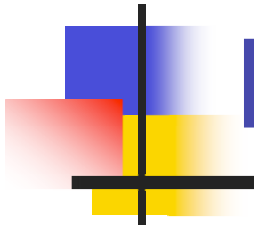
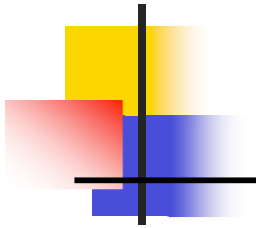


Business Processes

Noel J. Bergman

DevTech®



Session Overview

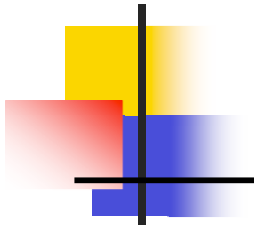
Big Business has a secret weapon in its IT arsenal. That secret weapon, heretofore costly, has allowed IT departments to respond faster, be more nimble, be more effective, and collaborate better with non-IT parts of the enterprise. That secret weapon allows its users to better understand and document how their business works. What is this secret weapon? What can possibly deliver these benefits?

This secret weapon is the Business Process. Business Processes can capture how a business performs a business task in a formalized, executable manner. Business Analysts can model and evaluate changes in processes. Business Processes can be standardized and exchanged amongst partners. Business Processes execute in run-time environments designed to support long-running, asynchronous processes.

This presentation dovetails with Doug Tidwell's "The All-Singing, All-Dancing Business Process" talk. In Doug's talk, you'll learn to use Apache ODE, Tuscany (SCA and SDO) and XForms. In this talk, you will learn more details about WS-BPEL, the standard for describing Business Processes.

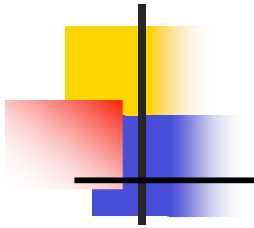
Specific topics to be covered in this session include standard WS-BPEL activities, flows, transactions and compensation, event handling, fault handling, and more.

PLEASE ASK QUESTIONS! ☺



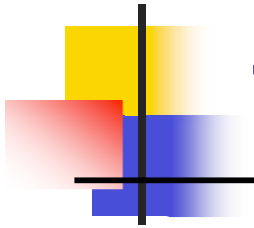
Realistic Expectations

- A WS-BPEL course is really a very full week.
- The core topics of basic activities, structured activities, exception handling, compensation handling, and event handling are each a good 90 minute lecture, plus a lab.
- This session simply attempts to serve as a primer, giving you the core ideas, and setting you up to better understand WS-BPEL, and what it can do for you.
  - Resource material is provided for future reading.



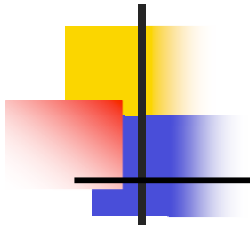
Background

- Businesses of all types and sizes want to be flexible, cost-effective, and responsive to changing conditions.
- I/T staff are often tasked with implementing the various operational workflows that represent a company's way of doing business.
- I/T staff are generally not business domain experts. Business domain experts are generally not technology experts.
- Typical programming languages are not ideal for implementing business logic, especially not when we add flexibility, cost-effectiveness and responsiveness to the requirements. And most definitely not when the operational process is long-lived and has asynchronous aspects.
- Business processes are often poorly documented, untested until put into production, and non-optimal.
- I/T implementations of business processes are often rigid, hard to monitor, and buggy.



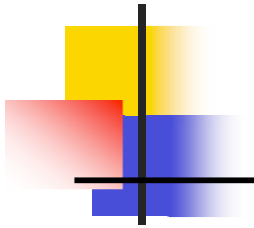
The SOA Panacea

- No problem, we'll adopt SOA! We have no idea what it means, but it solves all problems, right?
- Uh, no.
- No, c'mon we read that in (check all that apply):
  - ❑ Our vendor's whitepapers
  - ❑ _____ magazine
  - ❑ http://_____
  - ❑ Our consultant's proposal
  - ❑ Other: _____
- Sorry, no. Really. No. Not if they left out the rest of the story.



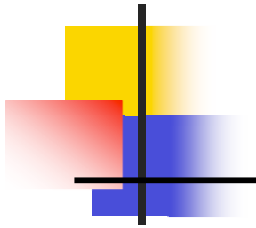
What SOA Addresses

- SOA: *Service* Oriented Architecture
- A *Service* is self-contained functionality.
- We expose *Services* in standard ways.
  - A *Web service* is one common, standard, way to expose a *Service*; the use of an Enterprise Service Bus (ESB) enables other interface technologies.
  - *Web Services* are described using WSDL
- But *Services* are not enough … they do not address the need to loosely couple business logic.
- In order to gain business flexibility, you need to introduce *Business Processes*.



