
Composing Web Services using BPEL4WS

Francisco Curbera, Frank Leymann,
Rania Khalaf
IBM

©IBM
2/03 Curbera, Leymann, Khalaf

Business Process Execution Language

- BPEL4WS enables:
 - Defining business processes as coordinated sets of Web service interactions.
 - Define both abstract and executable processes.
 - Abstract processes are for e-commerce specifications.
 - Executable processes provide a model to integrating enterprise applications.
 - BPEL enables the creation of compositions of Web services
 - Composition based on abstract descriptions
- Where it comes from:
 - Strong roots in traditional flow models.
 - Plus many concepts from structured programming languages.
 - All laid on top of WSDL and core XML specifications.
 - Merges WSFL (graph-oriented) and XLANG(algebraic) concepts.

©IBM
2/03 Curbera, Leymann, Khalaf

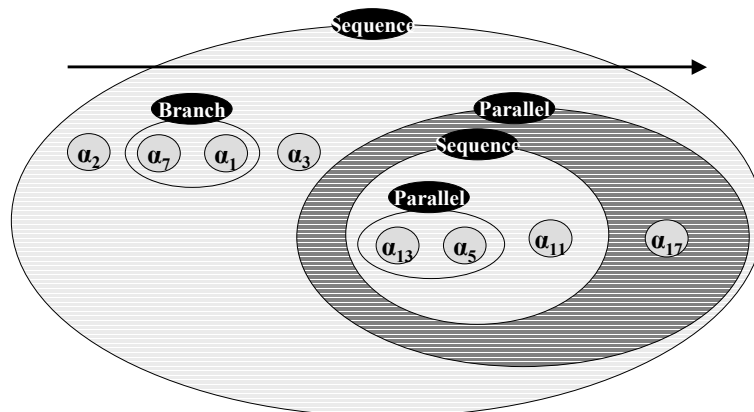
Algebraic/Calculus Approach To Flows

- Collection of „elementary“ activities
 - $A = \{\alpha_1, \dots, \alpha_n\}$
 - RPC, wait, send, ...
- Collection of „complex“ activities
 - $\Omega = \{\omega_1, \dots, \omega_m\}$
 - Sequence, Parallel, Branch, Loop, ...
 - Have other activities as parameters
 - Elementary as well as complex activities allowed
- $\omega_3(\alpha_2, \omega_1(\alpha_7, \alpha_1), \alpha_3, \omega_2(\omega_3(\omega_2(\alpha_{13}, \alpha_5), \alpha_{11}), \alpha_{17})))$

©IBM
2/03 Curbera, Leymann, Khalaf

Algebraic Flows Representation

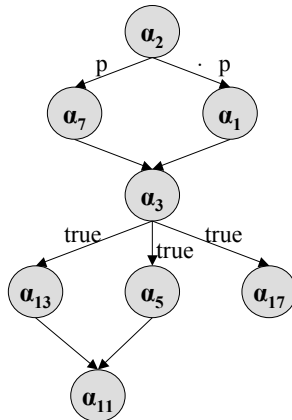
$\text{SEQ}(\alpha_2, \text{BCH}(\alpha_7, \alpha_1), \alpha_3, \text{PAR}(\text{SEQ}(\text{PAR}(\alpha_{13}, \alpha_5), \alpha_{11}), \alpha_{17})))$



©IBM
2/03 Curbera, Leymann, Khalaf

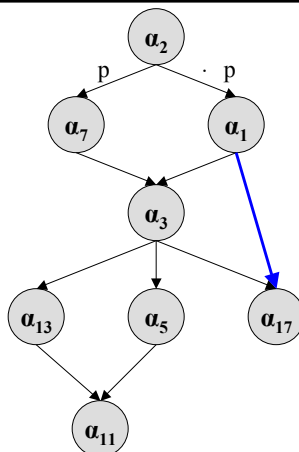
And As Pure Graph

$\text{SEQ}(\alpha_2, \text{BCH}(\alpha_7, \alpha_1), \alpha_3, \text{PAR}(\text{SEQ}(\text{PAR}(\alpha_{13}, \alpha_5), \alpha_{11}), \alpha_{17}))$



