BPEL for Programmers and Architects

Paul Brown <prb@fivesight.com>
Maciej Szefler <mbs@fivesight.com>

2003-12-16

Goal

The goal for this talk is to introduce BPEL and place it in the context of other technologies:

- What it is.
- Why it is useful.
- Who should use it and how it should be used.

Various details of syntax and philosophy will be covered along the way.

What?

BPEL

BPEL is the (Web Services) Business Process Execution Language, currently under standardization at OASIS.

BPEL is an XML language for defining the composition of (web) services into new (web) services.

What?

Standards Context

BPEL relies on WSDL for port types, messages/parts, and as a container for additional metadata, but BPEL itself is independent of protocol.

XPath 1.0 with some extension functions provides the basic expression language.

There is a light dependency on XML Schema, although other typing systems could be used.

What?

Abstract / Executable

BPEL supports both abstract processes and executable processes.

Abstract processes are useful for specifying expected protocols and publicly visible behaviors without too much detail. Some static analysis techniques may apply.

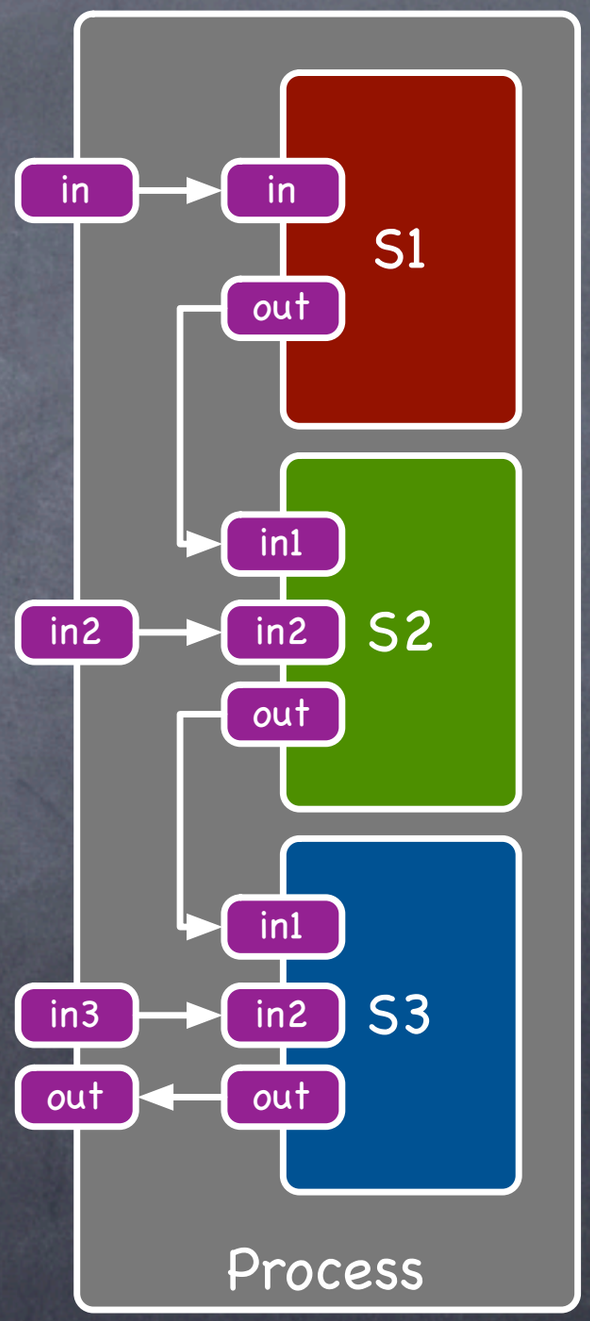
Executable processes contain enough detail to fully specify execution.

What?

Example: Statement

Three services play constituent roles in a complex computation.

The services are executed in sequence, and an input to the process and the output from a service provide the input for the next service in order.



Example: BPEL Sketch

```
<bpel:process name="simpleExampleProcess" targetNamespace="foo"
  xmlns:bpel="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
  xmlns:links="wsdl://elsewhere">

[... declarations omitted ... ]

    <bpel:sequence>
        <bpel:receive partnerLink="invoker" portType="links:in"
          operation="computation" variable="input" />
        <bpel:assign><bpel:copy>
            <bpel:from variable="input" part="argument1" />
            <bpel:to variable="input1" part="root" />
        </bpel:copy></bpel:assign>
        <bpel:invoke partnerLink="service1" portType="links:service1Input"
          operation="computation1" inputVariable="input1" />

[... more BPEL omitted ... ]

        <bpel:reply partnerLink="invoker" portType="links:in"
          operation="computation" variable="result" />
    </bpel:sequence>
</bpel:process>
```

Why?

