Service Composition and Modeling Business Processes with BPEL

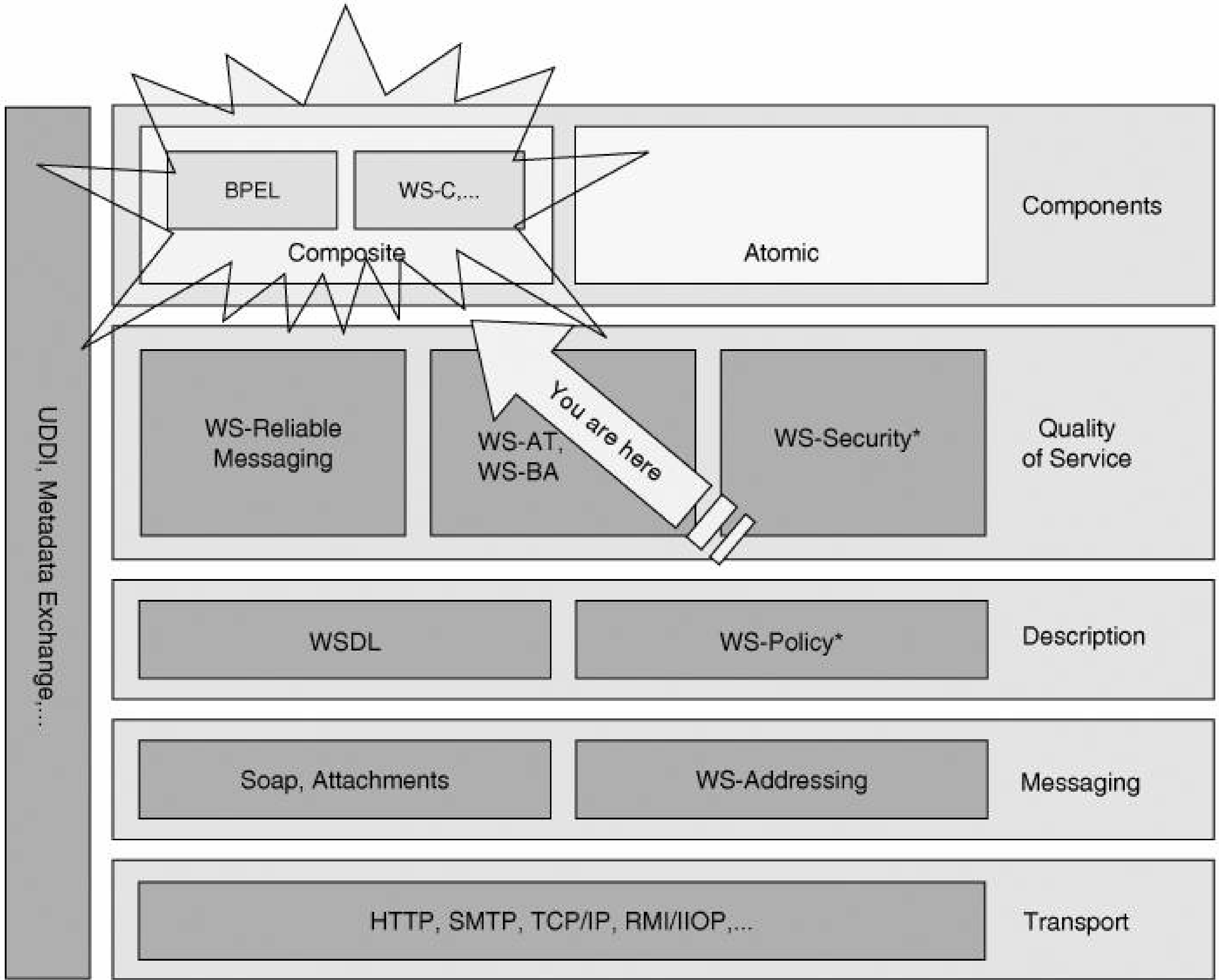
by

Sanjiva Weerawarana, Francisco Curbera, Frank Leymann, Tony Storey, Donald F. Ferguson

Reference:

`Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More', Prentice Hall PTR, 2005

Stack of Web services specifications



Background

- Web Services Business Process Execution Language specification (WS-BPEL, better known as BPEL) is being standardized at OASIS.
- BPEL enables service aggregations to be defined in both an abstract and an executable manner.
- A handful of other specifications that address different aspects or styles of composition have been proposed; however, BPEL is at this point the only one with significant following in the industry.

Introduction

- WS-* standards and specifications enable the creation of numerous services in the network and the mechanisms to discover and interact with them.
- To move beyond the "publish, discover, interact" model, it's required to have the ability to define logic over a set of service interactions.
- This not only enables you to compose a set of services, but it also enables the definition of the interaction protocol of a single service by specifying an ordering of its operations. Web services Business Process Execution Language comes into the picture.
- First known as BPEL4WS but soon to be renamed as WS-BPEL, this specification is more commonly known as BPEL.
- It is an extensible workflow-based language that aggregates services by choreographing service interactions.
- The aggregation is recursive, such that the process exposes WSDL interfaces to those that interact with it, and the corresponding services may be used in other choreographies.

Introduction

- Designed to work in a highly dynamic environment in which services might change frequently, a BPEL process is intentionally decoupled from particular instances of the services it choreographs.
- BPEL layers neatly at the top of the WS specifications stack, directly using WSDL interfaces to define the functionality it offers to and requires from the services that are being composed, and for defining its interactions with them.
- XPath can be used to access and manipulate data belonging to the choreograph
- You can copy and exchange the endpoints of composed services in the form of WS-Addressing endpoint references
- With the modularity of the framework, you can add additional aspects to BPEL processes, e.g you can use WS-Policy and WS-PolicyAttachments to attach quality-of-service policies to different levels of the process and the related WSDLs, and you can use different bindings to carry out the interactions.
- .

Example

- A client wants to purchase service packs for a set of computers. Service packs are bundled support services, such as on-site repair, hardware warranties, and so on. Someone needs to validate the order before the client can proceed with the purchase.
- The validation consists of checking several things, such as
  - whether the requested service packs are available in the client's context (geographical location, order date, and so on),
  - whether they are compatible with the given machine(s), and
  - whether they are compatible with the other service packs that are being requested.
- This example is derived from an IBM business application created by the Exploratorium Group at IBM's TJ Watson Research Center.

Example

- The BPEL process can accept the request for purchase order validation as a call to the Web service that it exposes.
- You then break up this request into smaller pieces, each sent to a separate specialized Web service that can validate a particular aspect.
- After that, you create the reply to the validation request based on the responses from the services that are called.
- The reply contains the results of validation for all the service packs you requested.
- You then return this reply to the caller. After that, the BPEL process can accept a purchase request to buy these service packs.
- The process then calls services to bill the client and update the client's profile to reflect the purchase.

Motivation for BPEL

- SOA demands a new approach that can handle, in a first class manner,
  - its mainstays of loosely coupled entities;
  - on-demand interactions;
  - frequent change of such parameters as location,
  - availability, and quality of service; and
  - lack of control over the platform and implementation of services that are being composed.

Motivation for BPEL

Flexible integration

- The composition model must be sufficiently rich to express business scenarios that partners might exchange. More importantly, it should rapidly adapt to changes in the services that it is interacting with.

Recursive composition

- Offering a process as a standard Web service enables third-party composition of existing services, the ability to provide different views on a composition to different parties, interworkflow interaction, and increased scalability and reuse.

Motivation for BPEL

Separation and composeability of concerns

- In keeping with the composeability of the Web services framework, the business/service-composition logic should be decoupled from the supporting mechanisms such as quality of service, messaging frameworks, and coordination protocols.
- Such information should be capable of being layered on or attached to different parts of the process definition if necessary.

Stateful conversations and lifecycle management

- A workflow should have a clearly defined lifecycle model, in which it can carry multiple stateful long-running conversations with the services that it is interacting with.

Motivation for BPEL

Recoverability

- Business processes (especially long-running ones) need to provide built-in fault handling and compensation mechanisms to deal with expected errors that might arise during execution.
- For example, if a credit card number is invalid, the process would ask for a new one instead of aborting entirely.

A Brief History

- BPEL emerged as a combination of two prior, competing XML languages for composition:
  - the IBM Web Service Flow Language (WSFL) and
  - Microsoft XLANG [XLANG].
- Although similar in their goals, each language had a different composition approach.

WSFL

- In WSFL, the flow model is a business process described as a directed graph of activities (the nodes of the graph) and control connectors (the edges of the graph).
- Nodes and edges of the graph are annotated with attributes, such as transition conditions, that determine the execution of the model.
- WSFL also provides the possibility to define a global model, in which you can specify the overall partner interactions.
- A global model is a simple recursive composition metamodel that allows you to describe the interactions between existing Web services and to define new Web services as a composition of existing ones.

