Project Report
Software Service Engineering

Tsinu Assefa Gezahgn (Mat: 523652)

1-Feb-2016

1. Introduction

The aim of this paper is to describe about the orchestration of Weather and Country Information Web Service. The diagram below illustrates the way how the orchestrator works In broad terms, from a given IPv4 address, the web service is capable to individuate the location and provide its weather and country information such as capital city, phone code, continent, currency code, flag and language.

The orchestration consists of three existing services:

- Service A: is a RESTful web service which takes IP address(IPv4)as input and returns the location's weather information as output;
- Service B: is a SOAP web service that takes IP address (IPv4)and provides the country code;
- Service C: is a SOAP web service that processes the input (country code) and, as output, returns the full country information.

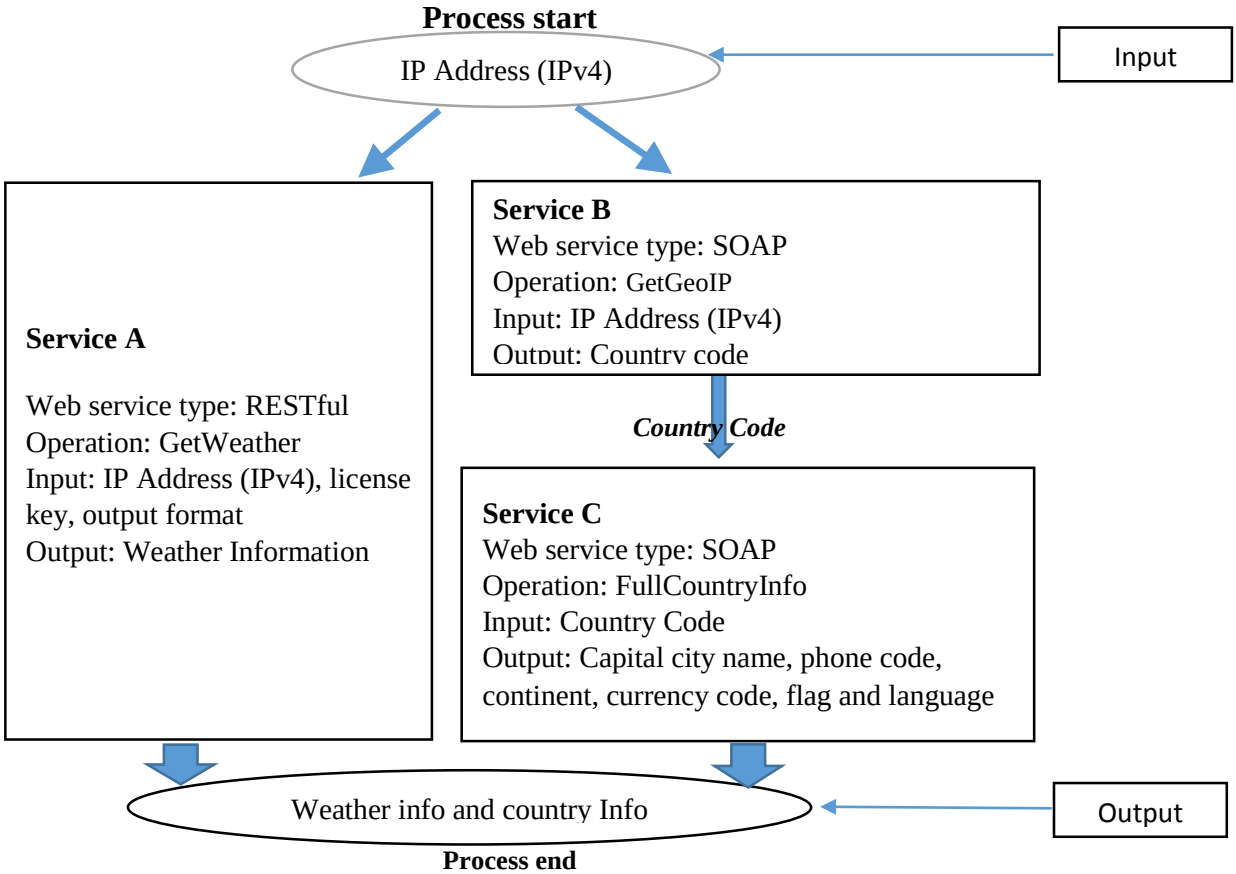

***Figure 1.1**: Representation of the orchestration*

1.1 Services Output Samples:

Service A - Worldweather: This is a RESTful service that provides city weather information by taking the IP address as an input. It is provided by http://www.worldweatheronline.com/ and located at http://api.worldweatheronline.com/free/v2/weather.ashx. Also, the project's key is "f31476d9f4d0a5d6469bb7c67da75".

Example:
http://api.worldweatheronline.com/free/v2/weather.ashx?q=8.8.8.8&key=f31476d9f4d0a5d6469bb7c67da75&fx=no

```
<data>
<request>
<type>IP</type>
<query>8.8.8.8</query>
</request>
<current_condition>
<observation_time>01:01 PM</observation_time>
<temp_C>12</temp_C>
<temp_F>54</temp_F>
<weatherCode>302</weatherCode>
<weatherIconUrl>
<![CDATA[
http://cdn.worldweatheronline.net/images/wsymbols01_png_64/wsymbol_0034_cloudy_with_heavy