BPEL does...

BPEL deals explicitly with the functional aspects of business processes:

- control flow (branch, loop, parallel)
- asynchronous conversations and correlation
- non-determinism
- long-running, nested units of work, faults, and compensation

What?

Basic Control Flow

BPEL provides the usual branching and looping control flow constructs.

A <sequence> executes activities in serial order.

A <switch> executes at most one alternative based on expressions specified on child <case> elements with an optional <otherwise>.

A <while> loops through activities while a variable's value is true.

What?

Fancy Control Flow

BPEL provides a parallel control construct through the <flow> activity.

Basic synchronization is achieved through named "links" with <source> and <target> on separate threads.

More complex synchronization is achieved through "join" expressions composed of link statuses and boolean operations (&& and ||).

What?

Correlation

BPEL correlates messages based on properties referenced in a <correlationSet>.

Multiple properties can be combined into a composite correlation key.

Properties are typed by XML Schema simple types and bound ("aliased") via XPath expressions to locations in message parts.

What?

Non-determinism

A <pick> activity waits:

- for a message specified by an <onMessage> child element, where correlation allows a specific process instance to be addressed.
- for an amount of time or until a time, specified with an <onAlarm> child element.

What?

Execution Context

A <scope> is an execution context for the contained activities, and a <process> is, itself, a <scope>.

A <scope> defines local variables and can catch and handle either specific faults or all faults that occur with it.

What?

Events on Scopes

A <scope> can have an <eventHandlers> child that behaves like a <pick> in that it can respond to <onMessage> or <onAlarm>, except that the triggered activities are executed concurrently with the <scope>.

Typical use cases include interrogation of a specific instance, receiving notifications, or sending notifications.

What?

Compensation

There is no way to roll-back an already executed activity, since a process may execute over hours or even months.

Instead, BPEL provides the option to attach a `<compensationHandler>` that can be run to "undo" the activities of a scope.

A `<compensationHandler>` is activated only after normal completion of a scope.

What?

Complex Compensation

The <compensate> activity can execute the <compensationHandler> of an immediate child <scope> by name or, by default, of all immediate children in reverse order of completion.

The execution of a <while> loop creates one <compensationHandler> per iteration.

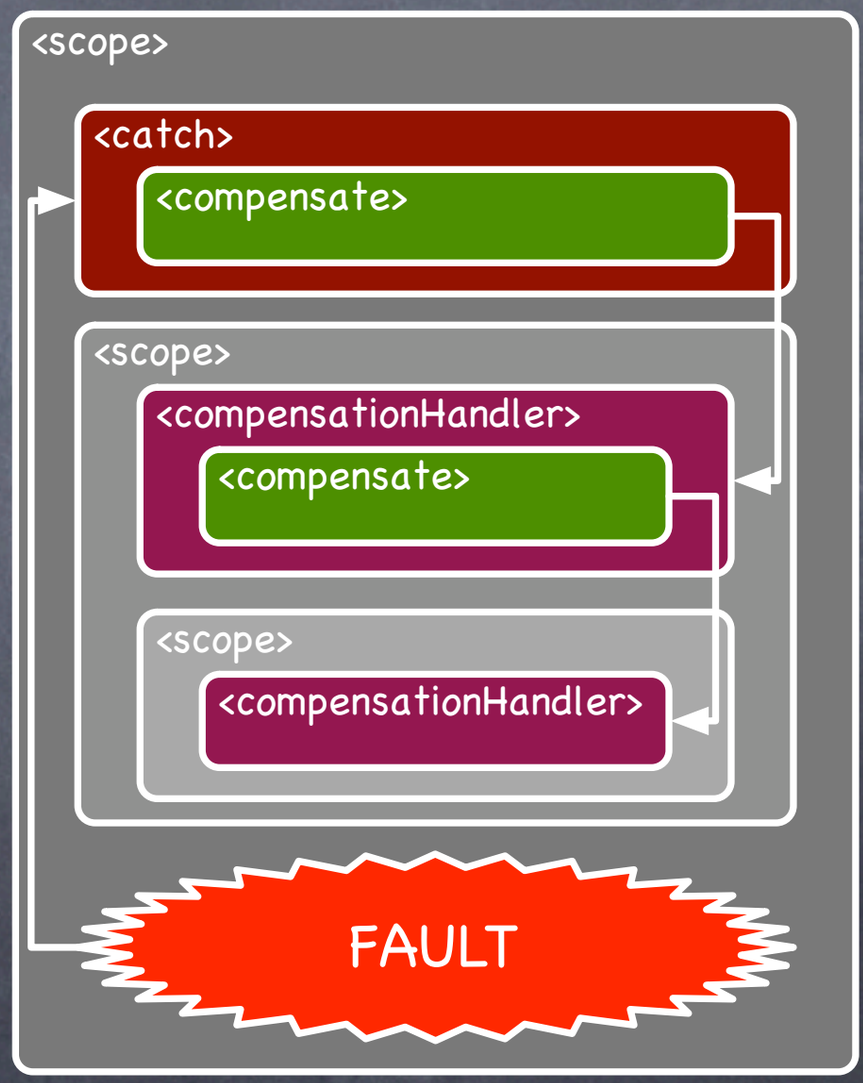
The <compensate> activity can only occur within a <compensationHandler> or during fault handling.

