WebSphere software

IBM

WebSphere Process Server V6.0 Business Process Choreographer Programming Model

Brian Heumann, Dieter König, Frank Neumann, Gerhard Pfau,
Stefan Rüttinger, Friedemann Schwenkreis
IBM Development Lab Böblingen, Germany

January 2006

Table of Contents

Abstract

As part of WebSphere® Process Server, V6.0, Business Process Choreographer provides support for business processes and human tasks. It offers a way to model your business process based on the WS-BPEL specification, and to model interactions that involve humans, such as human-to-human, human-to-machine, and machine-to-human interactions. Both business processes and human tasks are exposed as services in a Service Oriented Architecture. This Whitepaper introduces the programming model for processes and tasks provided by Business Process Choreographer.

1 Introduction

As part of WebSphere Process Server, Version 6.0, Business Process Choreographer (BPC) provides the support for two types of service components, business processes and human tasks. This Whitepaper provides an overview of the programming model provided for processes and tasks.

2 Service Component Architecture

Service Component Architecture (SCA) is a concept for modeling business services that consume or produce business data. The unified applications programming model for accessing and manipulating business data is provided by Service Data Objects (SDO).

SCA provides a model for implementing service components. It introduces the notion of an SCA module containing components as well as imports and exports for interactions with entities outside of the SCA module. Business processes and human tasks are represented as SCA components.

The definition of a component, import, or export is provided as an XML document in Service Component Definition Language (SCDL) format.

The key concept introduced by SDO is the data object. It holds a set of named properties containing either a value of simple data type or a reference to another data object. Data objects provide an interface for manipulating these properties.

2.1 Components

SCA components consist of interfaces, references, and an implementation. Interfaces are WSDL port types or Java™ interfaces and they describe the operations a component provides. References are also typed by WSDL port types or Java interfaces and they describe the services a component is dependent on; references are wired to interfaces provided by other components or imports. Different component implementation types are provided such as Java objects, business processes, or human tasks.

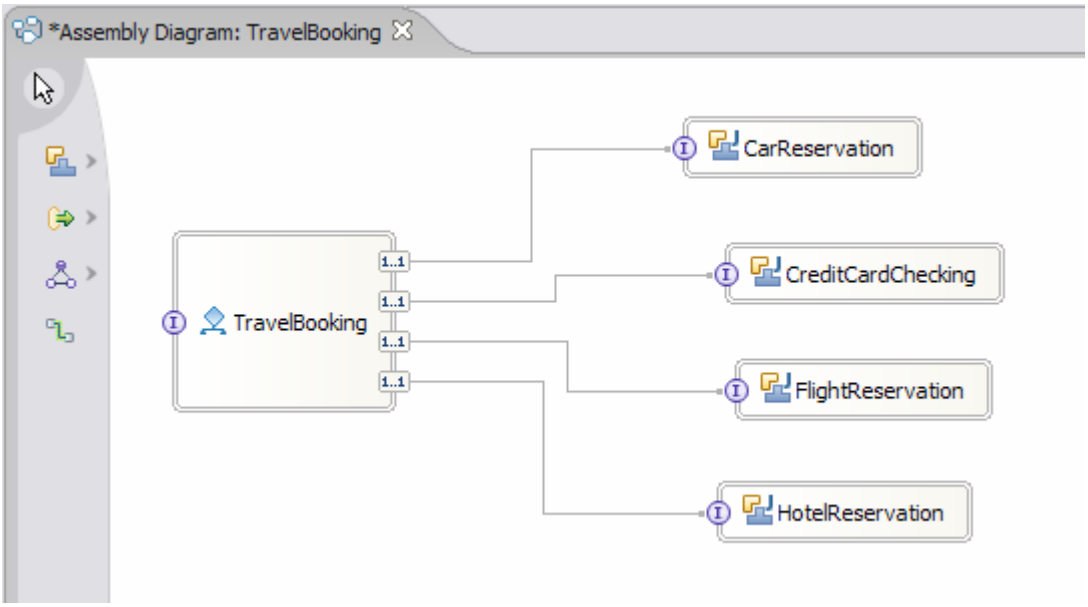

Figure 1: Service Component Architecture – Wired Components

SCA components with a ProcessImplementation implementation type represent business processes. These process components implement one or more SCA interfaces, specified using WSDL port types. From a process definition point of view, these port types define the Web service operations exposed by a process. Furthermore, process components have SCA references, again, typed with WSDL port types. These port types define the Web service operations consumed by a process. The implementation section of the process component points to the BPEL file containing the process definition.