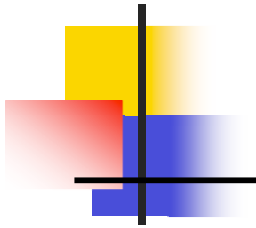
Business Processes

- *Business Processes* move the level of discourse and responsibility for process definition to process experts.
- Business Processes centralize orchestration amongst supporting services.
  - "[WS-BPEL] allows you to create complex processes by creating and wiring together different activities that can, for example, perform Web services invocations, manipulate data, throw faults, or terminate a process. These activities may be nested within structured activities that define how they may be run, such as in sequence, or in parallel, or depending on certain conditions."
- Business Processes are exposed as (Web) Services and use (Web) Services, preserving the SOA acronym.
  - Why (Web)? As far as WS-BPEL is concerned, a process is exposed as a Web Service and invokes Web Services, but by using an ESB, we can bridge to whatever other implementation(s) exist.



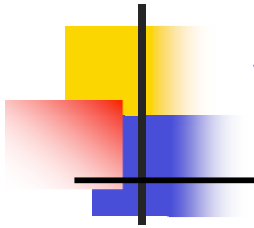
SOA with Processes

- An ESB provides loose coupling with respect to service interface, protocol and routing.
- A Business Process provides loose coupling for Business Logic.
- Perhaps, as Business Processes form the true core of your SOA solution, you should be thinking of it as a *Business Process Oriented Architecture.*



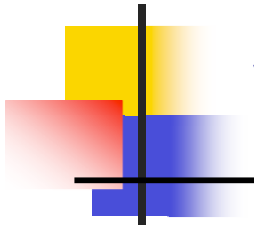
Development Lifecycle

- We recognize several distinct phases of a business process' development lifecycle:
  - Modeling – describe and simulate
    - This would typically involve a Process Expert using a modeling tool.
  - Development – complete the details
  - Deployment – get it running
  - Monitoring – monitor status/performance
    - Completes the loop, allowing management to see how our business is performing.
- Commercial vendors offer products to fill all of these areas.
- Open Source offerings are currently best in the middle phases. For the moment, we fall down on modeling and monitoring.



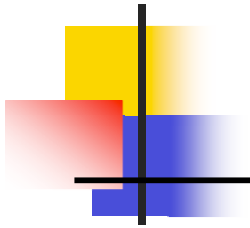
What is WS-BPEL?

- WS-BPEL is an Oasis Standard for Business Processes.
  - WS-BPEL is version 2; version 1 was known as BPEL4WS.
- WS-BPEL is an XML language.
- WS-BPEL is a programming language executed by a specialized engine.
- As previously noted, we will describe the interface to our WS-BPEL process as a Web service, and we will describe the interfaces to any other services that we invoke also as Web services.



WS-BPEL Advantages

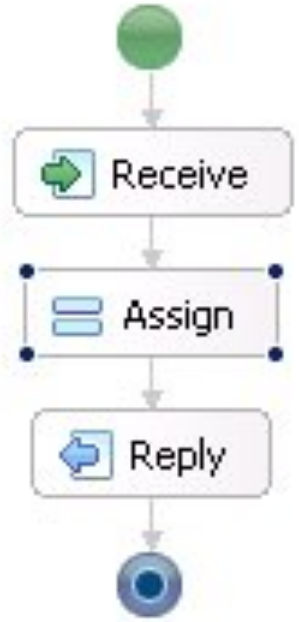
- Designed for long-running, business processes.
- WS-BPEL supports:
  - Process level persistent state.
  - Transactional behavior as appropriate.
  - A compensation model to enable adjustments in long-running processes, where atomic transactions are not feasible.
  - Concurrent execution paths.
  - Asynchronous messaging.
  - Exception (fault) handling.

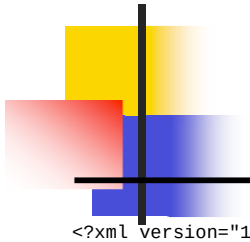


WS-BPEL Representation

- A WS-BPEL Business Process is defined in an XML document.
  - There are graphical editors for WS-BPEL.
    - Few people, and even fewer people of the type – Business Analysts – whom we want defining business processes are XML experts.
    - Would anyone, no matter how XML fluent, really want to be visualizing workflow in XML?
  - Although the XML for WS-BPEL is standardized, graphical representations are vendor specific.
    - IBM has distinctly different representations for their Business Process Modeling tool and their Business Process Integration tool.
- A good graphical WS-BPEL editor will insulate you from the XML representation, and in many cases you won't need to know the details. But we will occasionally want to look at the XML schema in order to identify technical aspects.

