One to many send/receive (with statically determined partners)

In this implementation we demonstrate the **one to many send receive pattern** wherein a single sender sends a message to several different partner services, where the addresses of these partner services are assumed to be given as deployment-time information. A typical example of this pattern is requesting quotes from an *a priori* known set of suppliers.

There are three constraints on the implementation of the pattern.

1. Specify that a certain specified *number of receivers* actually receive the message and respond. E.g. requests are sent to 3 suppliers but only 2 responses are required.
2. Specify that the necessary number of responses is achieved within a *timeout* period. E.g. if the required number of responses has not been received within the time limit it may be necessary to send requests to another set of suppliers.
3. Check, after processing has completed, whether a *success condition* has been met. E.g. if the process timed out and all the required responses have not been received the application may still be able to proceed with a smaller number.

Modeling the constraints

Number of receivers

To provide the ability to define how many responses are required a while loop is created that loops until the required number of responses is received.

The responses are collected in a pick activity where each possible responder is itemized and at each iteration through the while loop if a message has been received it is processed.

```
<while name="while-1" condition="bpws:getVariableData('counter')> 2">
    <pick name="pick-1">
        <onMessage partnerLink="rcvr1"
                        portType="tns:BasicCallback"
                        operation="onResult" variable="rcvd1">
            <assign name="rcvd1Update">
                <copy>
                    <from expression="bpws:getVariableData('counter')+1"></from>
                    <to variable="counter"/>
                </copy>
                <copy>
                    <from expression="concat(bpws:getVariableData('content'),' - ', bpws:getVariableData('rcvd1',
                                    'payload','/tns:BasicResponse/tns:result'))"></from>
                    <to variable="content"/>
                </copy>
            </assign>
        </onMessage>
        <onMessage partnerLink="rcvr2"
            …
        </onMessage>
        <onMessage partnerLink="rcvr3"
            …

        </onMessage>
    </pick>
</while>
```

A problem encountered with the Oracle BPEL implementation (version 2.1.2) is that the pick activity will select one of the available messages and discard the rest. This seems to point to a bug in this version of the Oracle BPEL engine.

Timeout

To provide the ability to stop collecting responses after some period of time, the while loop is contained within a scope activity that has an “on alarm”. If the alarm is triggered an exception (simply called “timeoutName” in this example) is thrown to be caught in the outer scope. The exception allows the process to exit the while/pick loop before it has finished.

```
<scope name="outerScope">
    <faultHandlers>
        <catch faultName="tns:timeoutName" faultVariable="timeoutFaultVar">
            <assign name="updateTimeout">
                <copy>
                    <from expression="concat(bpws:getVariableData('content'), '- Timed out –')">
                    </from>
                    <to variable="content"/>
                </copy>
            </assign>
        </catch>
    </faultHandlers>
    <flow name="sendReceiveFlow">
        <scope name="innerReceiveScope">
            <eventHandlers>
                <onAlarm for="'PT10S'">
                    <sequence>
                        <throw name="throw-1" faultName="tns:timeoutName"
                         faultVariable="timeoutFaultVar"/>
                    </sequence>
                </onAlarm>
            </eventHandlers>
            <while name="while-1" condition="bpws:getVariableData('counter')> 2">
                <pick name="pick-1">
                    …
                </pick>
            </while>
        </scope>
    </flow>
</scope>
```

The exception is caught in the outer scope and in this example the handler updates the content variable to note that the process did time out.
In this sample code, two nested scopes are used to capture the timeout and the interruption that it causes. However, it is possible to merge these two scopes into a single one as shown in the BPEL sample code provided for the “One-to-many-send pattern with dynamically determined partners” available on the www.serviceinteraction.com web site.