SCA components with a TaskImplementation implementation type represent human tasks. Participating tasks have SCA interfaces only and no references. Conversely, originating tasks have only SCA references. Finally, purely human tasks have neither interfaces nor references. These tasks are used only via BPC API services and not an SCA client.

2.2 Imports and Exports

SCA imports and exports represent the external access points from and to an SCA module. Imports allow the import of external services that are not part of an SCA module. These imported services can be accessed by clients within the SCA module like any other SCA component. Exports allow the export of SCA services to clients that are not part of the SCA module containing the exported SCA service.

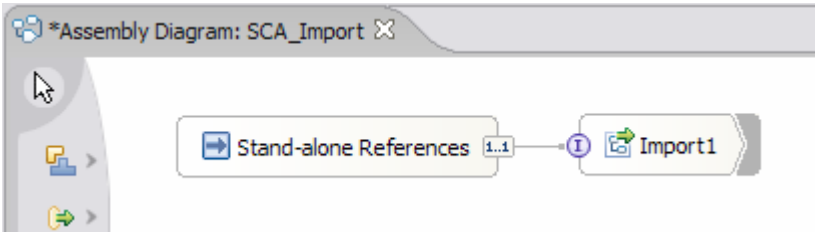

Figure 2: Service Component Architecture – Standalone Reference and Import

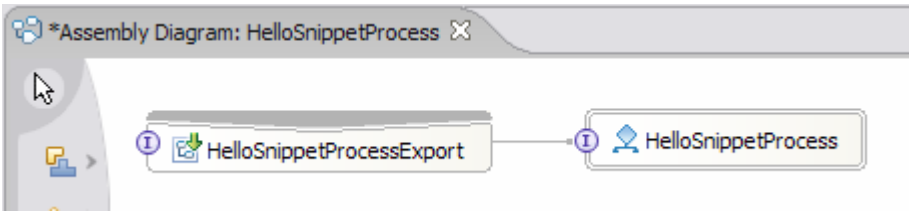

Figure 3: Service Component Architecture – Export and Exported Component

3 Business Process Programming Model

Business processes are defined using the Web Services Business Process Execution Language (WS-BPEL or BPEL). BPEL is a model and a grammar for describing the behavior of a business process based on interactions between the process and its partners. The interaction with each partner occurs through Web Service interfaces; therefore, BPEL is built on top of WSDL and XML schemas.

The following sections describe elements and attributes of a business process definition (see also [BPEL4WS 1.1], [WS-BPEL 2.0]), including standard modeling constructs and language extensions for additional features, such as user interactions, inline Java code, or quality of service (QoS) attributes.