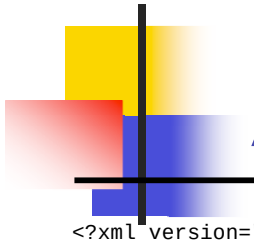
A Simple Process





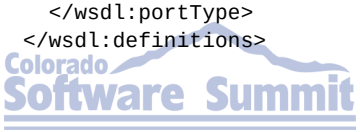
Not-So-Simple Document

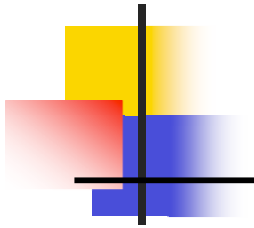
```
<?xml version="1.0" encoding="UTF-8"?>
<bpws:process xmlns:bpws="http://schemas.xmlsoap.org/ws/2004/03/business-process/" xmlns:ns="http://Sample/com/devtech/bpel/HelloArtifacts"
      xmlns:ns0="http://Sample/com/devtech/wsdl/HelloWorld" xmlns:wpc="http://www.ibm.com/xmlns/prod/websphere/business-process/6.0.0/"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" expressionLanguage="http://www.ibm.com/xmlns/prod/websphere/business-process/expression-lang/java/6.0.0/"
      name="Hello" suppressJoinFailure="yes" targetNamespace="http://Sample/com/devtech/bpel" wpc:displayName="Hello" wpc:executionMode="longRunning"
      wpc:id="1" wpc:validFrom="2008-10-14T19:35:39">
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="../wsdl/HelloWorld.wsdl" namespace="http://Sample/com/devtech/wsdl/HelloWorld"/>
  <bpws:import importType="http://schemas.xmlsoap.org/wsdl/" location="HelloArtifacts.wsdl" namespace="http://Sample/com/devtech/bpel/HelloArtifacts"/>
  <bpws:partnerLinks>
    <bpws:partnerLink myRole="HelloWorldRole" name="HelloWorld" partnerLinkType="ns:HelloWorldPLT"/>
  </bpws:partnerLinks>
  <bpws:variables>
    <bpws:variable name="input" type="xsd:string" wpc:id="2"/>
    <bpws:variable name="output" type="xsd:string" wpc:id="3"/>
  </bpws:variables>
  <bpws:sequence name="HiddenSequence" wpc:id="1073741828">
    <bpws:receive createInstance="yes" name="Receive" operation="hello" partnerLink="HelloWorld" portType="ns0:HelloWorld" wpc:displayName="Receive"
      wpc:id="5">
      <wpc:output>
        <wpc:parameter name="input" variable="input"/>
      </wpc:output>
    </bpws:receive>
    <bpws:assign name="Assign" wpc:displayName="Assign" wpc:id="7">
      <bpws:copy>
        <bpws:from variable="input"/>
        <bpws:to variable="output"/>
      </bpws:copy>
    </bpws:assign>
    <bpws:reply name="Reply" operation="hello" partnerLink="HelloWorld" portType="ns0:HelloWorld" wpc:displayName="Reply" wpc:id="6">
      <wpc:input>
        <wpc:parameter name="output" variable="output"/>
      </wpc:input>
    </bpws:reply>
  </bpws:sequence>
</bpws:process>
```



And Its Companion WSDL

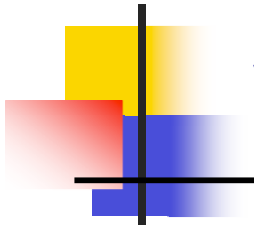
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:tns="http://Sample/com/devtech/wsdl/HelloWorld"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="HelloWorld" targetNamespace="http://Sample/com/devtech/wsdl/HelloWorld">
    <wsdl:types>
    <xsd:schema targetNamespace="http://Sample/com/devtech/wsdl/HelloWorld" xmlns:tns="http://Sample/com/devtech/wsdl/HelloWorld"
      xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:element name="hello">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="input" nillable="true" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
      <xsd:element name="helloResponse">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="output" nillable="true" type="xsd:string"/>
          </xsd:sequence>
        </xsd:complexType>
      </xsd:element>
    </xsd:schema>
  </wsdl:types>
    <wsdl:message name="helloRequestMsg">
    <wsdl:part element="tns:hello" name="helloParameters"/>
  </wsdl:message>
    <wsdl:message name="helloResponseMsg">
    <wsdl:part element="tns:helloResponse" name="helloResult"/>
  </wsdl:message>
    <wsdl:portType name="HelloWorld">
    <wsdl:operation name="hello">
      <wsdl:input message="tns:helloRequestMsg" name="helloRequest"/>
      <wsdl:output message="tns:helloResponseMsg" name="helloResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```





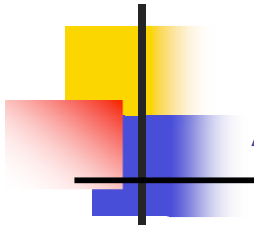
The Structure of a WS-BPEL Business Process

- A WS-BPEL Process will consist of two related files:
  - WS-BPEL – contains the actual process definition.
  - WSDL – describes the interface of the process and any invoked processes.
- A WS-BPEL Document contains the following items, as necessary to describe the process:
  - **Partner Links** – define interfaces between a process and its partners
  - **Variables** – hold a process' state during execution
  - **Activities** – the WS-BPEL instruction set; defines the business logic
  - **Correlation Sets** – used to match messages to process instances
  - **Compensation Handlers** – perform "undo" activities for work that has already been successfully completed.
  - **Event Handlers** – perform work based on exposing an interface for asynchronous messages.
  - **Fault Handlers** – perform work to recover from an exception thrown during processing.