XLANG

- XLANG provides a notation "for the specification of message exchange behavior among participating Web services" so that you can achieve "automation of business processes based on Web services."
- To achieve this objective, the major constructs of XLANG include sequential and parallel control flow definitions, long-running transactions with compensation, custom correlation of messages, exception handling, and dynamic service referral.
- An XLANG service description extends a WSDL service description with the behavioral aspects of the service.
- It allows the user to specify a set of ports in a service section of a WSDL document and extend it with a description of, for example, the sequence in which the operations that the ports provide are to be used.

A Brief History

- In BPEL, you create the processes by using a combination of the graph-oriented style of WSFL and the algebraic style of XLANG.
- BPEL also allows you to recursively compose Web services into new aggregated Web services.
- The first version of the BPEL4WS specification was released in 2002. Version 1.1, released in 2003, was submitted to OASIS for standardization.
- The group is set to produce the next version, renamed WS-BPEL version 2.0, as its first output. The last stable version of the specification is BPEL4WS v1.1. The changes proposed so far for version 2.0 have little impact on the architectural concepts described here.

Architectural Concepts

Architectural Concepts

- Overview of BPEL's service composition model and the elements that make up a BPEL process.
- BPEL's recursive type-based composition model and process instance lifecycle.
- Advanced capabilities, including
  - event handling,
  - dealing with exceptional behavior, and
  - using Web services policies to attach quality of service to a BPEL process.

Overview of the Process Composition Model

- The major building blocks of BPEL business processes are
  - nested scopes that contain relationships to external partners,
  - declarations for process data,
  - handlers for various purposes and,
  - most importantly, the activities to be executed.
- The outermost scope is the process definition.
- Data in BPEL is written to and read from lexically scoped, typed variables. The values of these variables are either messages exchanged between the process and its partners, or intermediate data that is private to the process.
- In keeping with the Web services framework, BPEL variables are typed using WSDL message types, XML Schema simple types, or XML schema elements.
- XPath is the default language for manipulating and querying

Overview of the Process Composition Model

- Data can be read or written by activities that exchange messages between the process and its partners or by specific activities that manipulate process-data (so-called assign activities).
- Conditional expressions also read data.
- BPEL contains several expression types that are used for different purposes, such as
  - defining conditions (Boolean expressions),
  - expressing time intervals and dates (duration and date expressions), and
  - accessing values of variables in the assign activity (general expressions).

Overview of the Process Composition Model

- The activities in a BPEL process are either structured or basic.
- Structured activities contain other activities and define the business logic between them.
- Basic activities are, for example, the inbound or outbound Web service interactions or the specific activities for data manipulation.
- Activities that deal with Web services are receive, reply, pick, invoke, and event handlers.
- These activities allow a process to exchange messages with the services that it composes. They are discussed in detail in section, "Recursive, Type-Based Composition."

Overview of the Process Composition Model

- The assign activity atomically copies data from one location to another, using a list of copy statements that copy one value each.
- The common usages of an assign activity include
  - constructing values by copying them from XPath expressions into a part of a variable,
  - copying data from the part of one variable into a part of another, and
  - copying endpoint references so that they can be exchanged between the process and its partners.
- This last case is the enabler of dynamic partner assignment in BPEL.

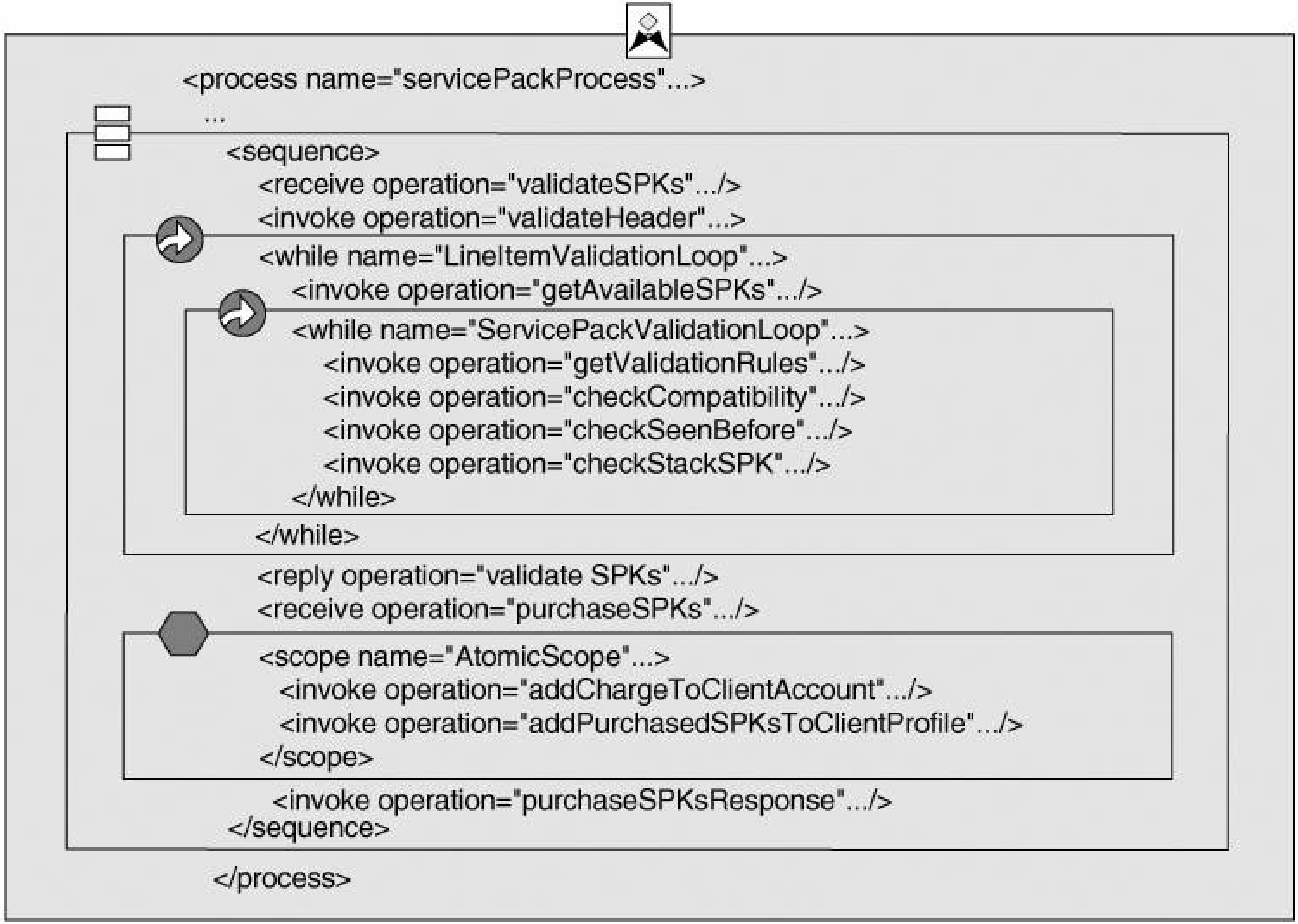
Overview of the Process Composition Model

- Structured activities combine multiple activities to provide higher-level business logic.
- These include sequence, switch, while, and flow activities.
- The sequence activity is a simple aggregation of activities executed in the order in which they are specified.
- The switch activity provides a multibranch decision construct
- The while activity provides a loop construct.
- Both switch and while have semantics from corresponding programming language constructs, such as Java.
- You can execute activities in parallel by nesting them in a flow activity
- If you have to constrain the degree of parallelism, you can use conditional control links to specify a partial (acyclic) order on the set o activities nested in a flow.

Overview of the Process Composition Model

- Next figure illustrates how to create a business process for the service pack validation (SPK) example using several of these constructs.
- You can see the receive activity and its reply further downstream, in addition to several invocations that invoke the specialized validators.
- Surrounding these and imposing control, you see a sequence activity that imposes order on the validation and the purchasing, several nested while loops, and a scope that surrounds the purchasing section at the end.