©IBM
2/03 Curbera, Leymann, Khalaf

Finally: A “Non-Algebraic” Graph



©IBM
2/03 Curbera, Leymann, Khalaf

Structure of a BPEL4WS Process

```
<process ...>
```

```
  <partners> ... </partners>
```

```
    <!-- Web services the process interacts with -->
```

```
  <containers> ... </containers>
```

```
    <!-- Data used by the process -->
```

```
  <correlationSets> ... </correlationSets>
```

```
    <!-- Used to support asynchronous interactions -->
```

```
  <faultHandlers> ... </faultHandlers>
```

```
    <!-- Alternate execution path to deal with faulty conditions -->
```

```
  <compensationHandlers> ... </compensationHandlers>
```

```
    <!-- Code to execute when "undoing" an action -->
```

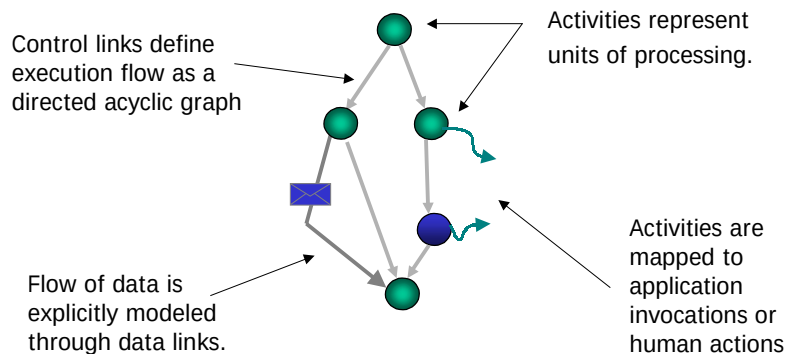
```
  (activities)*
```

```
    <!-- What the process actually does -->
```

```
</process>
```