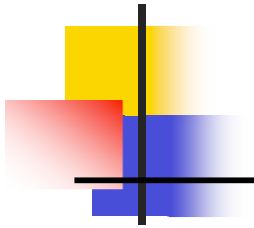
WS-BPEL XML Outline

- <process>
  - <extensions>
  - <imports>
  - <partnerLinks>
    - <partnerLink>
  - <messageExchanges>
    - <messageExchange>
  - <variables>
    - <variable>
  - <correlationSets>
    - <correlationSet>
  - <faultHandlers>
    - <catch>
    - <catchAll>
  - <eventHandlers>
    - <onEvent>
      - <correlations>
        - <correlation>
      - <fromParts>
        - <fromPart>
      - <scope> … </scope>
    - <onAlarm>
      - <for> (duration) OR <until> (deadline)
      - <repeatEvery>
      - <scope> … </scope>
  - *Activity* (Each business process has one main activity)



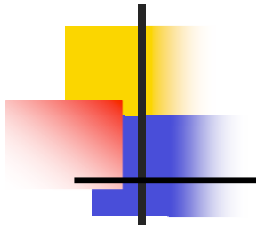
Analysis Of Our Sample

- Our sample Business Process consists of three simple activities:
  - A `receive` activity
  - An `assign` activity
  - A `reply` activity
- These activities are nested inside of a `sequence` activity, which is the main activity for our WS-BPEL document.
- There are two global variables, `input` and `output`, which are of `xsd:string` type.
- The `receive` activity loads the `input` variable from the input message.
- The `assign` activity copies the `input` variable to the `output` variable.
- The `reply` activity generates the output message from the `output` variable.
- The interface to our process is declared by the `partnerLink`, which references our WSDL for the actual interface definition.



Partner Links

- Partner Links provide a reference to the interface for a service.
  - A `partnerLink` element in a WS-BPEL document refers to a `partnerLinkType` element in the matching WSDL.
- A Partner Link can have two roles.
  - `myRole`: defines the interface to the Business Process as part of the partnership.
  - `partnerRole`: defines the interface to an external service as part of the partnership.
- We need a Partner Link to reference each interface into the Business process.
- We need a Partner Link to reference the interface for each external service invoked by the service.
  - Where such a service has a callback into the Business process, the Partner Link will have both roles.
- A GUI WS-BPEL editor will largely insulate you from the syntax and nuances of working with Partner Links.

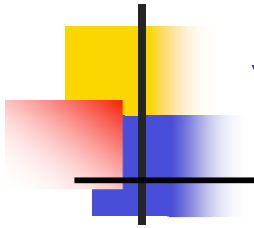


partnerLinkType

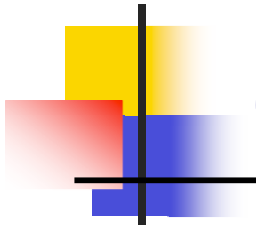
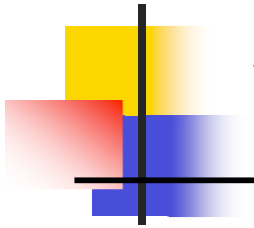
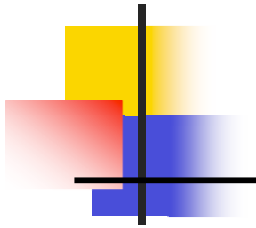
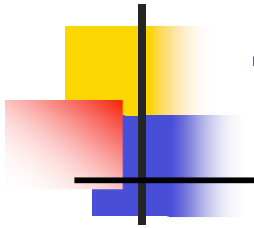
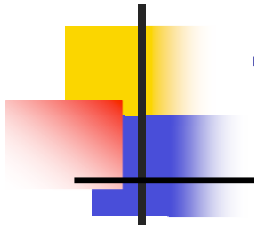
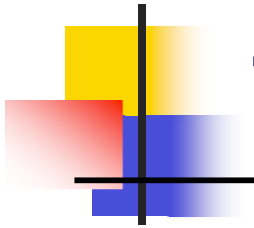
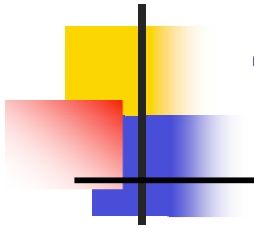
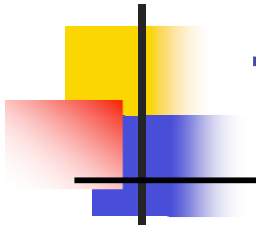
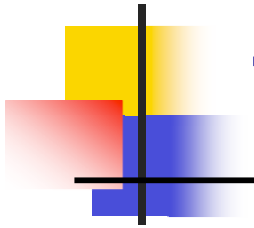
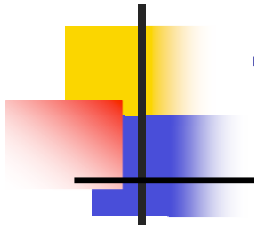
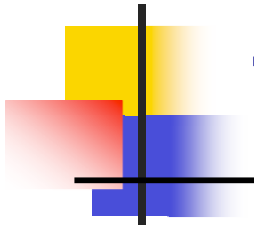
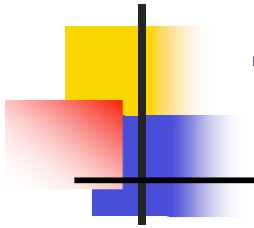
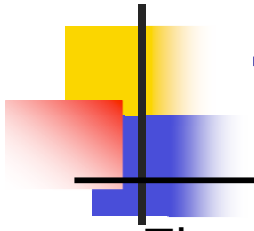
- Our WSDL document appeared to be missing the partnerLinkType element. In truth, the WSDL file we looked at is purely for defining the Web service interface, and is not "polluted" with WS-BPEL-isms.
- We'll use a second WSDL file with our process deployment:

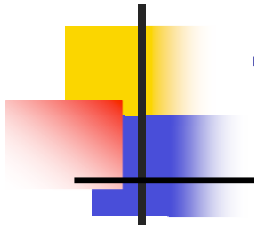
```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns:plnk="http://schemas.xmlsoap.org/ws/2004/03/partner-link/"
   xmlns:tns="http://Sample/com/devtech/bpel/HelloArtifacts"
   xmlns:wsdl0="http://Sample/com/devtech/wsdl/HelloWorld"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema" name="Hello"
   targetNamespace="http://Sample/com/devtech/bpel/HelloArtifacts">
  <plnk:partnerLinkType name="HelloWorldPLT">
    <plnk:role name="HelloWorldRole">
      <plnk:portType name="wsdl0:HelloWorld"/>
    </plnk:role>
  </plnk:partnerLinkType>
  <wsdl:import location="../wsdl/HelloWorld.wsdl"
    namespace="http://Sample/com/devtech/wsdl/HelloWorld"/>
</wsdl:definitions>
```

- You can see how a partnerLink in the WS-BPEL document references a partnerLinkType in the WSDL, which references a specific WSDL interface (portType) in the included WSDL.



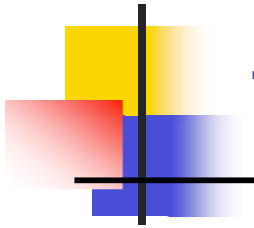
Variables

- We can declare variables in WS-BPEL at the global scope and inside of <scope> elements.
- Our sample contains two variables:
  - ```
    <bpws:variables>
       <bpws:variable name="input"
                      type="xsd:string" wpc:id="2"/>
       <bpws:variable name="output"
                      type="xsd:string" wpc:id="3"/>
     </bpws:variables>
    ```
  - These variables are of type `xsd:string`.
- Variables can be created using XML Schema types, XML Schema elements, and WSDL message types.
- Data in variables can be accessed using a "binding." The default binding for WS-BPEL is XPath.
- The name of a variable must not have a '.' in it, due to the fact that this is a delimiter in XPath.
    ```



Correlation Sets

- Correlation Sets are used to route an incoming message to the correct instance of an existing, long-running, business process.
  - An example Correlation Set could consist of a unique Order ID.
- One use for a Correlation Set would be to route messages for events, such as a cancellation event, to the appropriate process instance.
- A Correlation Set has a name and a set of properties.
  - Each Property consists of a name and a data type.
  - Message parts are mapped to properties.
- At run-time, values in the received message will be compared to see to which process instance the message should be routed, so we must initialize the Correlation Set before using it.



WS-BPEL Activities

- The bulk of a Business Process consists of instructions known as “activities” in WS-BPEL parlance.
- There is a set of standard activities, which is broken down into three categories:
  - Basic – unstructured, standalone, activities.
    - Exception related – activities that can only be used in a fault or compensation handler.
  - Structured – act as containers for other activities, such as a sequence, flow, scope, loop, etc.
- Vendors sometimes add their own. For example, IBM added activities such as <humanTask> and Java <snippet>.
- Activities exist to handle messages, workflow, service invocation, exception handling and compensation. WS-BPEL does not provide computational activities beyond what we can do with the <assign> activity – we should use services for that kind of behavior.



Standard Activities

- Basic Activities
  - <receive>
  - <reply>
  - <invoke>
  - <assign>
  - <throw>
  - <rethrow>
  - <compensate>
  - <compensateScope>
  - <exit>
  - <wait>
  - <empty>
  - <validate>
  - <extensionActivity>
- Structured Activities
  - <if>
  - <while>
  - <repeatUntil>
  - <forEach>
  - <pick>
  - <flow>
  - <scope>
  - <sequence>



The <receive> Activity

- The <receive> activity is used to receive a message into the Business Process.
  - It is a blocking activity
  - It can start a new process instance
  - It can restart a process that was blocked waiting for a message
  - It can be associated with correlation information, either to initialize or use for matching an existing process instance.
- The incoming message reflects an operation on a business process interface.
- The request can be synchronous or asynchronous.
- We can populate a variable to hold the incoming message.



The <reply> Activity

- The <reply> activity is used to send out a message.
  - The message can be a response or a fault message (remember, we're talking WSDL here ☺)
  - Typically used to send a response to a synchronous request
- We would populate the output message from a variable.



The <invoke> Activity