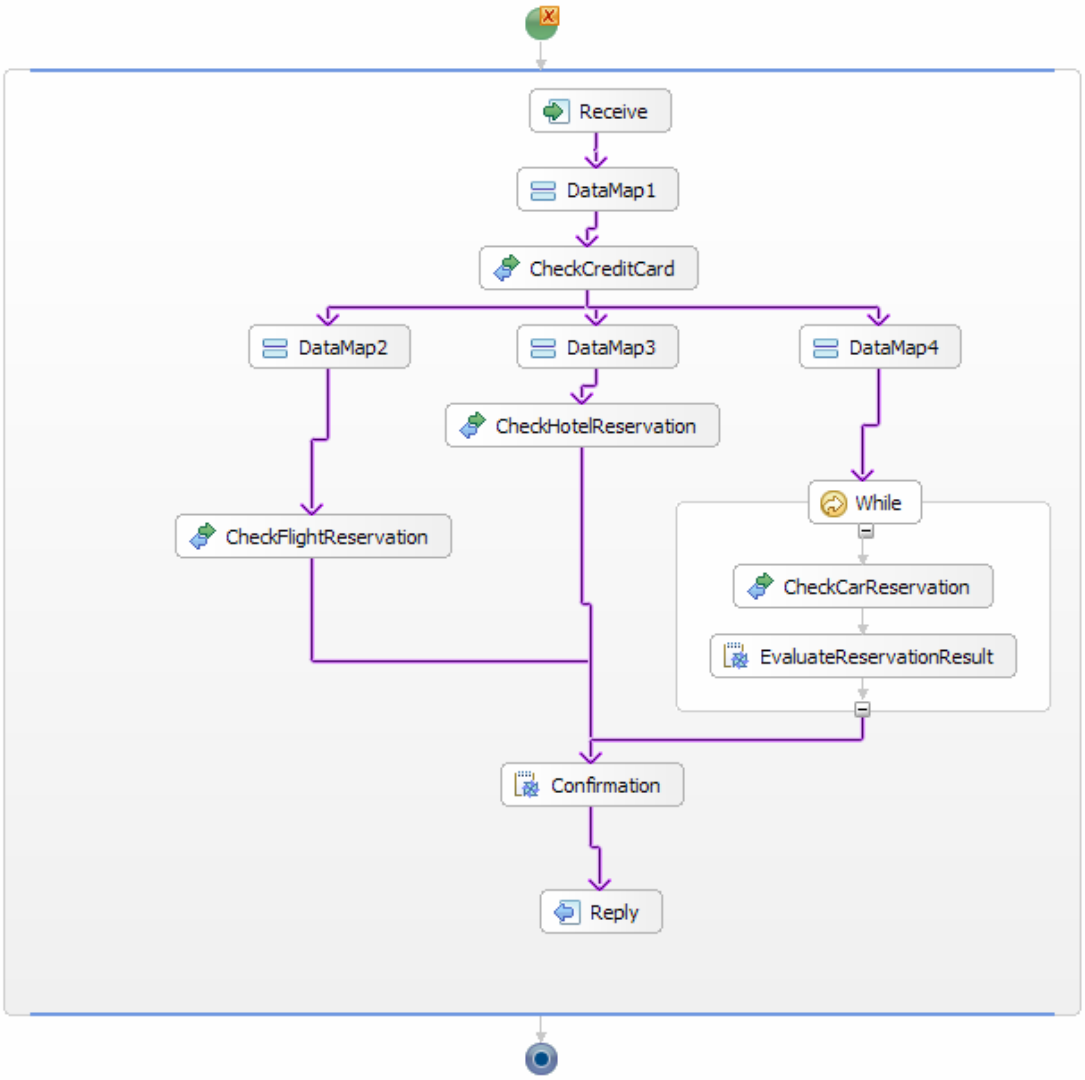

Figure 4: BPEL Process

WebSphere Integration Developer validates the process model using an extended XML schema for the syntax and a comprehensive set of rules for semantic constraints defined by the BPEL specification.

3.1 The Process


A business process definition is specified as an XML document that begins with the `process` root element. It contains process-global attributes and elements that either carry operational semantics or are descriptive in nature.

The major building blocks of business processes modeled in BPEL are nested scopes containing 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 itself.

Table 1 lists BPEL and BPEL extension attributes of the process element.