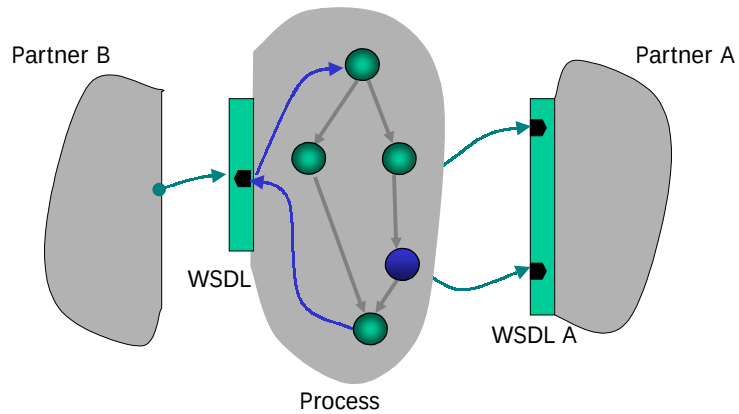
©IBM
2/03 Curbera, Leymann, Khalaf

Traditional Flow Models



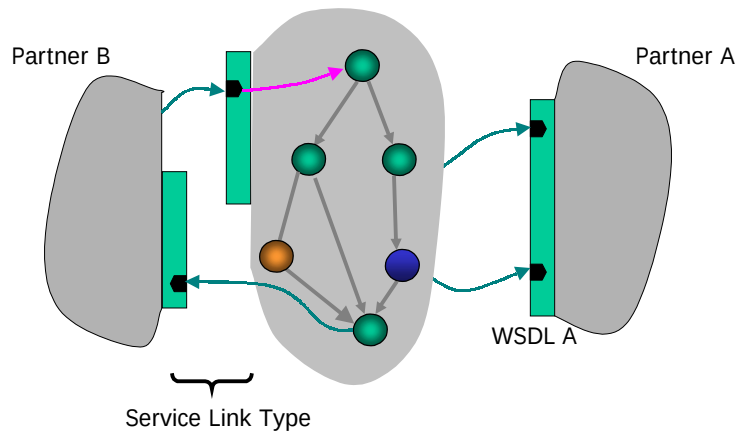
©IBM
2/03 Curbera, Leymann, Khalaf

BPEL and WSDL Partners



©IBM
2/03 Curbera, Leymann, Khalaf

BPEL and WSDL Partners



©IBM
2/03 Curbera, Leymann, Khalaf

Partner Definitions and Links

```
<partner name="..." serviceLinkType="..."
  partnerRole="..." myRole="..." />
```

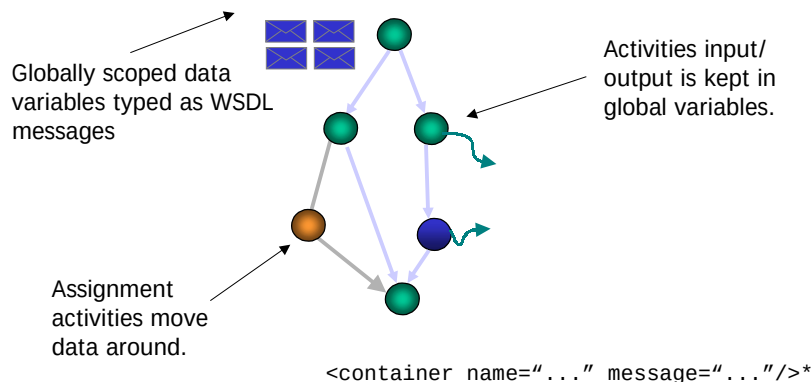
<!-- A partner is accessed over a WS "channel", defined by
a service link type -->

```
<serviceLinkType name="...">
  <role name="...">
    <portType name="..."/*>
  </role>
  <role name="...">
    <portType name="..."/*>
  </role>
</serviceLinkType>
```

<!-- A SLT defines two roles and the portTypes that each role needs to
support -->

©IBM
2/03 Curbera, Leymann, Khalaf

BPEL Data Model



©IBM
2/03 Curbera, Leymann, Khalaf

BPEL Basic Activities

```
<invoke partner="..." portType="..." operation="..."
  inputContainer="..." outputContainer="..." />
<!-- process invokes an operation on a partner: -->
```



```
<receive partner="..." portType="..." operation="..."
  container="..." [createInstance="..."] />
<!-- process receives invocation from a partner: -->
```



```
<reply partner="..." portType="..." operation="..."
  container="..." />
<!-- process send reply message in partner invocation: -->
```

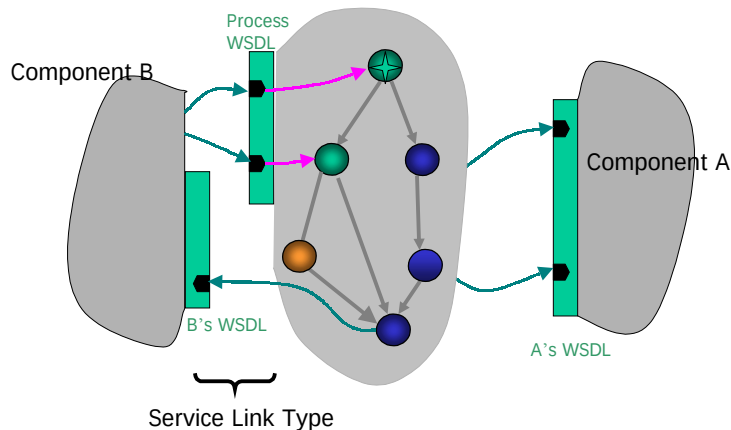


```
<assign>
  <copy>
    <from container="..." /> <to container="..." />
  </copy>+
</assign>
<!-- Data assignment between containers: -->
```