- The <invoke> activity is used to invoke a one-way or request-response operation on an interface offered by a partner (why do you think that we call them “partner links”? ☺)
- As appropriate for SOA, the operation is that of a Web service, which could be another business process, or some other form of service implementation.
- The operation will populate the outgoing message from a variable, and optionally populate a variable from a response message.
- The <invoke> activity can be associated with its own:
  - Fault handler
  - Compensation handler
  - Correlation information



The <assign> Activity

- The <assign> activity is used to set values on variables.
- The values can be copied from source variables.
- An expression language (XPath is the default) can be used for simple computation when assigning values.
- A single <assign> activity can perform multiple operations, so you just need a single <assign> activity to handle an entire set of operations at a given point in the workflow.
- An advanced use allows the endpoint references associated with partner links to be manipulated.



The <wait> Activity

- The <wait> activity stops the execution of the business process, either:
  - For a duration.
  - Until a specific deadline is reached.



The <pick> Activity

- The <pick> activity is similar to the <receive> activity, but is used when multiple operations are supported at that point in the workflow.
  - The <pick> activity is known as Receive Choice in IBM's terminology.
  - The <pick> activity can be used to initiate a process instance, or within an existing process instance.
- You might have noticed that <pick> is a structured activity.
  - Each operation on the interface is represented by an internal <onMessage> element.
  - <pick> acts as a container of <onMessage> elements, each of which acts as a container for the kind of information associated with a <receive>, plus an associated activity to be executed when that <onMessage> is matched.
- Another difference is that unlike <receive>, <pick> supports a timeout path, using <onAlarm>.
  - The timeout is expressed in the same manner as with the <wait> activity.



The <exit> Activity

- The <exit> activity immediately terminates a process, without any termination handling, fault handling, or compensation handling.



The <sequence> Activity

- The <sequence> activity contains a set of nested activites, each of which will be executed one at a time, in order.
- Our sample WS-BPEL, earlier, used a <sequence> as the activity.



The <scope> Activity

- The <scope> activity allows us to nest partner links, variables, correlation sets, handlers (of all types), and an activity.
- As usual, we'll often use a structured activity for the single activity nested inside a scope.



The <if> Activity

- The <if> activity is one of two ways in which conditional branching logic is supported by WS-BPEL.
- The easiest way to understand the <if> activity is to view its XML Schema definition:
  - ```
    <if standard-attributes>
    standard-elements
    <condition expressionLanguage="anyURI"?>boolexpr</condition>
    activity
    <elseif>*
      <condition expressionLanguage="anyURI"?>boolexpr</condition>
      activity
    </elseif>
    <else>?
      activity
    </else>
    </if>
    ```
- Notice that each conditional branch takes a single activity. In order to have multiple activities, you would need to use a structured activity, such as a <sequence>.
    ```



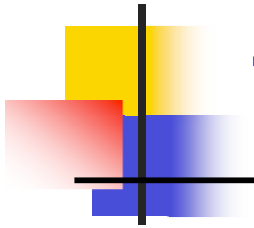
The <while> Activity

- The <while> activity loops as long as a condition evaluates to true, executing a single activity.
- From the <if> activity, we have seen how a <condition> is defined, and how a structured activity such as <sequence> can be used as the single activity.



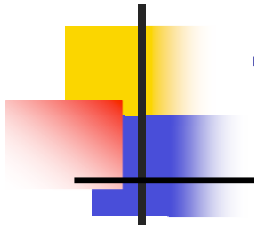
The <repeatuntil> Activity

- The <repeatuntil> activity executes a single activity, looping as long as a condition evaluates to true.
- This is the same behavior as the <while> activity, except it executes first and checks the condition second.



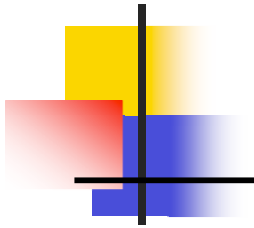
The <forEach> Activity

- The <forEach> activity provides a counting loop construct.
- Unlike the other structured activities, <forEach> contains a <scope>. Each time through the loop, the scope will be executed.
  - An option exists to execute the <scope> instances in parallel rather than iteratively.
  - Although <forEach> normally executes against its loop counter, an optional completion condition can be used to preemptively terminate the loop.



