

Developing BPEL processes

Fourth part: BPEL Events



Table of contents

- BPEL: Pick
- BPEL:Event Handler
- Exercise: a modified Buyer
- Faults & Fault handling in BPEL
- BPEL:Fault Handler
- BPEL: Throw
- Exercise: Faulty Anagrams
- BPEL:Compensation Handler
- BPEL:Compensate



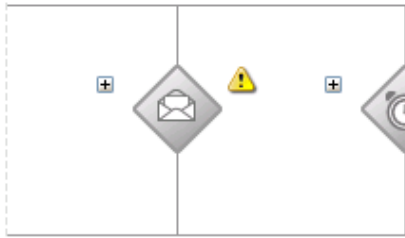


BPEL: Pick

- The Pick activity makes the process wait until a specified event occurs:
 - A message to arrive
 - A timeout to expire
- Depending on what comes first, different activity branches are selected and executed
 - A branch labeled “OnMessage”
 - Works like a BPEL:Receive
 - A branch labeled “OnAlarm”
 - Works like a BPEL:Wait

BPEL: Pick (2)

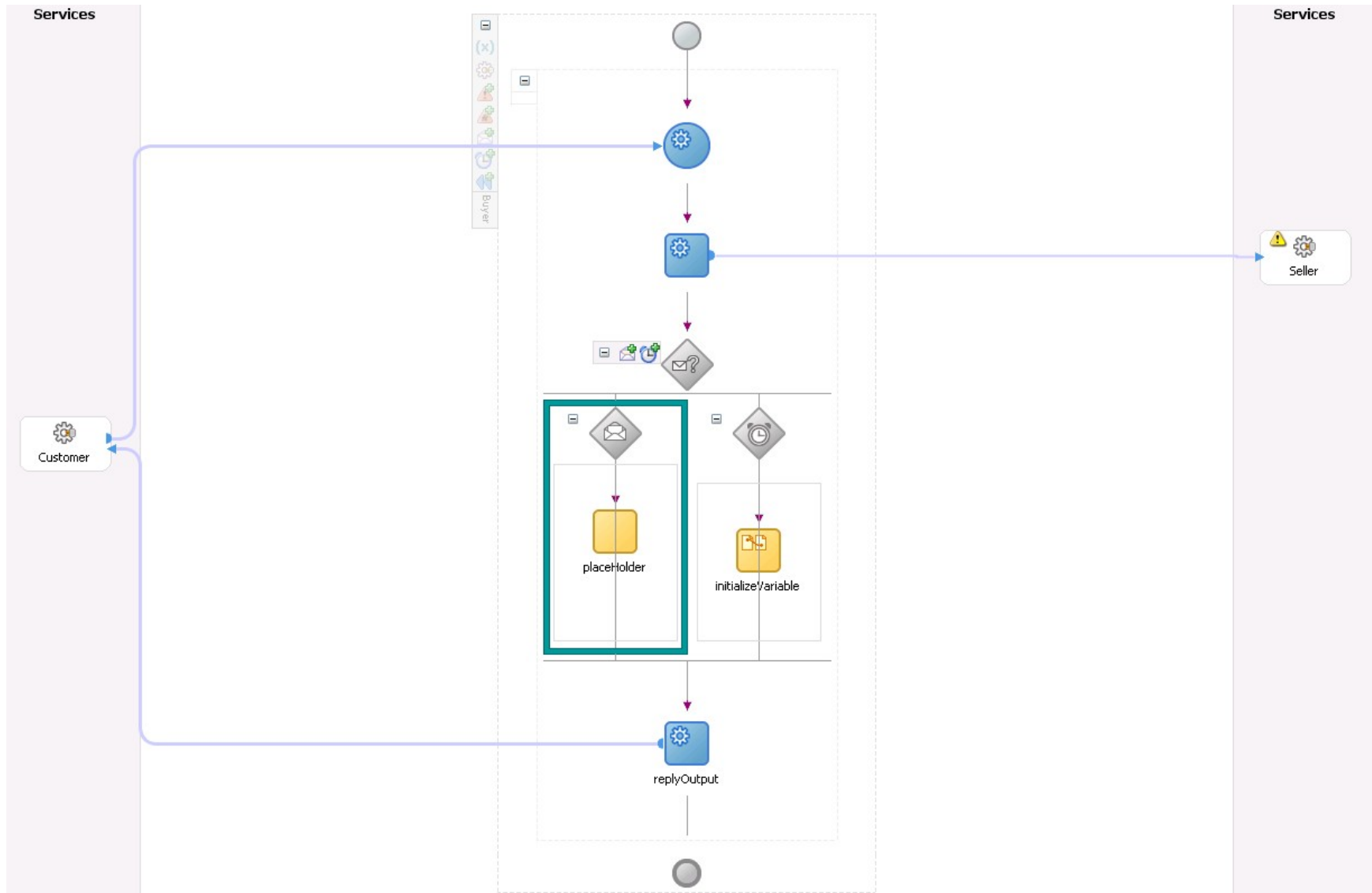
- Remember:
 - You can have as many “OnMessage” and “OnAlarm” as you wish
 - You MUST have at least one “OnMessage”
- What's Pick best suited for?
 - Adding timeout handling to Receive activities
 - Waiting for different kind of possible incoming messages rather than just one type
- In case of conflict, a fault is thrown