©IBM
2/03 Curbera, Leymann, Khalaf

BPEL Composition of Web Services



©IBM
2/03 Curbera, Leymann, Khalaf

More Basic Activities

`<throw faultName="..." faultContainer="..." />`
<!-- process detects processing error and switches into fault processing mode -->

`<terminate/>`
<!-- pull the plug -->

`<wait for="..."? until="..."? />`
<!-- process execution stops for a specified amount of time-->

`<empty>`
<!-- Do nothing; a convenience element -->

©IBM
2/03 Curbera, Leymann, Khalaf

BPEL Structured Activities

`<sequence>`
<!-- execute activities sequentially-->

`<flow>`
<!-- execute activities in parallel-->

`<while>`
<!-- iterate execution of activities until condition is violated-->

`<pick>`
<!-- several event activities (receive message, timer event) scheduled for execution in parallel; first one is selected and corresponding code executed. -->

`<link ...>`
<!-- defines a control dependency between a source activity and a target -->



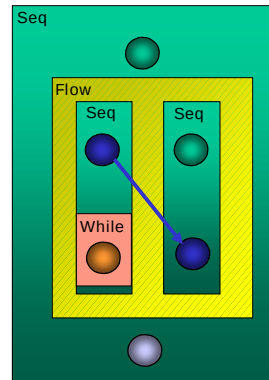
©IBM
2/03 Curbera, Leymann, Khalaf

Nesting Structured Activities. Example

```

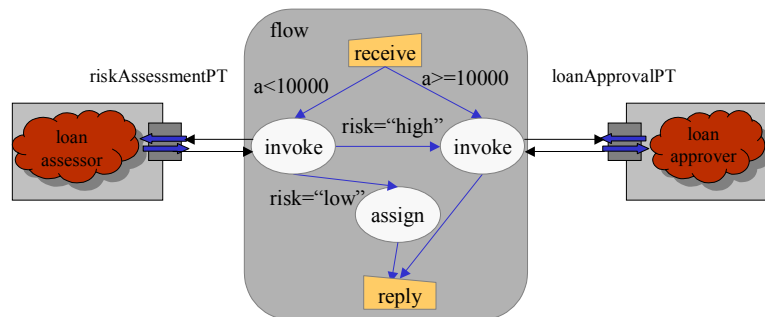
<sequence>
  <receive .../>
  <flow>
    <sequence>
      <invoke .../>
      <while ... >
        <assign> ... </assign>
      </while>
    </sequence>
    <sequence>
      <receive .../>
      <invoke ... >
    </sequence>
  </flow>
  <reply>
</sequence>

```



©IBM
2/03 Curbera, Leymann, Khalaf

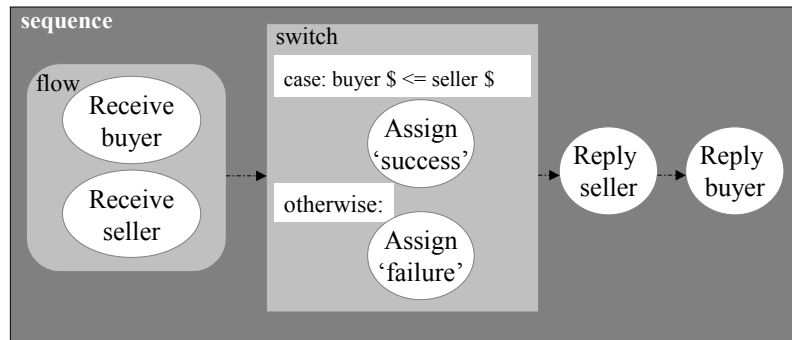
Graph-oriented Authoring Style



- Customer asks for a loan, giving name and amount info.
Two services are involved:
 - A risk assessor which can approve the loan if the risk is low
 - A loan approver which checks the name and decides whether to approve the loan.
- The reply goes back to the customer.

