:msgns="http://www.reallymoving.com/">
          Timeout_fault
      </city>
      <nightlifeEventList xmlns:msgns="http://j2ee.netbeans.org/wsdl/UkNigl
        <search>  <total_items>16</total_items>  <page_size>10</page_size>
        </search>
      </nightlifeEventList>
    </m:getListResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

As expected, also in this case the customer receives a reply with fault description.

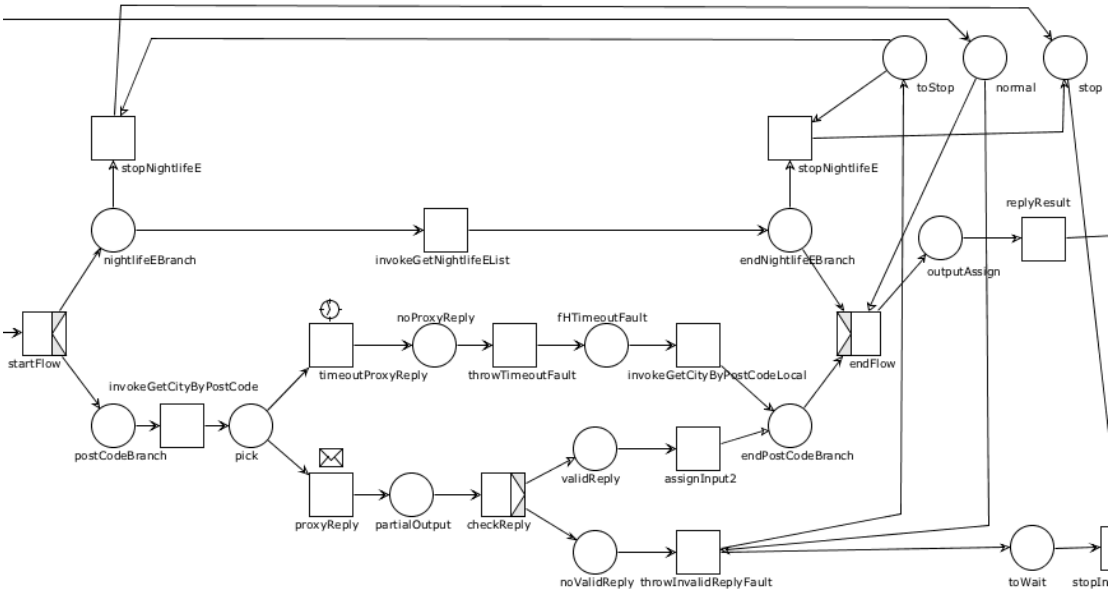
5. Analysis of the WS-BPEL specification

In this section we describe the workflow net associated to the WS_BPEL orchestration process. It could be considered as an extension of the net of section 3 with the addition of the fault handler modeling. The analysis will proceed in several steps to have focus in different parts of the net.



In this part we consider the beginning of the process with a token in *start* place followed by a transition to put a token into the *normal* place representing the normal status of execution of the whole

process. Than, we analyze the next part of the modeled net in which the process enters in the flow activity represented with an AND-split transition (*startFlow*) and an AND-join (*endFlow*) and it performs the invocation of the first two services represented with the *invokeGetCityByPostCode* transition referring to the asynchronous invocation of the proxy and the *invokeGetNightlifeEList* transition.



In the postCodeBranch when a token is in the *pick* place it represents the waiting of the reply from proxy that is managed by the transitions *timeoutProxyReply* with a time trigger and *proxyReply* with a message trigger. Here, an exception fault can be thrown depending on which transition is fired so could be the *throwTimeoutFault* or the *throwInvalidReplyFault* transition where the last is called after the check of the result received from proxy represented by the XOR-split transition (*checkReply*) followed by a token in the *noValidReply* place. The fault handling of the timeout_fault is simply represented by place *fHTimeoutFault*. In the nightlifeEBranch no faults are provided but a premature termination of it caused by an invalid_reply_fault thrown from the other branch is possible. This eventuality is managed by the *toStop* place that guarantees the security of the *stopNightlifeE* transitions denying their execution while the *toStop* place hasn't a token inside so, it happens only in case an invalid_reply_fault is thrown. When all the activities have stopped their execution a token is put in the *stop* place representing the status in which the normal execution of the whole process is stopped. The next step from the stopped status is the execution of the fault handling as shown in the last figure. After the termination of the main scope referred by the transition *stopInitialScope* the fault

handling status is represented with a token in the *fHInvalidReplyFault* place followed by the *replyInvalidReplyFault* transition modeling the fault reply to the customer. If no invalid_reply_fault is thrown in order to terminate the whole process, represented by a token in *end* place the final output is generated and it is represented by the *outputAssign* place that triggers the *replyResult* transition in order to provide a reply to the customer followed by a place and the *endInitialScope* transition before the termination of the whole process that corresponds to reach the *end* place. The described workflow net satisfies the soundness property.