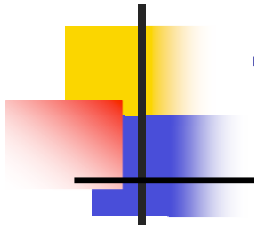
The <flow> Activity

- The <flow> activity acts as a container for activities, which will be executed in parallel.
- It is not uncommon for a <flow> to have a set of nested <sequence> activities.



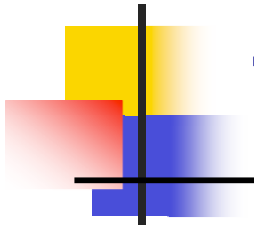
Links and Link Conditions

- The links between WS-BPEL activities can be associated with conditions.
  - Conditional execution
    - A condition is associated with the link, which is traversed if the condition evaluates to true.
    - If multiple links leaving an activity evaluate as true, each path can be taken in parallel.
  - Join behavior when links merge
    - Indicates the condition for waiting before continuing down the converged path.



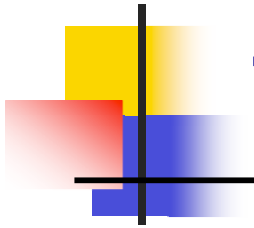
The <empty> Activity

- The <empty> activity is used:
  - As a synchronization point within a business process that has parallel paths of execution.
  - With a <catch> where we want to catch and suppress the fault.
  - A placeholder to be replaced later.
- There are some implicit uses, such as:
  - If a WS-BPEL document uses an extended, non-standard, activity, that activity must be registered as an extension in the document.
  - If the WS-BPEL engine does not recognize an extension **and** the extension is not declared with mustUnderstand="yes", all uses of it must be treated as an <empty> activity.



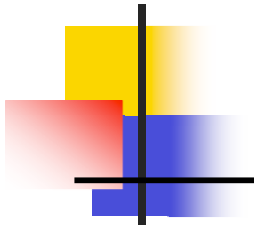
The <throw> Activity

- The <throw> activity is used to signal a fault.
- A fault has a name, and can be defined by the process, by WSDL or be a WS-BPEL standard fault.
- A fault can, optionally, have fault data.
- Throwing a fault will cause fault handling to occur.



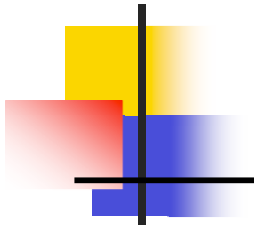
The <rethrow> Activity

- The <rethrow> activity is used within a fault handler, and is only valid within the context of a <catch> or <catchAll> to re-throw the original fault, including the original fault data, if any.
- The fault will propagate until another fault handler catches it, or it passes to the WS-BPEL engine.



Fault Handling

- Fault handlers can be associated with a process, a <scope> or an <invoke>.
- Faults can be initiated by a <throw> or <rethrow>, or by an error in the underlying activities (*e.g.*, an invoke exception).
- The WS-BPEL engine will work its way up the handler stack, looking to match the fault, using the following rules:
  - If the fault has no fault data, match to a <catch> by name. If none is found, use a <catchAll>. If none is present, check the next level handler until we reach the process top.
  - If the fault has fault data, match to a <catch> with matching fault name and variable values. If there is no fault name specified, then match to a <catch> with a matching fault type. If none is present, match to a <catchAll>. Again, if none is present, move to the next level handler, and repeat.

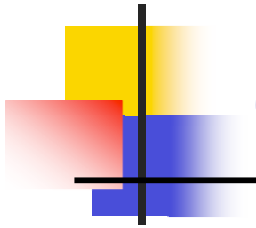


Fault Handling

- The default fault handler (WS-BPEL 12.5.1) is:

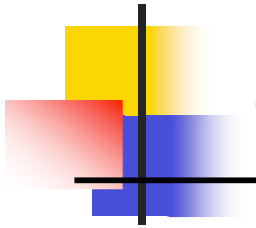
```
<catchAll>
  <sequence>
    <compensate />
    <rethrow />
  </sequence>
</catchAll>
```

- User-defined fault handlers can:
  - Attempt to correct for the problem, and allow the process to continue.
  - Determine that they cannot correct the problem at that level, and:
    - Invoke compensation
    - Reply with a fault message to the initiator
    - Throw a new exception
    - Re-throw the original exception
- If you define a fault handler and do not invoke compensation, no one will do it for you.



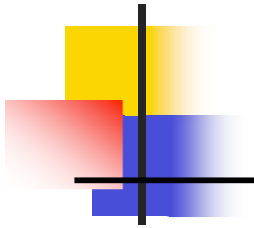
Compensation Activities

- The <compensateScope> and <compensate> activities cause compensation to begin processing.
- <compensateScope> causes a named, successfully completed, inner, scope to start compensation, whereas <compensate> causes all successfully completed inner scopes to start compensation, in the default order.
- These activities can only be used from within fault, compensation or termination (FCT) handlers.