Attribute	Description
name targetNamespace	Identifies the process definition.
queryLanguage	XML query language used for selecting nodes in assignments; default "http://www.w3.org/TR/1999/REC-xpath-19991116" for XPath 1.0
expressionLanguage	XML expression language used in the process; default "http://www.w3.org/TR/1999/REC-xpath-19991116" for XPath 1.0
suppressJoinFailure	Determines whether the joinFailure fault is suppressed for all activities in the process. Tthe default for this attribute is "no" at the process level (see *Parallel Activities* later in this document, for more information about join failures). When this attribute is not specified for an activity, it inherits its value from its closest enclosing activity, or from the process if this attribute is not specified in any enclosing activity.
bpc:displayName	Readable name for the process.
bpc:id	Unique identifier used, for example, for process debugging.
bpc:validFrom	Specifies the time when the process model becomes valid, that is, the point in time from which process instances can be created.
bpc:executionMode	Specifies whether a process is executed as a long-running process or as a microflow (the default is `"longRunning"`).
bpc:autonomy	Specifies whether a process can become a subprocess of another process; by default, it is treated as an independent peer process.
bpc:compensationSphere	Determines the compensation behavior for microflows; it has the values `"supports"` (the default) and `"required"`.
bpc:businessRelevant	Indicates whether the details on the execution of this process is stored as a part of the audit trail in the runtime environment (by default yes).
bpc:autoDelete	Determines whether a long-running process instance is kept around even when the process instance has reached a final state or automatically deleted upon completion (the default).

Table 1: Attributes of the BPEL process root element

A process can also contain descriptive elements:

`bpc:description` for a short description that can be queried at runtime

`bpc:documentation` for a long explanation of the process

`bpc:annotation` used for comments provided by the process modeler.

The `import` element is used to explicitly indicate a dependency on an external XML schema or WSDL definitions, or inline task definitions. The element `bpc:customProperty` is used to add additional attributes to processes beyond those provided by BPEL. The value of a custom property can be set in the process model or at runtime, for example, by Java snippets. Java packages needed by Java snippets can be globally specified using the `bpc:javaGlobals` element. The modeler can define administrative tasks for the process using `bpc:adminTask` and for an invoke activity using `bpc:activityAdminTask`.

The following sections introduce the remaining elements of a process definition – partner links, variables, correlation sets, fault, compensation and event handlers, and the activities that describe the dynamic behavior of the process.

3.2 Partner Links

 A BPEL process interacts with other partners through Web services, along a set of partner links. Partner links are instances of typed connectors which specify the port types the process offers to and requires from the partner at the other end of the partner link.

The `partnerLinks` section defines the different parties that interact with the business process. Each `partnerLink` is characterized by a partner link type and a role name. This information identifies the functionality that must be provided by the business process and by the partner service.

For process-to-process interactions, you can annotate a partner link using the `processResolver` element to specify the process template name of the process that offers the service to be invoked. In this case, the currently valid version of that process template is determined at runtime (this is also referred to as *late binding* of subprocesses).

3.3 Variables

 Variables hold the data that constitutes the state of a business process. Data in BPEL is written to, and read from, typed variables. The values of these variables are either messages exchanged with the process or intermediate data that is private to the process. Variables are typed using WSDL message types, XML schema types, or XML schema elements. XPath is the default language for manipulating and querying variables.

Variables are declared by specifying a name, and a WSDL message type, an XML schema type, or an XML schema element. The declarations are visible either in the complete process or in scopes, which are introduced later.

In addition, variables can have the `bpc:id` and `bpc:businessRelevant` attributes. These attributes uniquely identify the variable and control the generation of Common Base Events and audit trail records during process execution.

A predefined `service-ref` data type is defined by BPEL; it is used as a wrapper for endpoint references (EPRs, see [WS-Addressing]). BPC also provides a `bpc:StandardFaultType` data type which is used for data associated with BPEL standard faults.

3.4 Correlation Sets

 If a process offers multiple Web service operations, then subsequent request messages must be routed to the correct instance. BPEL defines a correlation mechanism to route messages using parts of the application data, that is, data in input and output messages of Web services. This correlation mechanism comprises properties and correlation sets. Properties are defined in WSDL and mapped (aliased) to parts of several of the WSDL messages that the process uses. Correlation sets are defined by specifying their `name` and a group of `properties`.