Structured activities

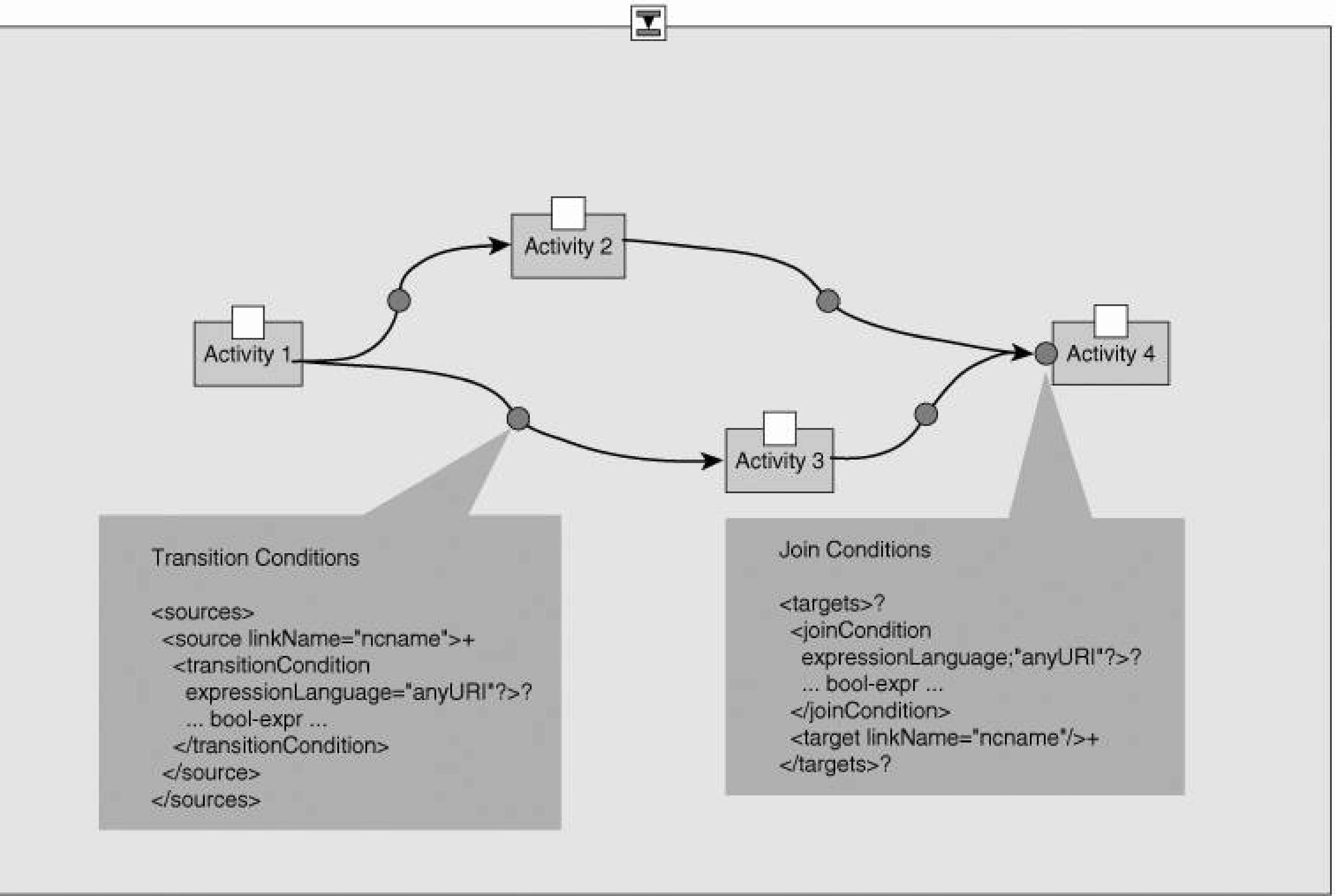


```
<process name="servicePackProcess"...>
 ...
  <sequence>
    <receive operation="validateSPKs".../>
    <invoke operation="validateHeader"...>
    <while name="LineItemValidationLoop"...>
      <invoke operation="getAvailableSPKs".../>
      <while name="ServicePackValidationLoop"...>
        <invoke operation="getValidationRules".../>
        <invoke operation="checkCompatibility".../>
        <invoke operation="checkSeenBefore".../>
        <invoke operation="checkStackSPK".../>
      </while>
    </while>
    <reply operation="validate SPKs".../>
    <receive operation="purchaseSPKs".../>
    <scope name="AtomicScope"...>
      <invoke operation="addChargeToClientAccount".../>
      <invoke operation="addPurchasedSPKsToClientProfile".../>
    </scope>
     <invoke operation="purchaseSPKsResponse".../>
  </sequence>
 </process>
```

Overview of the Process Composition Model

- A transition condition is associated with each control link and evaluated at completion of the link's source activity.
- This condition has a default value of true, but you can use any Boolean expression that relates to the values of data fields and other states of the process.
- On the other hand, a join condition is associated with an activity that is a target of links. It is a Boolean expression in terms of the link values.
- It is evaluated after you know all incoming link values and it must evaluate to TRue to run the activity.
- A joint condition's default value is an or. These concepts are illustrated in the next figure, in which four activities are joined with control links.

Links, transition conditions, and join conditions



Transition Conditions

```
<sources>
 <source linkName="ncname">+
  <transitionCondition
   expressionLanguage="anyURI"?>?
   ... bool-expr ...
  </transitionCondition>
 </source>
</sources>
```

Join Conditions

```
<targets>?
 <joinCondition
  expressionLanguage;"anyURI"?>?
  ... bool-expr ...
 </joinCondition>
 <target linkName="ncname"/>+
</targets>?
```

Overview of the Process Composition Model

- In addition to Web service interactions and nested structured activities, you can use a couple more basic activities to specify different types of behavior.
- These activities are called empty, wait, terminate, tHRow, and compensate.
- The empty activity is BPEL's rendering of a no-op. You can use it as a placeholder for other activities that you add later.
- The wait activity provides a means to interrupt the execution for a specified time interval or until a specified time has been reached, and the terminate activity stops the execution of the process immediately.
- Finally, the throw and compensate activities are used in conjunction with handlers for error detection and recovery, which will be discussed later.

Overview of the Process Composition Model

- A BPEL process might contain handlers to recognize unexpected problems and deal with them.
- These handlers can include operations that reverse effects from other activities that have been completed successfully before an error situation occurred.
- The constructs that BPEL provides present a rich toolbox with which to create both simple and sophisticated business processes.
- In the sections that follow, you see how to create and use BPEL processes.

Abstract and Executable Processes

- BPEL supports two fundamental usage patterns.
  - One pattern is for describing business protocols called abstract processes, and
  - the other is for describing executable processes.
- Business protocols or message exchange protocols describe the externally visible interactions between business partners without necessarily exposing the internal business logic of the individual partners.
- In contrast, executable processes contain the partner's business logic behind the external protocol.

Abstract and Executable Processes

- An enterprise might present abstract processes of the existing business processes that it offers to its partners so they can know how to interact with it.
- The abstraction enables the enterprise to keep its private mechanisms internal while still providing the partners with the information they need.
- You can think of an abstract process as a projection of an executable process.
- On the other hand, for a certain domain just an abstract process can be made publicly available.
- Service providers that want to supply the specified functionality then derive executable processes that comply with it.

Abstract and Executable Processes

- BPEL provides the same language constructs to define both abstract and executable processes.
- Both processes have a few additional constructs that are an extension of the base language and can be used only in that variant.
- For example, abstract processes might have "opaque" variable assignment, to signify that a value is set but that how or what that value is remains up to the internal implementation.