©IBM
2/03 Curbera, Leymann, Khalaf

“Structured” Authoring Style



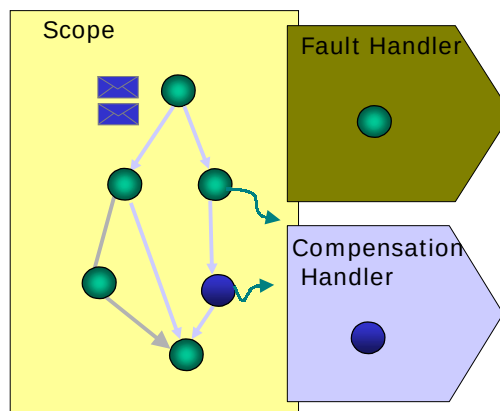
©IBM
2/03 Curbera, Leymann, Khalaf

BPEL Handlers and Scopes

A **scope** is a set of (basic or structured) activities.

Each scope can have two types of **handlers** associated:

- **Fault handlers.** Many can be attached, for different fault types.
- **Compensation handlers.** A single compensation handler per scope.



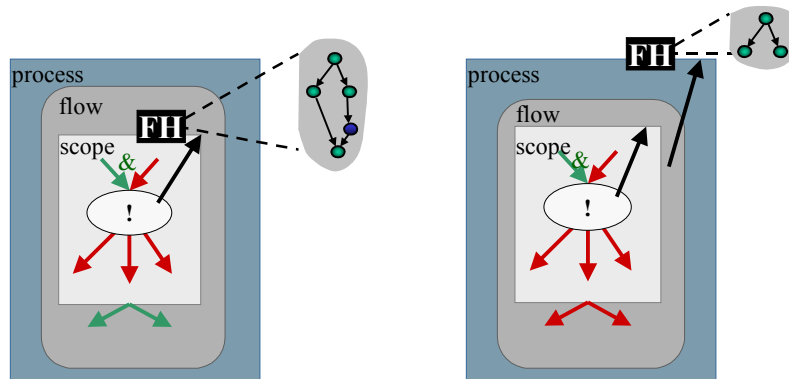
©IBM
2/03 Curbera, Leymann, Khalaf

How Handlers Work

- A compensation handler is used to reverse the work performed by an already completed scope
 - A compensation handler can only be invoked by the fault handler or compensation handler of its immediate enclosing scope
- A fault handler defines alternate execution paths when a fault occurs within the scope.
- Typical scenario:
 1. Fault is thrown (returned by invoke or explicitly by process)
 2. Execution of scope is terminated
 3. Appropriate fault handler located (with usual propagation semantics)
 4. Main execution is compensated to “undo” business effects of unfinished work.

©IBM
2/03 Curbera, Leymann, Khalaf

Fault Handling: scopes and links

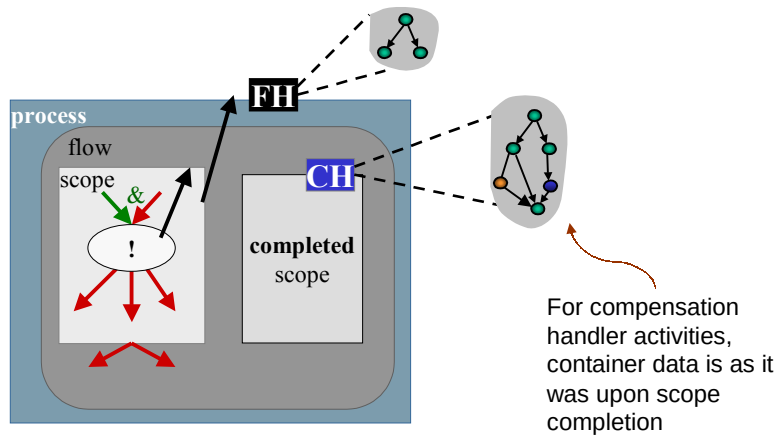


a) directly handling a fault

b) throwing a fault up

©IBM
2/03 Curbera, Leymann, Khalaf

Compensation Example



©IBM
2/03 Curbera, Leymann, Khalaf

What is Correlation?