Compensation Example

1. A fault in the top-level <scope> triggers a <catch>.

2. The <catch> invokes the <compensationHandler> in the middle <scope>.

3. The middle scope's <compensationHandler> invokes the <compensationHandler> in the innermost child <scope>.



Why?

BPEL is not...

BPEL is not workflow: there are no explicit abstractions for people, roles, work items, or inboxes in BPEL (among other things).

BPEL is also not BPM: no specified data model for measurement, reporting, or management.

Why?

BPEL is not...

BPEL is not integration: there is no explicit support for transformation, semantic interpolation, or specific protocols.

BPEL is not all-encompassing: there are some patterns that are difficult to model with BPEL.

Why?

Value Proposition

BPEL (i.e., an implementation) abstracts the common, non-process-level concerns of business process execution.

As a service-tier composition language, BPEL is a key ingredient in a service-oriented architecture.

BPEL provides a deployment-level interface for higher-level tools (e.g., modeling).

Who and how?

The Process of Process

"Low-hanging" innovation does not create sustainable competitive advantage.

Complexity will remain an unavoidable part of the enterprise technology landscape. Managing and minimizing that complexity will create leaders.

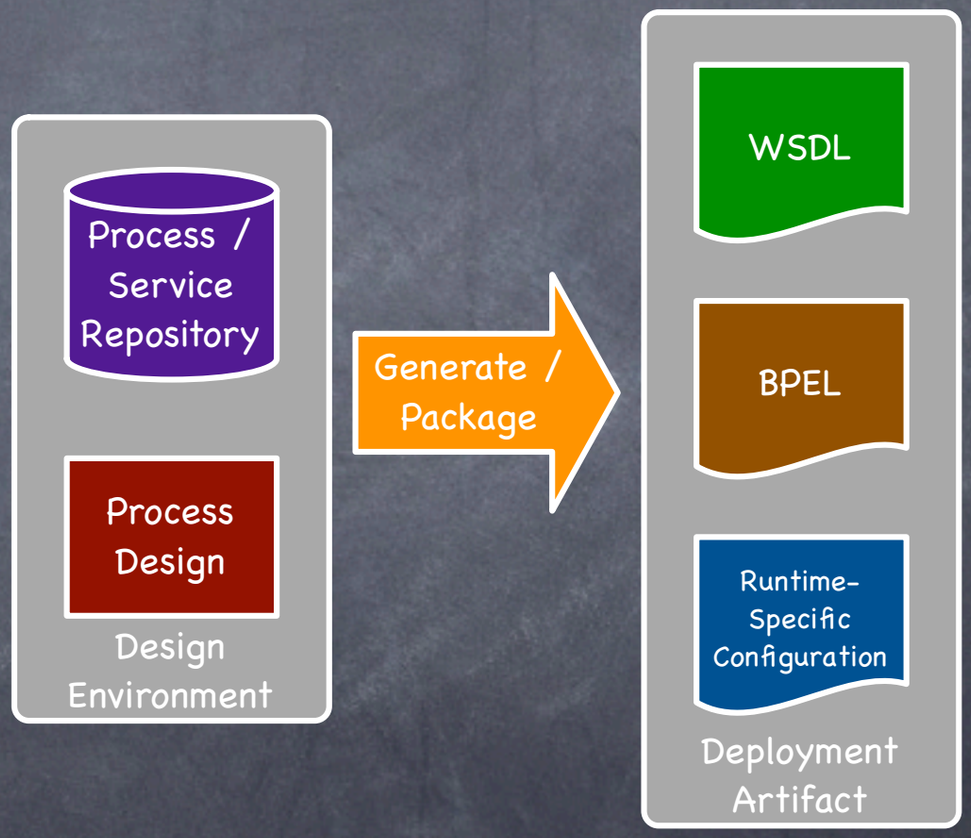
Defining the organizational processes around BPEL, i.e., who uses BPEL and how, is of paramount importance to the role and success of service-oriented architecture.