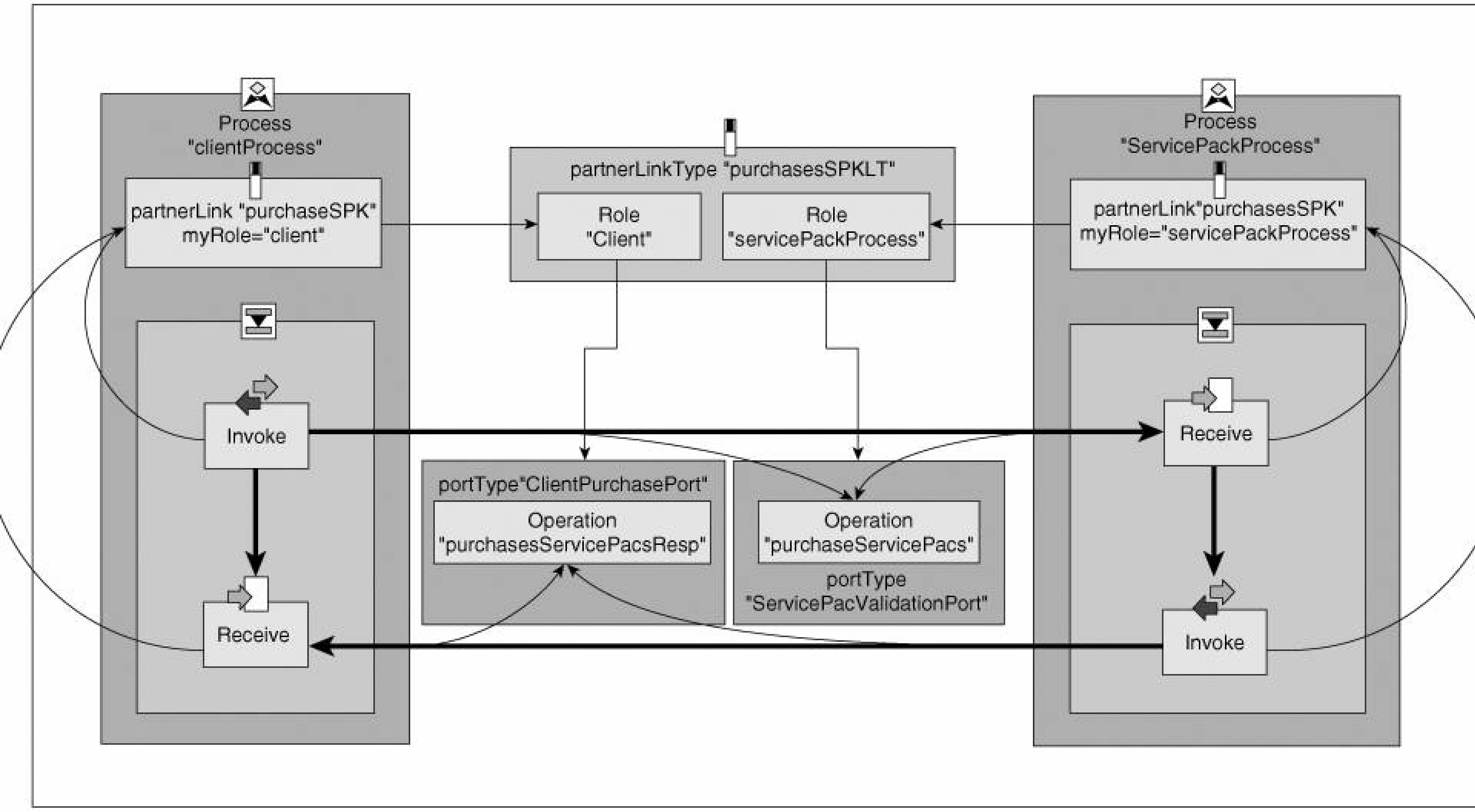
Recursive, Type-Based Composition

- A BPEL process refers to the parties that it is interacting with as partners.
- It interacts with each partner along a set of partnerLinks. Partner links are instances of typed connectors that specify the portTypes that the process offers to and requires from the partner at the other end of the link.
- You can think of a partnerLink as a channel along which a peer-to-peer conversation with a partner takes place.
- The composition model is recursive: The portTypes that the process offers form (one or more) Web services with a WSDL interface.
- The bindings and ports for those portTypes might be specified at deployment time, depending on where the service is deployed.

Recursive, Type-Based Composition

- To illustrate, the client and the service pack validator in the running example can both be Web services with a callback on the purchase operation.
- The validator offers the "purchaseServicePacs" operation to the client, and the client offers the "purchaseServicePacsResp" operation to the validator.
- Therefore, the validator BPEL contains a partnerLink connecting the two respective portTypes, as illustrated in the next figure.

Partner links


Process
"clientProcess"
partnerLink "purchaseSPK"
myRole="client"
Invoke
Receive
partnerLinkType "purchasesSPKLT"
Role
"Client"
Role
"servicePackProcess"
portType"ClientPurchasePort"
Operation
"purchasesServicePacsResp"
Operation
"purchaseServicePacs"
portType
"ServicePacValidationPort"
Process
"ServicePackProcess"
partnerLink"purchasesSPK"
myRole="servicePackProcess"
Receive
Invoke

Composing Services

- BPEL's composition model is type based, meaning that services are composed at the portType and not at the port/instance level.
- A BPEL process provides control semantics around receive, reply, and invoke activities.
- Each of these activities specifies the partnerLink-portType operation and the variables that it relates to.

Composing Services

- An input-only operation that a BPEL process provides corresponds to a receive activity in the process that explicitly marks the place in the business process logic where the inbound request message is accepted.
- For request-response operations, a corresponding reply activity is the place where the response message is handed back to the Web services client.
- If a process provides multiple operations, then multiple receive activities indicate that you must take all entry points into the process.
- Alternatively, a pick activity can refer to multiple operations to indicate that, at most, one entry point is taken.
- Later, you'll learn more about concurrent entry points that event handler activities define.
- Outbound calls to Web services that entities offer outside of the process are expressed as invoke activities.

Composing Services

- To summarize, the invoke activities always refer to the operations that the partners offer.
- Therefore, in the previous figure, you see that the client has an invoke that refers to the validator's "purchaseServicePacs" operation.
- On the other hand, receive/pick/onMessage and their (optional) corresponding reply activities refer to the operations that the process offers to its partners.
- These are the operations included in the WSDL portType(s) of the process. Again, you can see this in the figure, where the receive that consumes the purchase request in the validator's BPEL refers to its own "purchaseServicePacs" operation.

Binding to Concrete Endpoints of Partner Services

- BPEL decouples the business logic from the available service endpoints, allowing processes to be more adaptive and more portable.
- When a process must execute, however, you must be able to bind each partnerLink along which a partner is invoked to a concrete endpoint.
- Four binding schemes and their combinations are possible:

Binding to Concrete Endpoints of Partner Services

Static design time binding

- A process might be bound to known endpoints as part of the process logic. This is the least flexible form of binding.
- This binding proves useful when the endpoints are meant to be parts of the business logic.
- (That is, this process is made only for bank such-and-such.) Every deployment and every instance of this process will use the specified endpoint(s).
- For this form of deployment, an assign activity explicitly puts an endpoint into a variable. A subsequent <assign> activity then copies the value of that variable into the process's reference to a specific partnerLink.

Binding to Concrete Endpoints of Partner Services

Static deployment time binding

- In this instantiation, the process model does not refer to explicit endpoints, but a set of endpoints is defined at deployment time to the workflow management system into which the process is deployed.
- This form is useful if all instances of a particular deployment of a process must use the same endpoints.

Binding to Concrete Endpoints of Partner Services

Dynamic binding using lookups

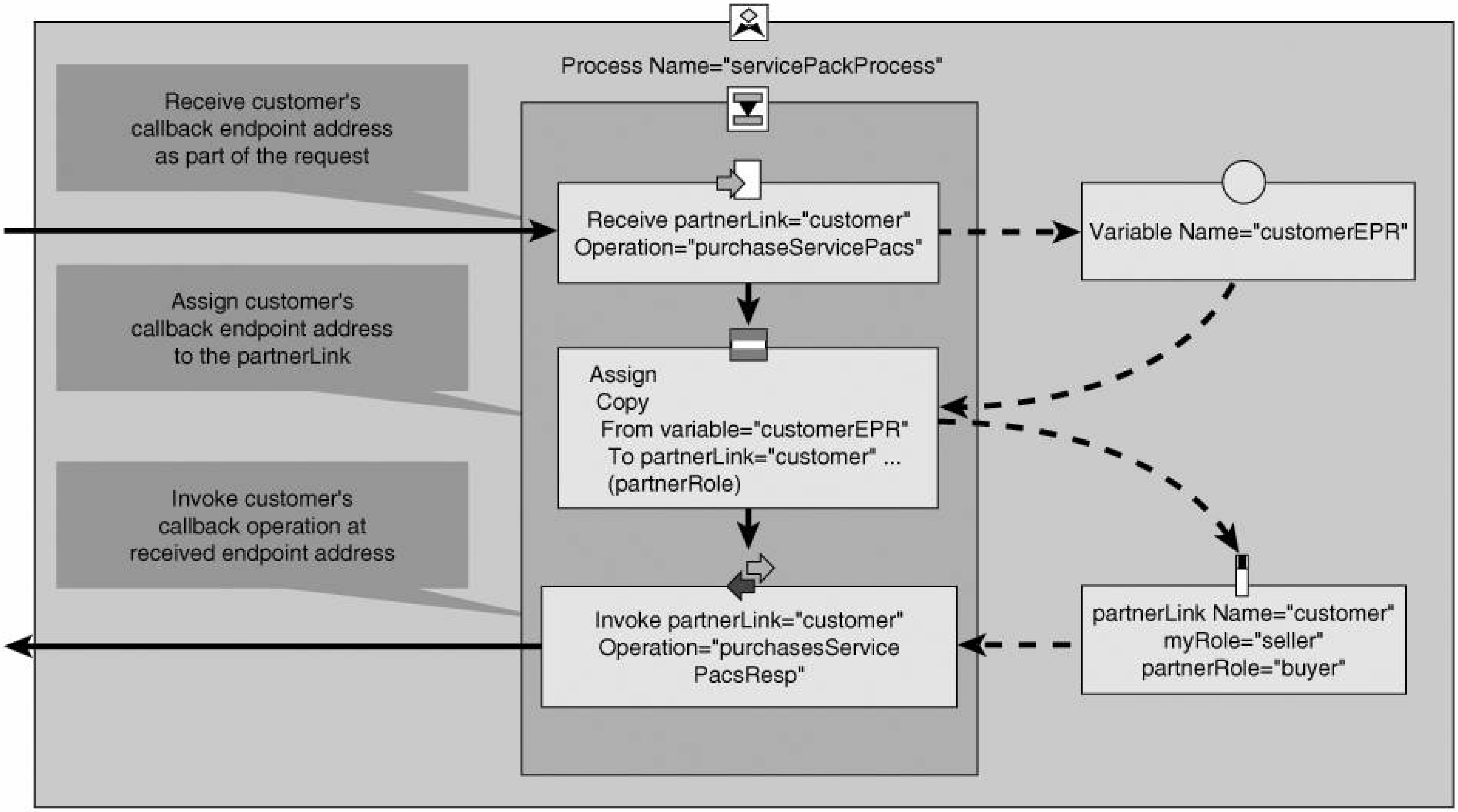
- In this form of binding, criteria are defined on a partnerLink that a viable endpoint must have.
- These criteria are evaluated either at runtime or deployment time to search for providers whose deployed services match.
- Such criteria might include quality-of-service policies (cost, security, speed), transactional capabilities, or functional requirements that are not in the WSDL portType.
- The location of the endpoints is the responsibility of the environment in which the process is deployed or executed.