_rain_night.png
]]>
</weatherIconUrl>
<weatherDesc> <![CDATA[ Rain ]]> </weatherDesc>
<windspeedMiles>9</windspeedMiles>
<windspeedKmph>15</windspeedKmph>
<winddirDegree>120</winddirDegree>
<winddir16Point>ESE</winddir16Point>
<precipMM>5.8</precipMM>
<humidity>88</humidity>
<visibility>13</visibility>
<pressure>1003</pressure>
<cloudcover>0</cloudcover>
<FeelsLikeC>10</FeelsLikeC>
<FeelsLikeF>51</FeelsLikeF>
</current_condition>
</data>
```

Sample output for IP address 8.8.8.8

Service B - GeoIp: This web service requires IPv4 address as input and resolve to, *Country Code*, and country name. This web service is provided by http://www.webservicex.net/ and its description language (WSDL) file is located at http://www.webservicex.net/geoipservice.asmx?WSDL.Then, in this orchestration, the *Country Code* it is used as an input for the Service C.

Example:

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <soap:Body>
        <GetGeoIPResponse xmlns="http://www.webservicex.net/">
            <GetGeoIPResult>
                <ReturnCode>1</ReturnCode>
                <IP>8.8.8.8</IP>
                <ReturnCodeDetails>Success</ReturnCodeDetails>
                <CountryName>United States</CountryName>
                <CountryCode>USA</CountryCode>
            </GetGeoIPResult>
        </GetGeoIPResponse>
    </soap:Body>
</soap:Envelope>
```

***Sample output for IP address* 8.8.8.8**

Service C - FullCountryInfo: This is a SOAP service which takes the *Country Code* as input and provides information such as capital city, phone code, continent code, flag and language, about the given country.

The web service description language (WSDL) file is located at http://www.oorsprong.org/websamples.countryinfo/CountryInfoService.wso?wsdl

Example:

```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <m:FullCountryInfoResponse xmlns:m="http://www.oorsprong.org/websamples.countryinfo">
      <m:FullCountryInfoResult>
        <m:sISOCode>US</m:sISOCode>
        <m:sName>United States</m:sName>
        <m:sCapitalCity>Washington</m:sCapitalCity>
        <m:sPhoneCode>1</m:sPhoneCode>
        <m:sContinentCode>AM</m:sContinentCode>
        <m:sCurrencyISOCode>USD</m:sCurrencyISOCode>
<m:sCountryFlag>http://www.oorsprong.org/WebSamples.CountryInfo/Images/USA.jpg</m:sCountryFlag>
<m:Languages>
          <m:tLanguage>
            <m:sISOCode>eng</m:sISOCode>
            <m:sName>English</m:sName>
          </m:tLanguage>
        </m:Languages>
      </m:FullCountryInfoResult>
    </m:FullCountryInfoResponse>
  </soap:Body>
</soap:Envelope>
```

Sample output for Country code USA

1.2 Considerations

In conclusion, this orchestration has three web services.
The Service Invocation is:

- Service A is invoked asynchronously; it is implemented to be invoked in asynchronous way via properly implemented proxy service.
- Both services B and C are invoked in synchronous way.
- Both services A and B are invoked in parallel and the output of service B is used to invoke service C.

The orchestrator has fault handling mechanism to manage unexpected replay from the services.

- If it doesn't receive a replay from Service A for 10 seconds then a fault to_fault will be thrown.
- If the replay received from service A or Service B is invalid then either A_reply_fault or B_reply_fault will be thrown respectively.

2. WS-BPEL implementation

2.1. WS-BPEL processes

This orchestration includes two BPEL processes:
1.**WeatherCountryInf.bpel:** used to orchestrate the three services by invoking service A in asynchronous way and services B and C in synchronous way.
2. **Proxyweather.bpel:** used as a proxy to invoke asynchronously service A.

2.1.1. WeatherCountryInf.bpel