- BPEL4WS can model many types of interactions:
 - simple stateless interactions
 - Stateful, long running, asynchronous interactions.
- Correlation sets (CSs) provide support for the latter:
 - CSs represent the data that is used to maintain the state of the interaction (a “conversation”).
 - At the process end of the interaction, CSs allow incoming messages to reach the right process instance.
- What is a correlation set?
 - A set of business data fields that capture the state of the interaction (“correlating business data”). For example: a “purchase order number”, a “customer id”, etc.
 - Each set is initialized once
 - Its values do not change in the course of the interaction.

©IBM
2/03 Curbera, Leymann, Khalaf

Defining Correlation Sets

```
<correlationSet name="..." properties="..."/>
```

<!-- A CS is a named set of properties. Properties are defined as WSDL extensibility elements: -->

```
<bpws:property name="..." type="..."/>
```

<!-- A property has a simple XSD type and a global name (Qname) -->

```
<bpws:propertyAlias propertyName="..."
    messageType="..." part="..."
    query="..."/>
```

<!-- A property is "mapped" to a field in a WSDL message type. The property can thus be found in the messages actually exchanged. Typically a property will be mapped to several different message types and carried on many interactions, across operations and portTypes -->

©IBM
2/03 Curbera, Leymann, Khalaf

Using Correlation

```
<receive partner="..." operation="..." portType="..."
    container="...">
```

```
  <correlations>
```

```
    <correlation set="PurchaseOrder"
        initiation="yes"/>
```

```
  </correlations>
```

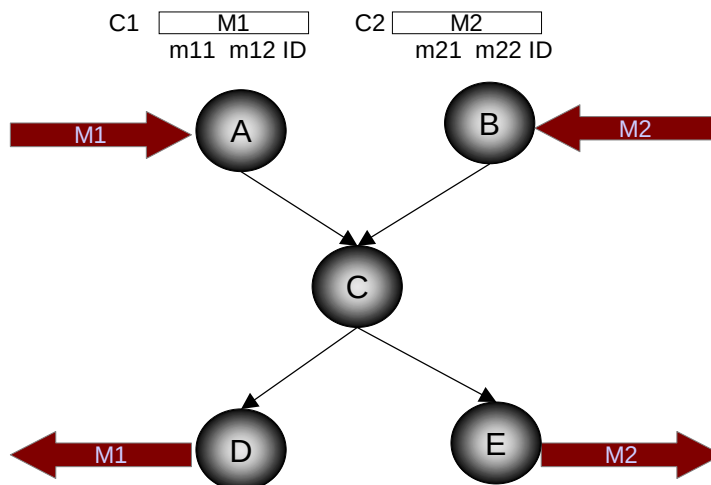
```
</receive>
```

<!-- An input or output operation identifies which correlation sets apply to the messages received or sent. That CS will be used to assure that the message is related to the appropriate stateful interaction.

<!-- A CS is initialized once, in an interaction where the set appears with the "initiation" attribute set to "yes". Its value may never be changed afterward -->

©IBM
2/03 Curbera, Leymann, Khalaf

Multiple Start Correlation



©IBM
2/03 Curbera, Leymann, Khalaf

BPEL4WS Status

- Published August 10, 2002 by BEA, IBM, and Microsoft.
- To be submitted to a standards body.
- Several Java implementations available

©IBM
2/03 Curbera, Leymann, Khalaf

Resources

- BPEL4WS 1.0:
 - <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/>
- BPWS4J Java Implementations:
 - <http://www.alphaworks.ibm.com/tech/bpws4j>
 - <http://www.collaxa.com/>
- Two introductions to BPEL:
 - <http://www.ibm.com/developerworks/webservices/library/ws-bpelcol1>
 - <http://www.ibm.com/developerworks/webservices/library/ws-bpelwp/>