Implementing BPEL4WS: The Architecture of a BPEL4WS Implementation

Francisco Curbera, Rania Khalaf, William Nagy, Sanjiva Weerawarana
IBM, TJ Watson Research Center

Intro to BPEL4WS

- BPEL4WS is a workflow based composition language geared towards Service Oriented Computing and layered as part of the Web Services technology stack.

- The BPEL4WS process model
  - Laid on top of WSDL and core XML specifications.
  - Blends algebraic and graph-based process models
    - Combines the use of structured activities and control links.
  - Recursive: process as a service.
  - Results in multi-party, conversational composition of services based on their abstract descriptions
  - Emerged from merger of WSFL and XLANG concepts.
Example of using Links and Structured Activities

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

© IBM Corp
BPWS4J: A BPEL implementation

- Process model w/ parser and writer
- Independent of runtime; similar to WSDL4J
- A BPEL4WS container:
  - Handles interactions with the outside world, process lifecycle and provides deployment environment.
- Event-driven process interpreter
- Executes the process model
- Tool: Builder to create processes visually
The Container

- Separating the container from the interpreter decouples the execution semantics of the BPEL process model from the Web Services infrastructure it uses.

- Deployment: In the current version, GUI-based deployment binds a process to partners it invokes statically at deployment time.

- Invoking other services:
  - Invocation requests are served by a pool of invocation threads that call back when the response becomes available.
  - Multi-protocol invocation support is provided by WSIF (Web Services Invocation Framework).

- Being invoked by partners: It performs routing of incoming requests, as shown next.
BPEL Processes have implicit lifecycle:
- A process instance is started when a message is sent to one of its startable activities.
- The instance is destroyed once its activities have completed.
- An instance is identified using a set of application specific message fields (correlation sets).

In BPWS4J
- Upon receiving a message, the container checks:
  - The operation being targeted
  - The correlation values of running instances and the fields in the message corr. to correlation sets for a possible match.
  - The process definition for startable activities
- Then, message is put into incoming queue of an existing instance, or a new instance created for it, or it is dropped if neither matches.
The Interpreter

- Event driven, consisting of runnable activities the top level being the process itself.
- The in-memory model is compiled into the runnable model, with activities corresponding nearly one-to-one.
- Unaware of outside world; interacts only with the container.
- Single thread per process instance, w/ simulated parallelism.
Runtime Events

Events are handled at three levels:

- The Process level, mainly interactions with the container:
  - Invocation requests/results
  - Incoming requests and their replies.
  - Alarms
  - XPath evaluations in an assign.
  - Termination

- The Scope level handles faults, compensation, and links.

- The Compound Activity level, where events control enclosed activities.
Activity Lifecycle

- An activity activates once it receives control from its parent.
- An active activity enables once all its incoming links have fired, and they satisfy the join condition. Once enabled, it attempts to run to completion.
- Different forms of fault processing may disable the activity.
Runtime Activities

- Compound and Simple Activities
  - Compound activities are all activities that enclose other activities. They listen to lifecycle events from their enclosed activities, and control them.
  - Control defined thru nesting in complex activities and due to explicit control links.
Faults

When an activity faults:

- It disables itself:
  - Fires a disabled event
    - Sets its status to disabled, and cleans itself up.
    - Disables all its enclosed activities, starting with scopes.
    - Events it receives while it’s disabled are ignored.
  - Sends all its links out “false”
- Fires the fault to its scope
  - The scope looks for a handler. If it find one it runs it.
  - Otherwise, the scope throws the fault up
Using faults for Dead Path Elimination

- **DPE**: ensures completion in graph-oriented process models that always synchronize on joins.

- **In BPEL4WS:**
  - BPEL handles it through a built-in fault, "joinFailure"
  - An activity can explicitly suppress it
    - equivalent to wrapping it in a scope with an empty fault handler for joinFailures.
  - When join failure occurs, it is caught and the outgoing link is set to false.
  - The Join failure propagates along process graph, with effects equivalent to DPE.

- **BPWS4J** natively handles suppressing the join failure by desugaring the suppressJoinFailure attribute.
Simple flow example

Receive Message

Invoke

Send Reply
Thread execution for simple rcv-inv-rpl process

Web App Thread
- Get Message
- Find/create instance
- Deliver incoming message to queue
- Wait for reply

Flow Instance Thread
- Activate
- Activate Receive
- Complete Receive
- Activate Invoke
- Deliver invocation
- Request to queue

Invoker Thread
- Do Invocation
- Post “done” to queue

Return reply msg to WSGW

Update Flow
Manager state
Die
Conclusions and Future Work

- Illustrate a lightweight implementation of BPEL4WS.
- Highlight Challenges in Supporting BPEL4WS:
  - Implicit lifecycle model
  - Combining graph and calculus based approaches
  - Recovery: Compensation and fault handling mechanisms
  - Composition in the abstract with flexible binding capabilities
- BPWS4J provided extremely valuable insight into the deployment and runtime operation of BPEL4WS processes.
- Future Work
  - Additional binding possibilities, support of WS-Addressing, multi-protocol on the incoming channels.
Acknowledgement and Contacts

BPWS4J was architected and created as a group effort by Francisco Curbera, Rania Khalaf, William Nagy, Sanjiva Weerawarana, Matthew Duftler, and Nirmal Mukhi.

Questions?
  - rkhalaf@watson.ibm.us