This is the main BPEL process, where service orchestrator begins by accepting an input from the customer. When the process starts, a new instance is created and the receive activity waits for an incoming message. The received message is assigned to the input variable by AssignIP activity.

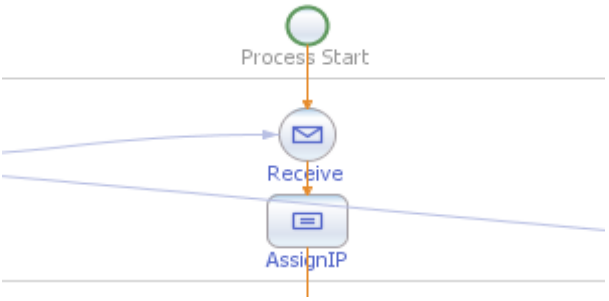

Figure 2.1: *Receive and AssignIP activity*

Then the invocation of service A (InvokeAsync) and service B (InvokeGeoPlugin) is done in parallel, i.e. the flow starts.

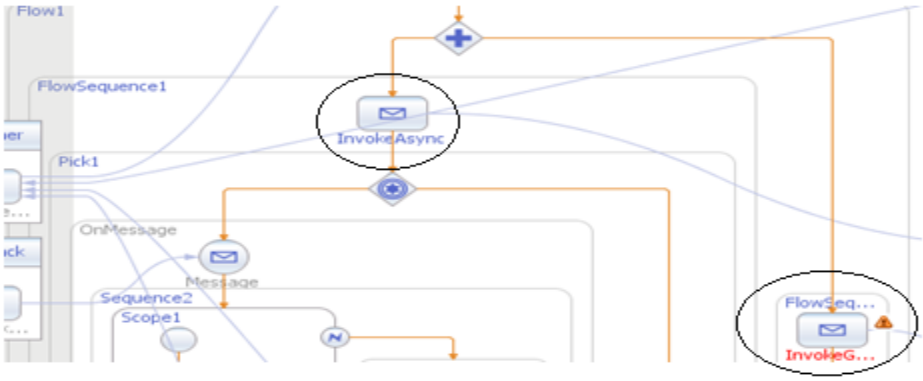

Figure 2.2: *Invocation of service A and B*

The first sequence of Flow (FlowSequence1) contains the invocation of asynchronous service (service A) InvokeAsyn activity, which is followed by a pick activity that consists of an onMessage and onAlarm activity.

On the onMessage side, the received message from the proxy service is verified using "if" activity. The "if" activity then checks either the message contains the location *weather information* or not. If the message contains *weather information*, then it is assigned to the customer output message and the sequence terminates. Otherwise, if the message does not contain the *weather information*, the A_*reply_fault* will be thrown and the fault handler will assign "Invalid Input" to the customer output and the sequence terminates. The throwing of A_reply_fault does not terminate the invocation of services B and C, since it reside on the different scope with respect to the scope of service B and service C.

In regards to the pick activity side (onAlarm), this is triggered if no result is received within a specified amount of time (i.e. 10 sec). Then, *to_fault* will be thrown and the throwing of to_fault terminates the invocation of Service B and Service C.

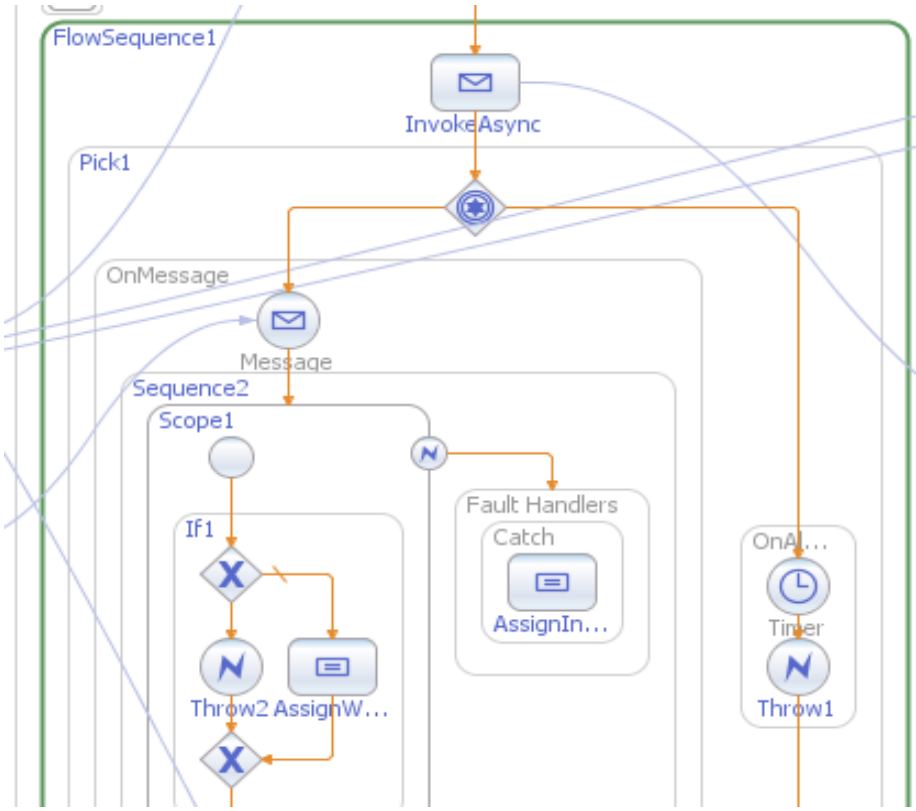

Figure 2.3: *The first sequence of Flow* (FlowSequence1)