Binding to Concrete Endpoints of Partner Services

Dynamic binding using passed-in endpoints

- The last form of dynamic binding includes the process copying in an endpoint from a variable that was previously assigned not by the process, as in (1), but by either a response to an invocation (result of an invoke activity) or a request from a partner (input to a receive activity).
- The latter is illustrated in the next figure. The process uses the assign activity as in (1) to copy that value into its reference to the partnerLink in question.

Dynamic binding of partner links


Process Name="servicePackProcess"
Receive customer's callback endpoint address as part of the request
Assign customer's callback endpoint address to the partnerLink
Invoke customer's callback operation at received endpoint address
Receive partnerLink="customer" Operation="purchaseServicePacs"
Assign Copy From variable="customerEPR" To partnerLink="customer" ... (partnerRole)
Invoke partnerLink="customer" Operation="purchasesService PacsResp"
Variable Name="customerEPR"
partnerLink Name="customer" myRole="seller" partnerRole="buyer"

A Conversational Approach

- receive or a receive/reply activity pair maps to one WSDL operation exposed to the partners, with the conceptual implementation of that operation being the part of the process that is executed as a result of the receive.
- In the case of a request-response operation, that is logically the block between the receive and its corresponding reply.
- For example, in a figure of Structured activities, you can see the implementation of the "validateSPKs" operation that the process offers between the first receive and the first reply, both of which refer to that operation.

A Conversational Approach

- Having different activities for incoming and outgoing messages and breaking up a request-response operation by using receive/reply enables BPEL to support a conversational pattern of composition and interaction with the partners.
- In BPEL, the activities between the receive and the reply might include other receives, even from the same partner along the same partnerLink, and other invokes. It's like natively being able to call a method within a method.
- Depending on the process design, the process, thus, might be a controlling entity, a player on an equal footing with its partners, or a mix of the two.
- The result is the capability to model a wide range of relationships between a process and its partners, from peer-to-peer to master/slave.

A Conversational Approach

- A key architectural enabler is the bidirectionality of partnerLinks. A conversation occurs along each partnerLink for each process instance. That conversation is potentially two sided.
- The process instance invokes the partner, and the partner invokes the process instance.
- The lifetime of a conversation is the lifetime of the entire process instance, and you can see the different conversations of an instance through the specification of which partnerLink each interaction activity refers to.
- In BPEL, only one endpoint is active at any one time at each end of the partnerLink (one being the process instance and the other being the instance of the partner).
- Although that can change at different points in time because of such things as dynamic assignment, it changes for the entire instance and not only for the lifetime of one operation invocation.

Process Views

- Offering possibly different portTypes along each partnerLink, the process can expose to each partner only the functionality that is relevant to it.
- One BPEL process might be exposed as one or more Web services, each service being a view on the entire process. This provides the separation of concerns of a process's provided functions.
- For example, consider a process whose functions include sending requested payments to utility companies and providing auditors with a summary of payment activity.
- Using BPEL, such a process could publish only the payment portType to the utility companies and only the auditing portType to the auditors. This way, the utility companies do not need to know or care about the auditing capability, and the auditor cannot call the payment functions that are not meant for him.

Process Views

- Likewise, the partner might not need to know about the process's interactions with other entities.
- In addition to providing for each partner an abstract WSDL view of the parts of the process that the partner is involved with, you can go a step further with BPEL and accompany that WSDL view with the definition of an abstract process showing the parts of the process that the partner is concerned with.
- Such an abstract process might be simply a projection of the interaction activities that are concerned with that partner. Of course, this is not always necessary.
- Other options include exposing the whole executable process, exposing an abstract variant that shows interactions with multiple partners and is visible to all, or completely hiding the process based on the circumstances and the goals that the process was created to fulfill.

Process Instance Lifecycle

- Business processes that are defined in BPEL represent stateful Web services, and as such, they might have long-running conversations with other Web services.
- Whenever you start a new BPEL process, a new instance of that process is created, which might communicate with other business partners.
- Web service definitions in WSDL are always stateless, so the messages that are exchanged in long-running conversations must be correlated to the correct business process instance. BPEL offers the concepts that allow correlating application data to process instances.
- After you define and deploy a process, you can create multiple instances of it. All the instances can run concurrently and completely independently of each other. This section considers how such instances are created, destroyed, and identified at runtime.

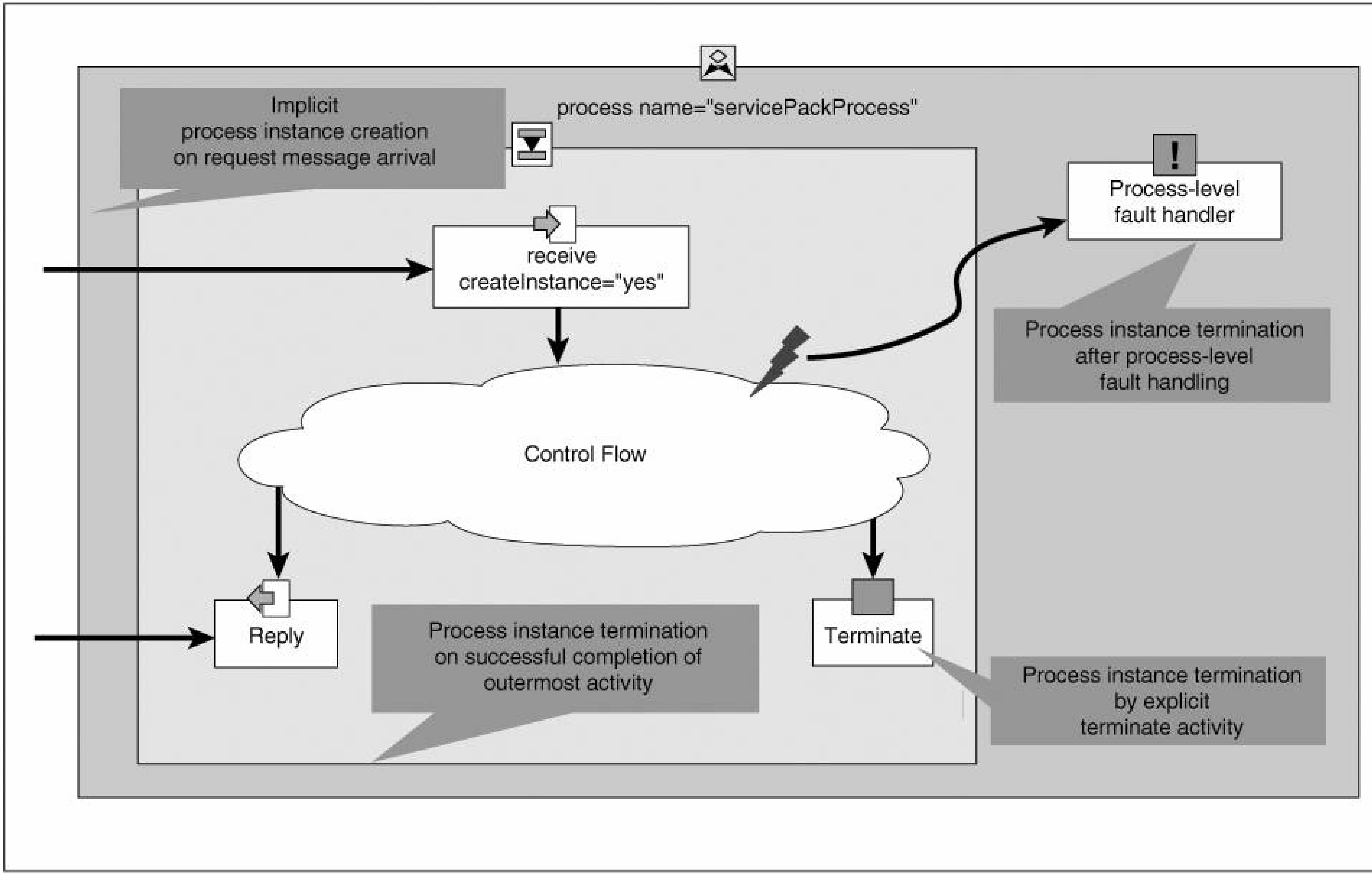
Process Instance Lifecycle

- The creation and destruction of BPEL process instances is by design implicit. In some workflow systems, each process has an explicit start and end activity.
- In BPEL, however, this restriction is not needed and multiple activities in a process model can kick off a process instance.
- You must tag these activities with the capability of instance creation by setting the createInstance attribute to yes.
- These activities must be able to receive messages (receive, pick) and come near the beginning of the process so they can be navigated to as soon as the instance is created. An instance can be created if a partner invokes an operation of the process that corresponds to one of these activities.
- The condition is that no active instance that already existed could consume the incoming message.
- After an instance is created, the "createInstance" attribute on any other activities than the one that created it loses its creation potency.