Success condition

A switch activity after the conclusion of the outer scope allows the process to check if enough responses have been received and to take whatever actions are necessary. In this example, if at least one message has been received the process is considered successful .

```
<switch name="successCondition">
    <case condition="bpws:getVariableData('counter') >= 1">
```

```
        <assign name="success">
            <copy>
                <from expression="concat(bpws:getVariableData('content'), ': counter = ',
                                         ora:getContentAsString(bpws:getVariableData('counter')),
                                         '- SUCCESS')">
                </from>
                <to variable="output" part="payload" query="/tns:MultiSendReceiveResponse/tns:result"/>
            </copy>
        </assign>
    </case>
    <otherwise>
        <assign name="failure">
            <copy>
                <from expression="concat(bpws:getVariableData('content'), ': counter = ',
                                         ora:getContentAsString(bpws:getVariableData('counter')),
                                         '- FAILURE')">
                </from>
                <to variable="output" part="payload" query="/tns:MultiSendReceiveResponse/tns:result"/>
            </copy>
        </assign>
    </otherwise>
</switch>
```

The MultiSendReceive process in ORACLE BPEL

The overall process flow is shown in figures 1 and 2. Figure 1 shows the flow activity that invokes the Basic (provider) services while also starting the receive responses activities in parallel, the scope containing the while loop and the pick activities is shown in detail.