The second sequence of the flow (FlowSequence) contains sequential invocation of services B and C. The activity InvokeGeoIP invokes a service B (GeoIp), takes as input the IPaddress (IPv4) and then generates the country code. The reply received from the service B is verified using "if" activity. The "if" activity then checks either the reply message is invalid or valid. If the message contains *Country code*, then it will be used as an input for the invoketion of service C(FullCountryInfo) ,then, service C returns the full country information (capital city name, country phone code, continent code, currency code, flag and language), which will be a part of the final result and the sequence terminates. Otherwise, if the reply message is invalid, B_*reply_fault* will be thrown and the fault handler will assign "Invalid Input" to the customer output message and the sequence terminates. Throwing of B_reply_fault does not terminate the invocation of services A.

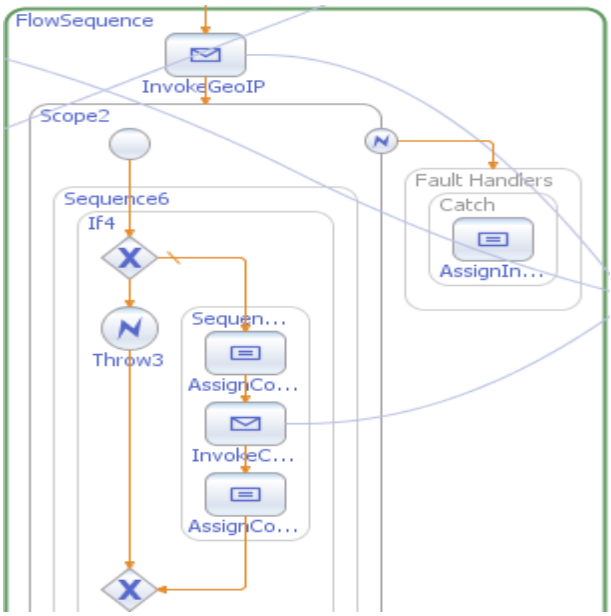

Figure 2.4: *The second sequence of Flow* (FlowSequence)

2.1.2. Proxyweather.bpel

This BPEL process abilities us to call asynchronously the proxy BPEL for service A. The proxy BPEL process is implemented with a WSDL client having a one-way operation type. It receives an IP address and a BPEL Id to keep correlations between the message and its corresponding process instances. After assigning the received IP address, the static key and the output format (xml) to the input of service A (weatheronline), it is invoked synchronously. After that, it will assign the result to the input of callback WSDL interface, then it will invoke callback interface. Lastly, the callback interface from the WeatherCountryInf.bpel will be receiving the result of the asynchronous call.

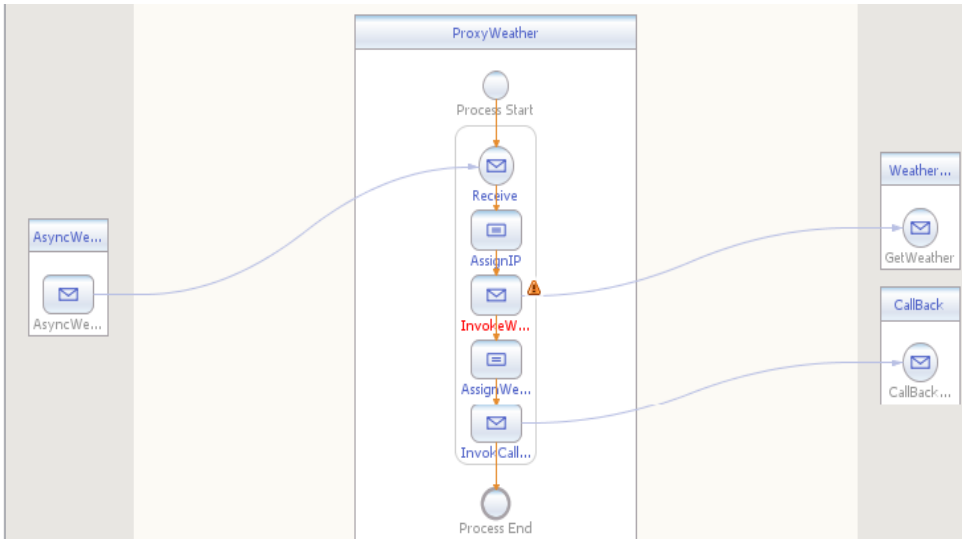

Figure 2.5: *Proxyweather.bpel*

2.2 Testing

The Customer executes CustomerOperatioin and he/she is required to fill in the IP address. To verify this, I successfully run three different test cases outlined below.

2.2.1. Test case 1 - Input without fault

In this case, the input is a valid IP address (IPv4): 8.8.8.8. The following output was generated by the orchestrator.