Process Instance Lifecycle

- An instance terminates when its last activity completes, a terminate activity executes, or the instance experiences a fault from which it cannot recover.
- The next figure summarizes instance creation and termination in a BPEL process.

Process instance lifecycle



process name="servicePackProcess"
Implicit process instance creation on request message arrival
receive createInstance="yes"
Process-level fault handler
Process instance termination after process-level fault handling
Control Flow
Reply
Process instance termination on successful completion of outermost activity
Terminate
Process instance termination by explicit terminate activity

Correlation

- After you create an instance encompassing multiple receives, you have to figure out how to route subsequent messages to the correct created instance.
- In traditional middleware systems, you do this using opaque tokens that the middleware generates and maintains.
- BPEL, however, defines an (optional) correlation mechanism to enable you to do just that using parts of the application data.
- Two items make up these mechanisms: *properties* and *correlation* sets.

Correlation

- Typed properties that are named and mapped (aliased) to parts of several of the WSDL messages that the process uses are defined in WSDL.
- For example, to use a person's name as correlation identifier, you would define a property called "name" of type "xsd:string" that is aliased to the "name" field in the clientInformation part of a "validateServicePacks" WSDL message and the "lastName" part of a "purchaseOrder" WSDL message.
- These messages could be inputs to two different operations.

Correlation

- BPEL correlation sets are groups of these properties that identify which process instance an incoming message should be routed to.
- Zero or more correlation sets might subsequently be attached to the interaction activities (receive/reply/invoke) with a flag on whether the activity will initiate the set's data.
- With invoke, you must also specify whether you are initiating it based on the outgoing request message or the incoming response message. Initiating a correlation set corresponds to setting the values of the properties for a particular instance.
- You can then use values in incoming messages to look up the instance that the message is directed to by matching against the values of correlation sets of active instances.

Correlation

- For example, consider that the receive activity on the "validateSPKs" operation shown in the figure showing Structured activities creates an instance of a process and initiates a correlation set containing the "name" property. Assume that a message "A" comes in for the process and matches that receive activity.
- An instance is created, and the alias is used to find the "name" in the incoming message and copy it into the value of that property.
- Subsequent referrals to that property will have that value.
- More importantly, the correlation set with that value for the name property will be registered as a key to that particular instance.

Correlation

- Now consider that the other receive on the "purchaseServicePacs" operation further downstream refers to the same correlation set.
- A message comes in for that receive activity with the same name.
- The workflow engine looks up the name field and finds that the message must go to the same process instance as the one that message "A" had created earlier.

Correlation

- Correlation is a powerful concept. It enables you to maintain conversations between process instances and their partners using clearly defined and aliased portions of the application data.
- You can use multiple correlation sets to point to the same process instance, using different groups of properties.
- This enables a process instance to use different sets of identifiers for each partner or for different parts of a conversation.
- If you use opaque middleware tokens for routing, correlation still enables validation of the business logic in ensuring that certain pieces of data have the same value in each instantiation of a process.
- One example is validating that the same account number is used in all of a person's interactions with an instance of an ordering process.

Event Handling

- A BPEL process provides one or more Web service interfaces. In addition to the interfaces that are implemented by receive or pick activities in the regular flow, BPEL provides a means to deal with asynchronous events, which are handled concurrently to the execution of the process.
- These constructs are associated with the process as a whole or with a scope, and they are called event handlers.
- You can use event handlers in BPEL for message events and timer-driven alarm events.
- For the outside world, message events are regular one-way or request-response Web service operations that the business process implements.
- Alarm events occur whenever a specified time is reached or a specified time interval expires. Both types of event handlers can contain any BPEL activity.

Event Handling

- As an example, you can use event handlers to monitor the progress of a business process.
- For this purpose, the event handler is modeled as a request-response operation, in which the request specifies a query expression, and the response contains process instance status data that corresponds to the query.

Dealing with Exceptional Behavior

- Because BPEL processes are composed out of Web services, they have to deal with faults that are a result of Web services invocations.
- Likewise, because BPEL processes implement regular Web services, they might return exceptional situations as faults to the invokers of these Web services.
- Finally, abnormal behavior might be recognized within the business logic, either raised explicitly or recognized by the BPEL runtime infrastructure.
- Business processes can be long running.
- As a consequence, you cannot always execute them as a single atomic operation.
- Results of intermediate process steps are made persistent and may be visible to others. When a fault occurs in such a long-running business process, you have to undo work that is partially or successfully completed.

Dealing with Exceptional Behavior

- Within a BPEL process, the construct that is associated with handling fault situations and reversing intermediate results is the scope, which you can consider as an encapsulation of a recoverable and reversible unit of work.
- You deal with faults by using fault handlers.
- Undoing work is the responsibility of compensation handlers.
- Both handlers are either implicit or are explicitly provided by the process modeler.

Dealing with Exceptional Behavior

- In addition to these concepts that are local to a business process, abnormal behavior sometimes occurs between multiple business processes.
- Again, you either can achieve an all-or-nothing behavior, or you need explicit means to reverse results of work that have already been persisted.
- In the environment of distributed, multiplatform, multivendor business transactions, all-or-nothing behavior is described by the WS-AtomicTransaction [WS-Atomic Transactions] specification, and exception handling in long-running business transactions is described by the WS-BusinessActivity [WS-Business Activity] specification.

Fault Handling

- The primary purpose of a fault handler is to catch faults and reverse partial and unsuccessful work.
- In a BPEL process, faults can originate from different sources.
- A fault can be the result of a Web service invocation, or it can be thrown explicitly from within the business logic.
- Furthermore, the BPEL specification lists numerous situations in which the BPEL infrastructure must detect and signal faults, such as when a BPEL variable is used before a value has been assigned to it.

Fault Handling

- For faults that are returned from a Web service invocation, the Web service interface defines a fault message that allows data to be passed along with the fault.
- Within the process, you can also raise faults without associated fault data.
- BPEL fault handlers can deal with both and can examine the fault data.
- When a fault handler catches a fault, the regular processing within a scope stops, and exception processing begins.
- Therefore, in a BPEL process, fault handling is considered a mode switch from doing regular work to repairing an exceptional situation.

Compensation

- The BPEL specification mandates a particular transaction model for executing business processes: In many business processes you cannot complete all work within a single atomic transaction, because processes are long-running business transactions in which many committed ACID transactions create persistent effects before a process is completed.
- Application-specific compensation steps undo these effects when necessary.

Compensation

- In a BPEL scope, the compensation handler reverses completed activities.
- You can invoke the compensation handler after successful completion of its associated scope, using the compensate activity.
- You can perform the invocation of a compensation handler either from a fault handler or from the compensation handler of the enclosing scope.
- In its first version, the BPEL compensation handler cannot change data within the business process. The activity that is enclosed in a compensation handler typically consists of one or more Web service invocations that undo effects of other previously called Web services.

Compensation and Business Agreement Protocols

- The Web Services Business Activity specification (WS-BusinessActivity) describes a model in which the compensation actions are coordinated across several business applications.
- When you apply this model to BPEL processes, long-running work is distributed over scopes within multiple BPEL processes and executed as nested scopes.
- In addition, WS-BA conceptually describes the states, state transitions, and stimuli underlying BPEL scope-based local transaction management.

Extensibility and the Role of Web Services Policies

- Sometimes you need to define additional capabilities that BPEL's constructs do not provide. One approach uses BPEL's extensibility.
- It involves defining private extensions and providing support for them in specialized engines.
- The problem is that these processes can no longer be shared with others. Portability is compromised. When portability is not a goal, you can certainly take this approach.
- You can create domain-specific BPEL extensions for certain specialized areas. The relevant industry sectors can then standardize these extensions.