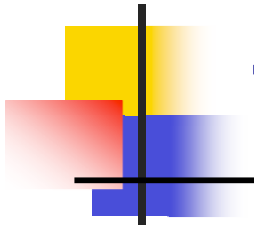
Compensation Handling

- Compensation handlers can be associated with a process, a <scope> or an <invoke>.
- Compensation activities are whatever we want to code for the purpose of undoing successfully completed activities.
  - Kevin deposits $100 in his account.
  - The bank, as per its policy, credits him $100, and waits for confirmation from the payer's bank.
  - The payer's bank eventually notifies the bank, via a message that correlates to the proper business process instance, that the check has bounced.
  - A checkBounced exception can be thrown, and in the fault handler, compensation invoked.
  - The compensation logic can:
    - Remove the $100 from Kevin's account
    - Charge a $20 fee
    - E-mail a notice to Kevin
- Remember that compensation works with successfully completed activities that are nested inside the scope invoking compensation.
  - How does it work if I only declare a fault handler at the process level?
    - There is a default compensation handler at each level, and it invokes compensation.



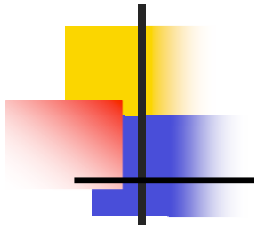
Compensation Order

- Compensation handlers are executed, by default, in the order determined by the following rule:
  - "Consider scopes A and B such that B has a control dependency on A. Assuming both A and B completed successfully and both must be compensated as part of a single default compensation behavior, the compensation handler of B MUST run to completion before the compensation handler of A is started."
  - In other words, the more recent things in the chain are compensated first, followed by older things in the chain.
- This "strict reverse order of completion" can be changed by invoking <compensateScope> with an explicit compensation target.



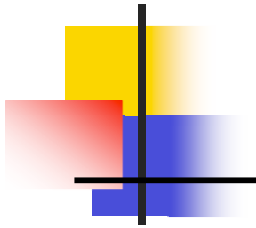
Transactions

- Transactions, in the WS-BPEL specification, are either handled by Compensation or deferred to a vendor. There is no innate support for atomic, database, transactions.
- Some vendors go beyond the WS-BPEL Specification in handling transactions, and identify which WS-BPEL activities have what effect on transactions and transaction boundaries.



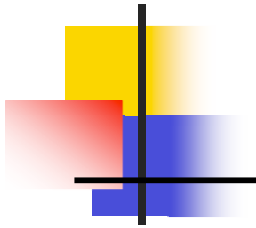
Event Handling

- Event Handlers provide us the means to accept asynchronous events into a Business Process.
  - During the main-line processing of a Business Process, partners are given the able to invoke other interfaces, effecting change within the process.
- The process will expose an interface, whose operations will manifest as events in an event handler, each handled by an <onEvent> element.
  - Each event handler may also have a timeout event, handled by an <onAlarm> element.
  - Refer back to the “WS-BPEL XML Outline” slide for the structure.
- An Event Handler is available as long as the scope to which it is attached is active.



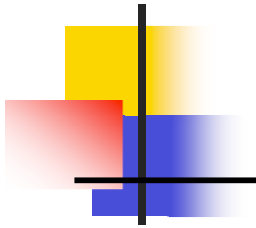
Related Sessions

- "The All-Singing, All-Dancing Business Process" – Doug Tidwell
  - Doug will demonstrate using Apache ODE to execute a WS-BPEL defined business process. Doug will also demonstrate using the Apache Tuscany ESB to allow the business process to access SCA defined services. Finally, Doug will use XForms to define a user interface to the process.



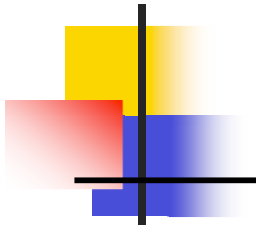
Resources

- Apache Ode
  - http://ode.apache.org
  - Be sure to view the *compliance* page.
- Eclipse BPEL
  - http://www.eclipse.org/bpel/
- WS-BPEL
  - http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel
  - The specification & a primer are available.



Resources

- "Develop and execute WS-BPEL V2.0 business processes using the Eclipse BPEL plug-in"
  - http://www.ibm.com/developerworks/opensource/library/os-eclipse-bpel2.0/
- "BPEL fault handling in WebSphere Integration Developer and WebSphere Process Server"
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0704_desai/0704_desai.html
- "Using compensation in business processes with Business Process Choreographer"
  - http://www-128.ibm.com/developerworks/websphere/library/techarticles/0604_robeller/0604_robeller.html
- Event Handlers in Business Process Choreographer
  - http://www.ibm.com/developerworks/webservices/library/ws-eventhandler/
- An Eight Part Series on BPEL4WS
  - http://www.ibm.com/developerworks/views/webservices/libraryview.jsp?search_by=BPEL4WS:
- "Transforming business models to BPEL with WebSphere Business Modeler 6.02"
  - http://www.ibm.com/developerworks/websphere/library/techarticles/0706_fasbinder/0706_fasbinder.html



Q&A

- Any Questions? ☺
- Please remember to turn in your session evaluation.