```
<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://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Body>
   <m:CustumerOperationResponse xmlns:m="http://j2ee.netbeans.org/wsdl/CountryDetail_Information/src/Custumer">
    <Weather xmlns:msgns="http://j2ee.netbeans.org/wsdl/ProxyWeatherInfo/src/CallBack">
     <request>
      <type>IP</type>
      <query>8.8.8.8</query>
     </request>
     <current_condition>
      <observation_time>02:44 PM</observation_time>
      <temp_C>5</temp_C>
      <temp_F>41</temp_F>
      <weatherCode>113</weatherCode>

<weatherIconUrl>http://cdn.worldweatheronline.net/images/wsymbols01_png_64/wsymbol_0008_clear_sky_night.png</weatherIcon
Url>
      <weatherDesc>Clear </weatherDesc>
      <windspeedMiles>4</windspeedMiles>
      <windspeedKmph>6</windspeedKmph>
      <winddirDegree>270</winddirDegree>
      <winddir16Point>W</winddir16Point>
      <precipMM>0.0</precipMM>
      <humidity>75</humidity>
      <visibility>16</visibility>
      <pressure>1024</pressure>
      <cloudcover>0</cloudcover>
      <FeelsLikeC>4</FeelsLikeC>
      <FeelsLikeF>39</FeelsLikeF>
     </current_condition>
    </Weather>
    <Country_Info xmlns:m="http://www.oorsprong.org/websamples.countryinfo"
xmlns:ns0="http://j2ee.netbeans.org/wsdl/CountryDetail_Information/src/Custumer">
     <m:sISOCode>US</m:sISOCode>
     <m:sName>United States</m:sName>
     <m:sCapitalCity>Washington</m:sCapitalCity>
     <m:sPhoneCode>1</m:sPhoneCode>
     <m:sContinentCode>AM</m:sContinentCode>
     <m:sCurrencyISOCode>USD</m:sCurrencyISOCode>
     <m:sCountryFlag>http://www.oorsprong.org/WebSamples.CountryInfo/Images/USA.jpg</m:sCountryFlag>
     <m:Languages>
      <m:tLanguage>
       <m:sISOCode>eng</m:sISOCode>
       <m:sName>English</m:sName>
      </m:tLanguage>
     </m:Languages>
```