Extensibility and the Role of Web Services Policies

- More important, however, is the ability to recognize when these capabilities are orthogonal to the business logic. Examples include quality-of-service requirements and capabilities.
- People often rush to add functionality to the base language when the requirements they have are better met by the other pieces of the Web services stack.
- This is when you apply the modularity of the Web services stack. In these situations, you can attach Web services policies to different parts of the BPEL process, the corresponding WSDLs, or both.
- You can attach policies to the partner construct to denote requirements that must apply along all the interactions with the specified partner. You can apply them to the partnerLink or even to each interaction activity.
- On the other hand, you can attach policies to the WSDL portTypes or ports that correspond to the process. These specify the capabilities that the process offers to its partners.

Extensibility and the Role of Web Services Policies

- The attachment of policies to the business process enables the separation of concerns that is desirable to keep the business logic intact and first order.
- Therefore, when you consider adding proprietary constructs to BPEL, you should first think of whether the functionality that the constructs are trying to add is better achieved through the use of Web services policies that are orthogonal to, yet can be integrated with, the business logic.

BPEL Processing Model

BPEL Processing Model

- The previous sections explained the different aspects that make up a process.
- This section delves into the runtime aspects of BPEL processes.

Deployment

- A BPEL process exposes zero or one portType along each partnerLink ("myRole").
- After the portTypes are deployed, they must be made available for invocations from the partners.
- With WSDL specifying that one portType corresponds to one port with the address of the available service, engines that deploy BPEL processes are expected to create such ports.
- The BPEL specification does not specify a deployment approach or syntax for a deployment descriptor.

Deployment

- The engine or workflow system to which a process is deployed is responsible for creating instances based on the instance creation activities and maintaining conversations between each of these instances and their partners, usually using correlation sets.
- The approach or interfaces that bind to partners at deployment time or specify locators for their binding to happen at runtime, such as in section “Binding to Concrete Endpoints of Partner Services”, are omitted from the BPEL specification.
- The workflow systems must present and support such approaches.

Interacting with the Process

- After you deploy a process, partners interact with it by invoking the operations it offers them.
- The first of such invocations, targeted at an instance that creates receive or pick activity, creates an instance of the process and registers the values of correlation properties that are initialized at that point in time.
- The interaction activities of a process specify which variables to read outgoing messages from and which variables to write incoming messages to.

Interacting with the Process

- The messages in invocations of a process's operations are eventually consumed by a running receive activity, pick activity, or event handler.
- After an activity starts running, it waits for a message to arrive that it can consume, saves it in a specified variable, and completes.
- To disambiguate which activity might consume an incoming message, only one receive/pick/onMessage with the same partnerLink, portType, operation, and correlation set can be running at the same time.
- If the operation is request-response, a subsequent reply activity must exist further downstream in the process.
- After the reply activity executes, a response is sent to the invoker, and the call is complete.

Interacting with the Process

- The environment checks incoming messages for correlation information and determines whether they match an existing instance, or whether it has to create a new instance, or do neither.
- You can also use correlation to validate business data.
- For example, if you use a middleware-generated opaque token to identify process instances but the process that it refers to uses correlation sets, you can view the correlation information as a business-level check of the validity of an incoming message.

Interacting with the Process

- The invocations that a process instance sends to its partners are blocking.
- An invoke activity that corresponds to a request-response WSDL operation begins by sending a request and ends after it has received the response.
- The invoke activity specifies which data variable to read the request message from and which optional variable to write the response into. By the time an invoke activity must run, it must know the endpoint of the partner link to send the message to.
- As discussed earlier, multiple schemes are available for binding partnerLinks to actual deployed endpoints.

Interacting with the Process

- The result of an invocation might include an endpoint reference that you can copy using an assign activity to one of the process's partnerLinks.
- In this case, the process will use the new endpoint reference for all further interactions along that partnerLink.
- You can use an invoke activity to lookup an endpoint of a partner in a registry; that lookup is followed by an assign activity to copy the endpoint reference into a partner link.

Navigating the Process Model

- Control flow in BPEL, as noted earlier, is defined using a combination of structured activities and control links.
- You can only define control links inside of a <flow> activity, at arbitrary levels of nesting.
- An activity must wait to proceed until it knows all the link values of its incoming links. The topmost activity of a process has no incoming link, thus it is started once you create an instance of the process.

Navigating the Process Model

- To understand the processing model of the activity lifecycle, consider the states that a BPEL activity must go through.
- At first, all activities are in a default state. After an activity gets control from its parent (structured) activity and all its incoming links have fired, it evaluates its join condition to see whether it can run.
- If the join condition is TRue, the activity starts running.
- If it is a structured activity, its execution consists of running its child activities according to the prescribed semantics.
- After the activity completes successfully, it fires all its links with the values that result from the evaluation of their respective TRansitionConditions.

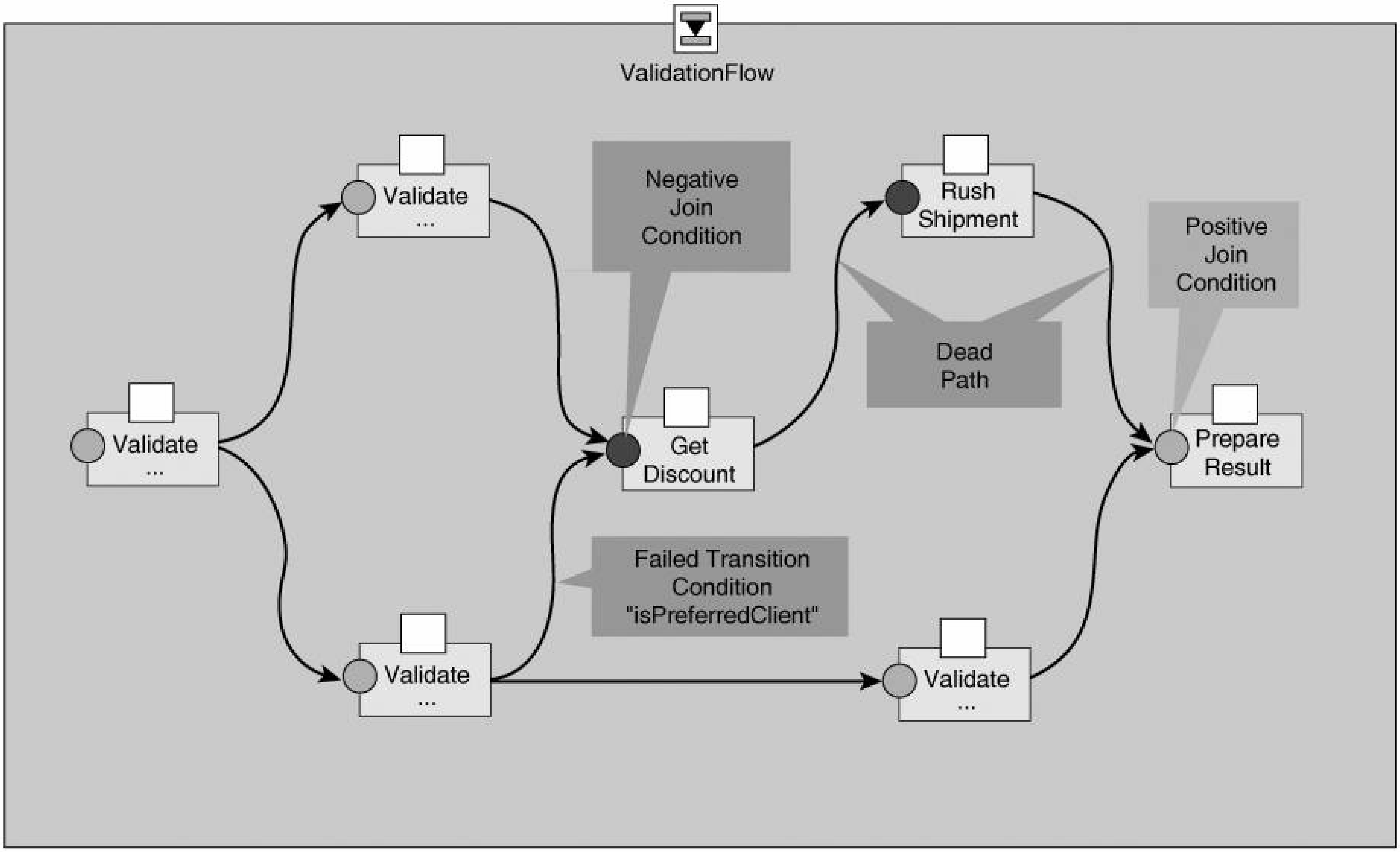
Navigating the Process Model

- For example, a sequence that has two activities begins by activating its first activity.
- After that first activity runs to completion, the sequence activates its second activity.
- After the second activity completes, the sequence is finished.
- Note that already completed activities might need to run again because of being contained in while loops.
- Sometimes an activity gets disabled, either because it faulted while it was running or because it is within a part of the process that no longer runs.
- The former includes activities whose join condition evaluates to false.
- The latter includes activities in an untaken branch of a switch or in a scope that faulted. In such situations, the activity's outgoing links fire with negative values.