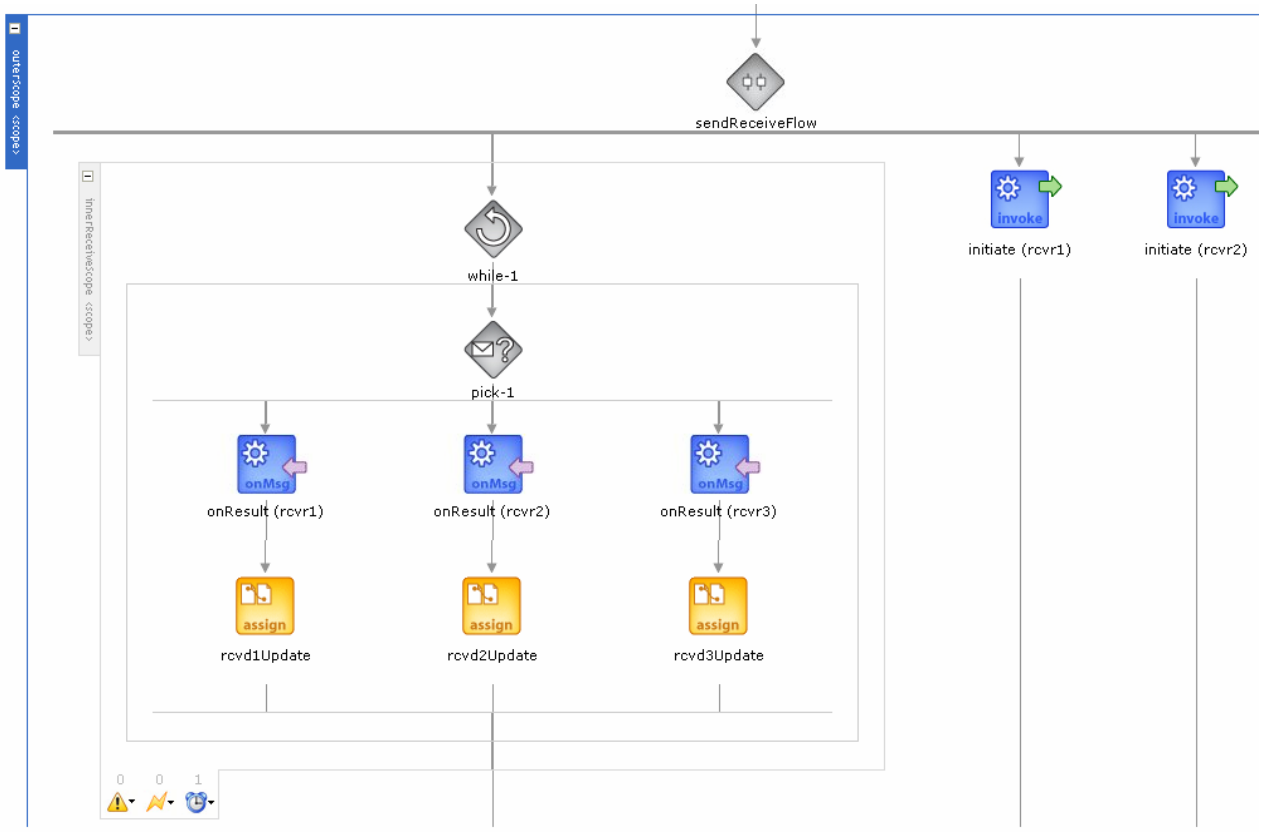

Figure 1The receive responses activities in detail

Figure 2 gives an overall view of the MultiSendReceive process where it is easier to see the parallel invocations of the receivers and the activities that collect their responses.