```
    </Country_Info>
   </m:CustumerOperationResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.2.2. Test case 2 - to_fault

This test was run to show the orchestrator's response when the orchestrator does not receive a reply from the proxy for a certain amount of time (i.e. 10 sec). To demonstrate this case set the onAlarm activity to zero.

Example: input IPaddress equal to 8.8.8.8

```
<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://schemas.xmlsoap.org/soap/envelope/">
 <SOAP-ENV:Body>
  <SOAP-ENV:Fault>
   <faultcode>SOAP-ENV:Client</faultcode>
   <faultstring>A_reply_fault</faultstring>
   <detail>
    <fault_msg>Time Out</fault_msg>
   </detail>
  </SOAP-ENV:Fault>
 </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

2.2.3. Test case 3 A_reply_fault

For an Invalid input, the proxy service will replay an error message, then A_reply_fault will be thrown and the orchestrator will return an error message ("Invalid Input!!") to the customer.

2.2.4. Test case 4 B_reply_fault

Service B (GeoIp) reply error message for the invalid IP address input then B_reply_fault will be thrown and the orchestrator will return an error message ("Invalid Input!") to the customer.

3. Analysis of the WS-BPEL specification

This section covers the Workflow net of the WS-BPEL orchestrator.

Description of the few most relevant places:

- ***normal***: is used to represent the normal execution of the process. A token is consumed from this place when the flow ends with or without throwing fault.
- ***toStopper***: is used to represent a status where a fault was thrown and other processes inside the scope are expected to terminate before the fault handler is fired.
- ***Stopped***: is used to execute the fault handler, considering that all activities in the scope terminated.

The workflow net, as shown in Figure 3.2, starts with the *initial* place, which has one token inside it. The receive transition starts by putting a token in the *normal* place then, the *FlowStart* transition (AND-split) is fired, the parallel execution will start and the token will split in the two branches of the flow.

The first branch is concerning the asynchronous service interaction, which starts with *InvokeAsync* transition. After that, the token goes to the pick activity and it will either go to *OnAlarm* (with Time Trigger) or to *OnMessage* (Message Trigger) activities. A fault can be thrown from both sides of the pick. In the left side of the pick a token moves from *OnAlarm* transition to the *Throw_to_fault* transition, when the *Throw_to_fault* transition is fired, it consumes the *normal* place token, puts it into the i*faultHandler and to the