Dead Path Elimination

- A BPEL flow activity contains a directed acyclic graph where the nodes are other activities and the edges are the links that define a partial execution order of the activities.
- When multiple links have the same target activity, multiple parallel execution paths are joined before this target activity is considered to run.
- A join condition determines whether the target activity is actually eligible for execution.
- When join conditions evaluate to false, you have two options: raising a fault or continuing to navigate through the flow.
- You can use both in BPEL. The actual behavior is determined by the suppressJoinFailure attribute of the process or an individual activity.

Flow graph and dead-path elimination



Dead Path Elimination

- Study the figure. If you suppress join failures by setting the corresponding attribute, the navigation of the flow continues with dead-path elimination (cf. [Ley 2000] for more details).
- The activity with the false join condition (see "Get Discount") is skipped, and the status of all outgoing links is set to false.
- For each of these links, this step is repeated for subsequent activities (see "Rush Shipment") until a join condition is reached that can be evaluated to not false.
- In the figure, this occurs at "Prepare Result" which can run successfully.

Scopes and Handlers

- Scopes are the only structured activities that have activities both nested within them as well as attached to them in the form of handlers.
- Scopes also define data. BPEL variables are always referred to by name, and that name refers to the variable in the nearest enclosing scope.
- After a scope starts running, it gives control to its enclosed activity and enables all its event and fault handlers.
- After a scope completes, it deactivates those handlers.
- If a scope completes successfully, it enables its compensation handler.
- Activities in an event handler routine run concurrently as many times as the handler is triggered. The scope that the handler is defined on remains active.

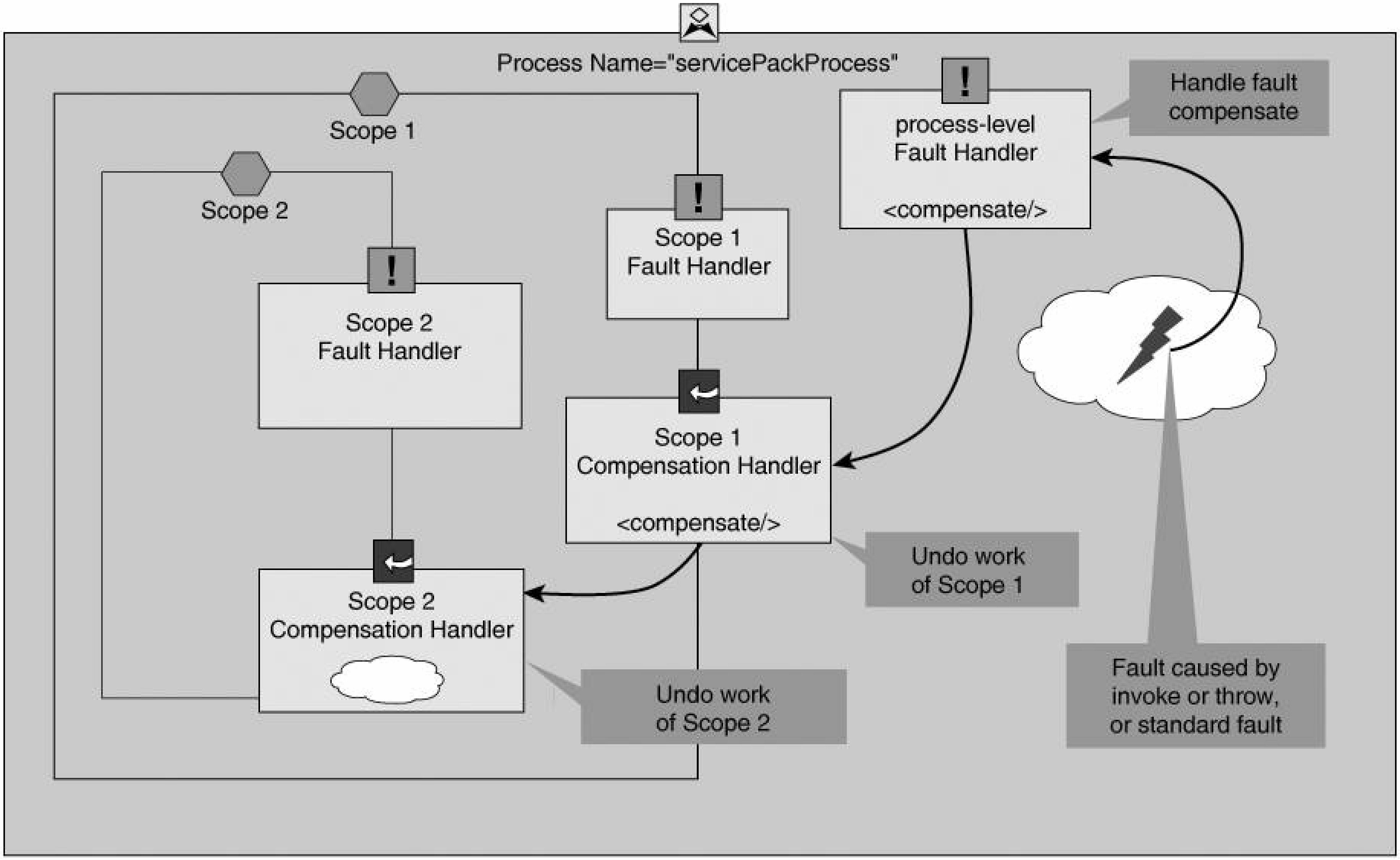
Fault Handling and Compensation

- If a fault occurs within a scope, the scope stops execution, disables its enclosed activities, and negatively fires all unevaluated links whose sources are in the scope but targets are outside of it.
- If the scope has a handler for the fault, the activities in the fault handler routine execute.
- Otherwise, the fault is thrown up the scope hierarchy until a handler is found to remedy the situation.
- If the fault reaches the process root and doesn't find a handler, it terminates the entire process.

Fault Handling and Compensation

- Each fault handler of a scope can initiate the reversal of previous activity results by invoking compensation handlers for its nested scopes.
- This applies only to nested scopes that successfully terminated on their normal execution path.
- In the next figure illustrates the interplay between fault and compensation handlers. Here, an activity in the main process scope throws a fault after the successful completion of scopes 1 and 2.
- The fault is caught by the process-level fault handler, which calls the compensation handler of scope 1, which in turn calls the compensation handler of scope 2.
- After the compensation and any additional processing in the process-level fault handler are completed, the process instance terminates.

Fault handling and compensation



Fault Handling and Compensation

- The compensation handler for a scope can operate only on a snapshot of the data that was taken at the time the scope terminated normally, and not on live data of the running process instance.
- However, the activities in a compensation handler might invoke operations that undo external effects of the activities executed in the normal path of the scope.
- If no compensation handler is defined for a scope, an implicit compensation handler invokes the compensation handlers of nested scopes in reverse order of their former completion.

Future Directions

- BPEL is currently undergoing standardization at the OASIS organization, with many companies and groups contributing to this effort and will likely be called WS-BPEL 2.0
- Web service composition, however, has many open areas for research and industrial efforts.
- A few of these areas that are already seeing some activity include checking compliance of executable processes with their abstract counterparts, global models, extensibility, and the definition of new standards.

Future Directions

- The use of abstract processes beyond the ability to formally define a business process is currently being studied. Its usage to check for compliance of a running Web service is one possible approach. It was noted earlier that you can derive an abstract process from an executable or vice versa by filling in missing parts.
- The tricky part comes from proving compliance between an executable and an abstract process.
- You can make this easier by providing harsh restrictions, such as only allowing a compliant executable to replace "empty" activities and provide the needed information for any variable's assignments that are labeled opaque. Although this drastically simplifies the problem, it is simply too restrictive to provide a general solution.
- Discussions and research are currently underway on what it means for such compliance to exist for processes in which the differences from the abstract one are more involved.

Future Directions

- Another future direction is the ability to wire together different processes or even Web services to form something that is similar to WSFL's global models.
- Although partnerLinks provide some information on what is required from a compliant partner, they do not have enough information to derive a complete picture of wirings and relationships when you consider scenarios in which multiple processes are wired together.
- For example, partnerLinks are not enough to describe the setup in which two separate business processes should use the same bank process.
- The capability to define such global models, however, is at a level of abstraction above that of BPEL and would be expected to be defined in a compatible yet separate specification.