BPEL: Event Handler

- In BPEL, one could associate Scopes with Event Handlers
 - OnMessage, activates whenever messages arrive
 - OnAlarm, waits for timeout to expire
- Looks like Pick activity
 - But it's **always** active
- Event handlers affect only activities included in surrounding Scope
 - Like all BPEL handlers
 - Another feature of Scope: separating handling contexts
- You can have as many handlers as you want
- In case of conflict, “conflictingReceive” fault is thrown

Exercise: a modified Buyer

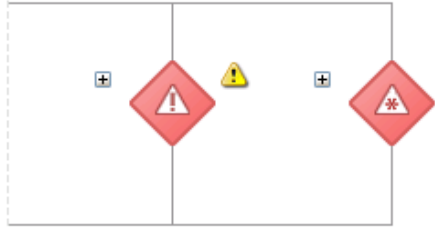


Faults & Fault handling in BPEL

- Sometimes BPEL processes fails
 - Invoked web services return faults
 - BPEL process itself generates a runtime error
- When **faults** occur
 - Execution is interrupted
 - Process is put into faulted state
- No answer is sent back to the requester, unless BPEL process does it explicitly
 - With a Reply inside a request/response: define a “fault” message besides “input” and “output” messages
 - With an Invoke to a callback

Faults & faults handling in BPEL (2)

- Two types of faults
 - Business faults
 - Occurs because of
 - A throw activity inside the BPEL process itself
 - An invoked web service that returned a fault
 - May have message type
 - Defined in the WSDL
 - Runtime faults
 - Standard faults, cannot be defined by users
 - They are 11: selectionFailure, conflictingReceive, conflictingRequest, mismatchedAssignmentFailure, joinFailure, forcedTermination, correlationViolation, uninitializedVariable, repeatedCompensation and invalidReply
 - No messageType associated



BPEL:Fault Handler

- Faults resemble Java Exceptions
 - Like an Exception, a fault can be caught
- To catch faults in BPEL, you use **Fault handlers**
 - Fault handlers work like “Java try ... catch” blocks
 - When a fault is raised inside a Scope
 - If there's a valid Fault handler, it's invoked
 - Otherwise, fault get propagated to enclosing Scope
 - Propagation stops when global scope is reached
 - ...same behavior as with Java Exceptions

BPEL:Fault Handler (2)

- Fault handlers come in two distinct flavors:
 - Catch: you need to declare what type of fault has to be caught
 - Done through a fault name
 - ?Must have been defined in WSDLs or be among default ones?
 - CatchAll: catch all possible faults
- If you use both, CatchAll has lowest priority
- You can include any kind of activity inside fault handlers
- Fault handlers are attached to Scopes
 - Only faults generated inside the scope get caught
 - Fault handlers can be attached to the global scope also



throwActivity

BPEL: Throw

- The Throw activity is used to raise a Fault explicitly
 - Is a way to trigger the execution of a Fault Handler
- Thrown fault are handled just like normal faults
 - If no handler is defined for that fault, fault is propagated upwards, traversing defined Scopes in reverse order of execution until it reaches the outer process scope
- Need to define:
 - Fault name: defined in the process WSDL
 - Fault variable: will hold fault message
- Unlike Java methods, you don't have to catch every fault you could throw

Exercise: Faulty Anagrams

- We would like the Anagrams example perform a check on input string before processing it
- If it's empty, throws a “noInputFault” fault
- It catches the fault, and sends fault message to the client
- Suggestions:
 - Define the fault message inside the “process” operation, just after input and output messages, in Anagrams.wsdl
 - Define a messageType for the fault message inside project's XML schema Anagrams.xsd
 - Add a Switch activity inside the BPEL process, with no “else” clause
 - Test the condition (hint: normalize-space function may come handy) and, in case, throw the fault
 - Add a fault handler on the global scope, to catch the fault
 - Inside the fault handler, put a Reply activity
 - specify fault information in the “Fault QName” section of activity's properties dialog (use torchlight button...)
 - Don't forget to initialize output variable before using it, and to complete the handler with a Terminate activity



BPEL: Compensation Handler

- Compensation handlers have the goal of **compensating** activities
 - Reversing the effects of a failed transactions, by undoing its work
- No default compensation policy defined
 - Developers have to declare an appropriated sequence of operations to be executed upon invocation
- One Compensation handler could be added to
 - A Scope
 - An Invoke activity
- To invoke a Compensation Handler, use a Compensate activity

BPEL: Compensation Handler (2)

- Compensation handlers show no direct analogy with Java language constructs
 - They are more akin to the idea of database transactions
 - Compensation handlers need arises because many business process takes long to execute, yielding to Long Running Transactions
- Generally speaking, a transaction consist of many Scopes
 - Each Scope has its own Compensation Handler
 - If whole process fails, already executed Scopes are compensated
 - To be compensated a Scope must have been executed successfully
- Example: a travel planning process includes
 - Ticket booking
 - Hotel reservation
 - Car reservation
- If trip get canceled...
 - ...reservation transactions have to be compensated
 - by means of other transactions
 - in the appropriate order



compensate

BPEL:Compensate

- Invokes a compensation handler by its surrounding Scope's name
 - If name is not defined, invokes compensation handler of the immediately enclosed Scope
 - To compensate an Invoke activity, use activity's name
- Can be used only
 - Inside a Fault handler
 - Inside a Compensation handler
- You can compensate only Scopes that completed normally
 - If you try to compensate a Scope that did not complete normally, nothing happens
- You cannot compensate the same Scope twice