toStopper places. F*urther, when one of the *stopper* transition is fired *(Stoper1-Stoper8*), it consumes a token from *toStopper* and consumes a token from the second branch of the flow (invocation of service B and C). When the stopper transition is fired, it will terminate the second branch of the flow and puts a token in the *stopped* place. Lastly, the *tr*ansition *FaultHandler* will be fired, and this will assign and reply a fault message and ultimately terminate the process.

During the throwing of *to_fault*:

1. If service *B (GeoIp)* had not been invoked then a token is consumed by *Stoper1* which will terminate the invocation of *GeoIp* and full*CountryInfo* (Service C).
2. If *GeoIp* had been invoked then a token is consumed by *Stoper2* which will terminate the invocation of full*CountryInfo*. If the reply is verified as a valid then, the token is consumed by *Stoper3* else *Stopper4* (replay is invalid) terminates the invocation of full*CountryInfo*.
3. If the throwing of *to_fault* is after the invocation of full*CountryInfo*, in this case the token is consumed by *Stoper6 or by Stoper8*.
4. If the throwing of *to_fault* is after B_replay_fault, in this case the token is consumed by *Stoper5*.

On the right side of a pick activity branch (OnMessage) after the receiving of a message, the workflow net is followed by *if_1* transition (XOR-split) to verify whether the received result is valid or not. If the result is invalid, the token goes to *Throw_A_reply_fault* transition. After that, *AssignInvalid* transition is fired and the token goes to the E*ndflow* (AND-Join). In the case of the asynchronous service returns a valid result the *AssignWeather* transition is fired and the token goes to the E*ndflow* (AND-Join). Thereafter, reply transition is fired then finally the end place will hold the token.

The second branch of the flow is about the invocation of GeoIP and fullCountryInfo service interaction, which starts with *InvokeGeoIp* transition then, the token goes to if _2 transition (XOR-split to verify whether the received result is valid or not. If the result is invalid, the token goes to Throw_B_reply_fault transition. After that, *AssignInvalid* transition is fired and the token goes to the E*ndflow* (AND-Join). Conversely, the received result is valid then *AssignCountryInf* transition is fired and the token goes to the E*ndflow* (AND-Join). Thereafter, reply transition is fired then finally the end place will hold the token.

Soundness:-The workflow net is sound.

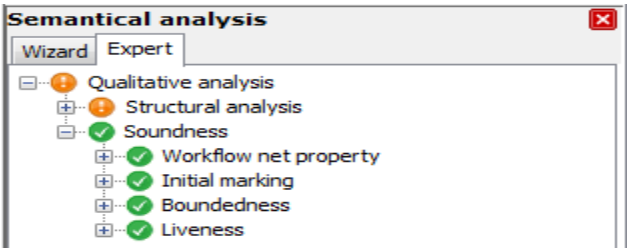

Figure 3.1: semantic analysis of workflow net

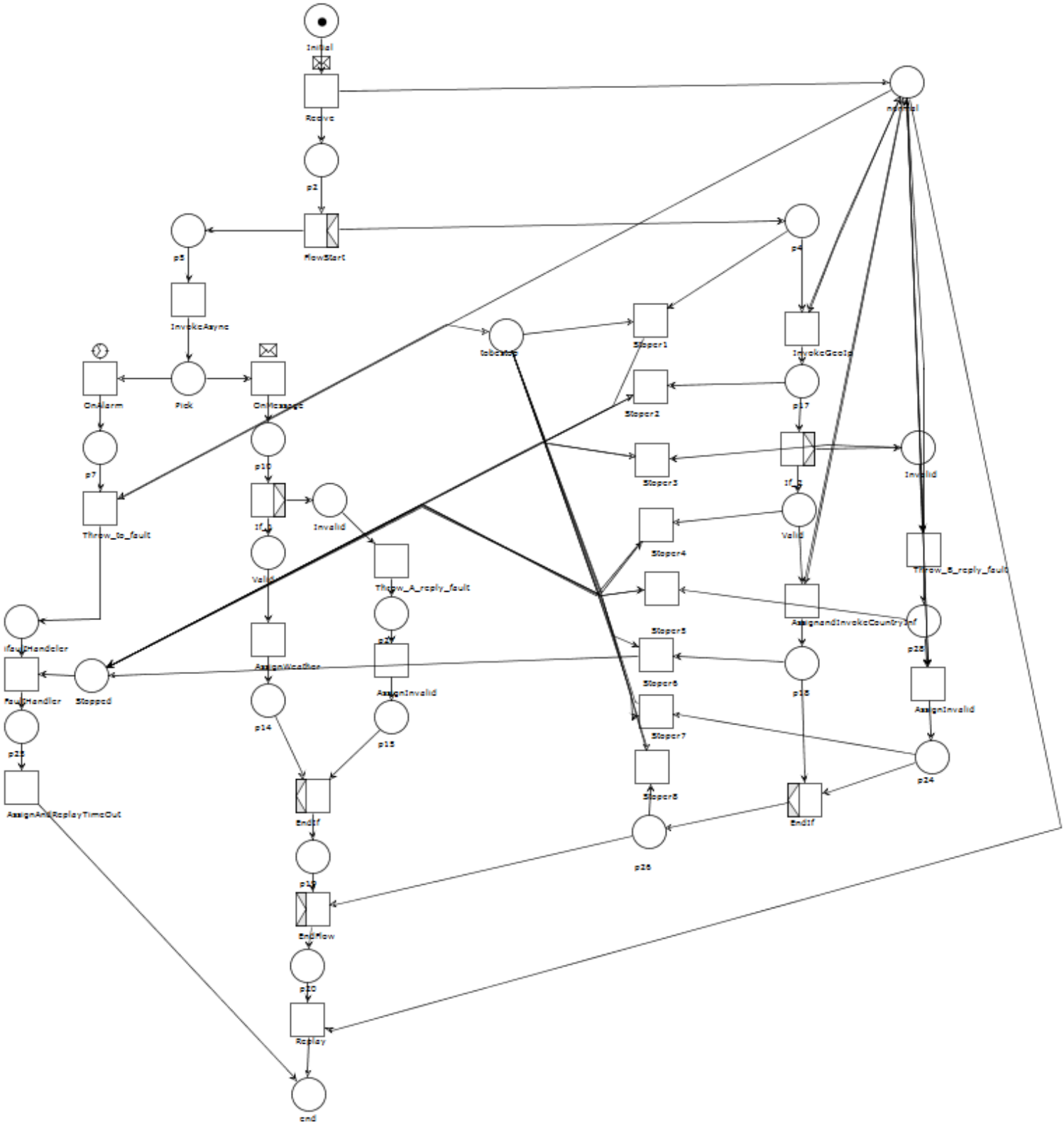

Figure 3.2. *Workflow net*