Who and how?

Modeling

Multiple modeling environments and standards are emerging to provide a high-level visual language for compilation into BPEL.

Business Process Modeling Notation ("BPMN", from BPMI) and Business Process Definition Metadata ("BPDM", from OMG) are both notable in this regard.

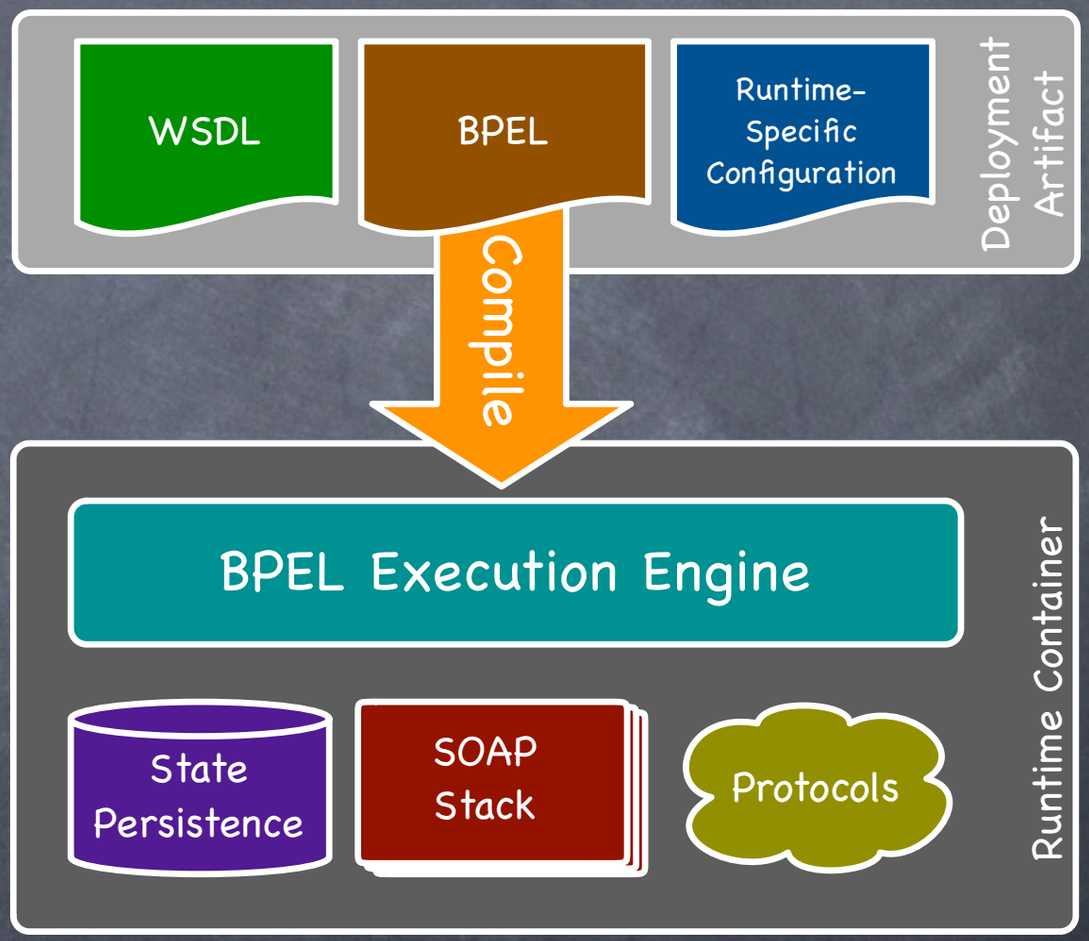


Who and how?

Runtime

Engines will come in a variety of form factors and use different techniques for runtime representation of processes, including:

- interpretive
- code generation
- OO models of activities
- reduction engines based on π-calculus and related concepts.



Many Roles

BPEL will be a programmer's tool for low-level service orchestration and integration-oriented tasks.

BPEL will be an artifact of high-level modeling and process design environments.

The runtime, state persistence, and SOAP stack for the BPEL execution engine will expose hooks for security, management, monitoring, and analysis functionality.

When?

To Do Now

Build a deeper understanding of web services, including the SOAP stack, management, coordination, and security.

Learn more about BPEL and consider use cases for your organization. Process and model repository? One-off integration applications? Legacy/host enablement? Best-of-breed SOA or all-in-one product?

Watch for the official specification release in 2004.