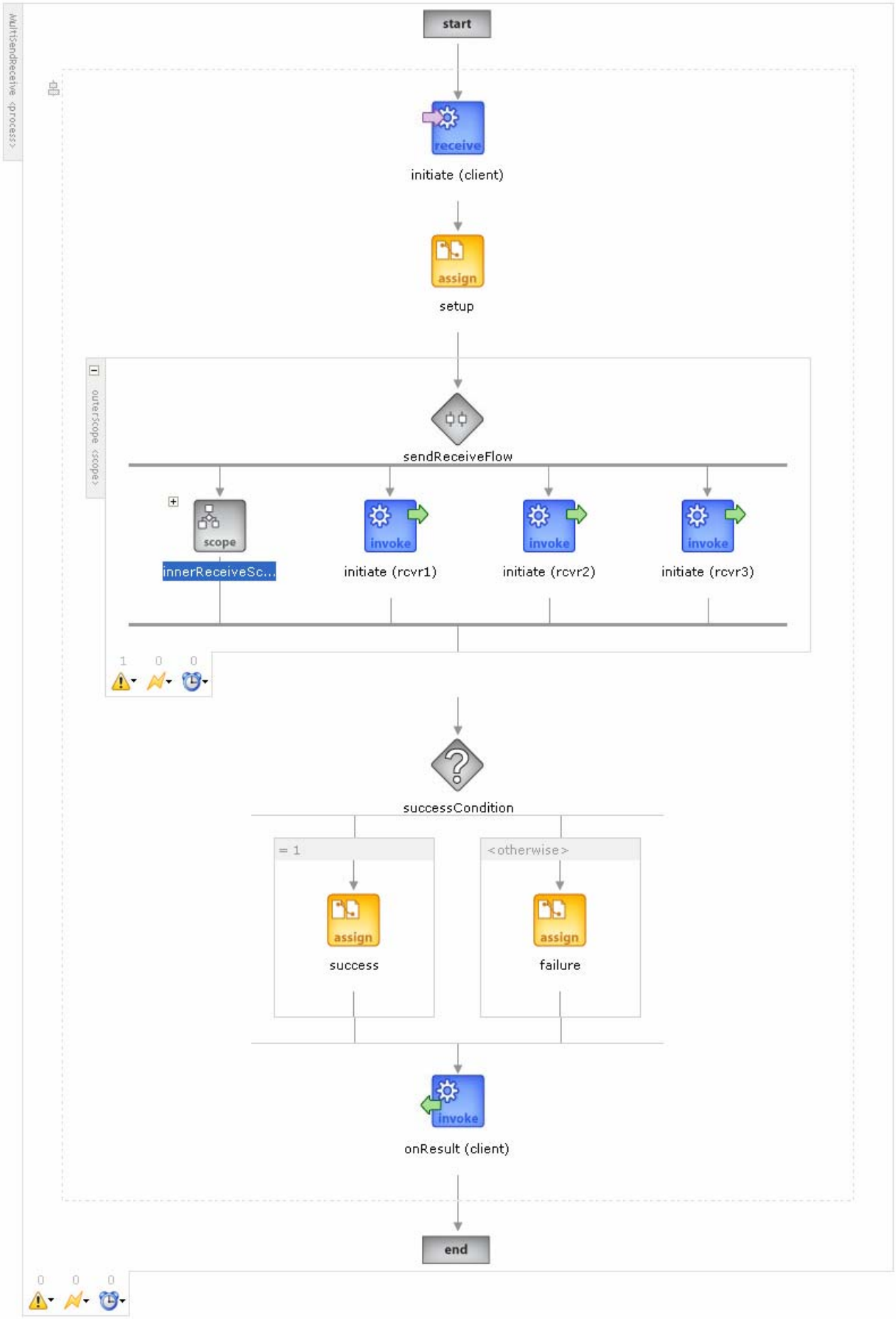

Figure 2 The overall MultiSendReceive process

Figure 3 shows the initiation of the process in the Oracle BPEL console.

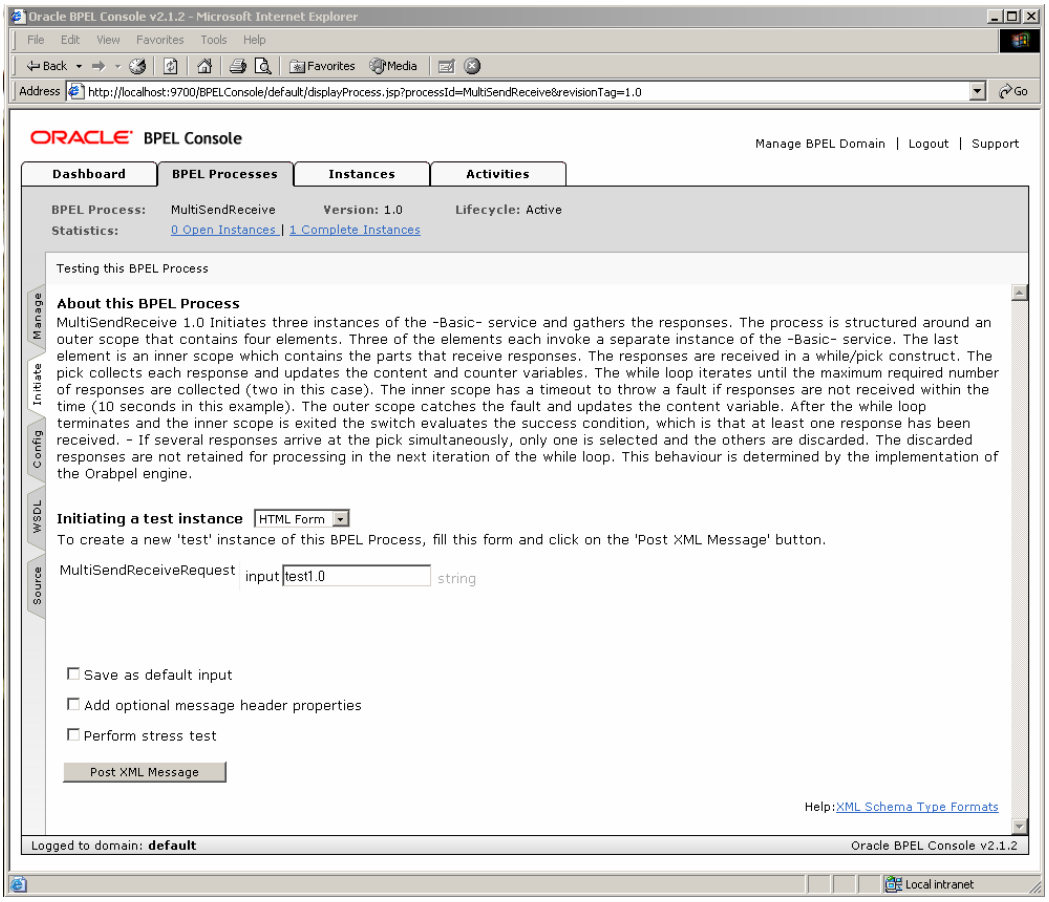

Figure 3 Initiating the process in the ORACLE BPEL Console