You can attach zero or more correlation sets to the interaction activities using the `correlations` element. This element has a flag that determines whether the activity will `initiate` the correlation set's data.

3.5 Activities

Activities are the most important elements of a process definition and describe the business logic of the process. BPEL offers different types of basic and structured activities that are described in the following sections. All activities can carry the BPEL standard attributes `name` and `suppressJoinFailure`, and the extension attributes `id`, `displayName`, `businessRelevant`, `transactionalBehavior`, `continueOnError`, `fault`, and `compensable`. Activities can also be the source or target of a link (see *Parallel Activities* later in this document). For this purpose, the `sources` and `targets` standard elements contain the specification of the corresponding links.

Receive

 The `receive` activity is one of the activities needed for providing Web services to partners. It specifies the `partnerLink` and the WSDL `portType` and `operation` for the Web service. The specified `variable` holds the request data received from the caller of the Web service. The receive activity has one or more associated reply activities if it is used to provide a WSDL request-response operations.

When a Web service request is received, the request message can either lead to the creation of a new process instance or be consumed by an existing process instance. The `createInstance` attribute of the receive activity determines whether a new process instance can be created.

Reply


The `reply` activity, typically used in conjunction with the receive activity to implement a WSDL request-response operation, provides the means to return data to the caller of a Web service. It specifies the `partnerLink` and the WSDL `portType` and `operation` for the Web service. The specified `variable` holds the response data or fault data returned to the caller of the Web service. If fault data is returned, the `faultName` identifies the corresponding WSDL fault.

Invoke


The `invoke` activity is used to call a Web service provided by a partner. It specifies the `partnerLink` and the WSDL `portType` and `operation` for the Web service to be invoked. For WSDL request-response operations, an `inputVariable` and an `outputVariable` are specified. These variables hold the data passed to, and received from, the Web service. For WSDL one-way operations, only the `inputVariable` is needed.

The `input` and `output` extension elements allow the mapping of WSDL message contents to multiple BPEL variables. This is convenient when the Web service interaction is compliant with the document-literal wrapped style, which is the default for interactions that are created with WebSphere Integration Developer, V6.0. Each parameter element that is nested in the document is assigned to an individual BPEL variable. This approach avoids the need for variables that are defined with the wrapper document type or even the WSDL message, and it allows you to use the relevant business object types directly.

When you specify correlation sets on the invoke activity, you must also use the `pattern` attribute to specify whether the correlation set applies to the request (`"out"`) or the response (`"in"`).

The `timeout` extension is used to tell BPC when an activity is supposed to be finished, which is important for asynchronous or long-running interactions. Administrative tasks can be associated with activities using the `adminTask` extension.

You can specify fault handlers, a compensation handler, or both on the invoke activity as a shorthand notation for a scope activity that contains the handlers and the invoke activity. For more information on handlers, see sections 3.6 *Fault Handling*, and 3.7 *Compensation Handling*.

The two most important extensions, `task` and `script`, allow you to “invoke” a user interaction or run inline Java code instead of calling a Web service. These variants of the invoke activity are discussed in the *Human Task* and *Snippet* sections.

Human Task


The human task activity (`task` extension of the BPEL `invoke` activity) is a basic activity which is “implemented” by an action performed by a human being. To define the implementation of an activity involving people, tasks are used. For more information on

tasks, see section 4 *Human Task Programming Model*. You can also refer to [BPEL4People], which explains basic concepts of user interactions.

Snippet

 The snippet activity (`script` extension of the BPEL `invoke` activity) allows you to specify Java code as part of the activity implementation. This Java code has access to the enclosing BPEL environment, for instance, it can work with BPEL variables, partner links, correlation sets, and custom properties (see section 5.6 *Java Snippet Programming Model* for more information). For optimization purposes, you can specify a list of variables that are accessed read-only.

Assign

 The `assign` activity provides the means for basic data manipulation. Expressions can be used to perform simple computation. An assign activity also provides the means to map service endpoint references to or from partner links. One or more `copy` elements describe how the data is assigned, and contain `from` and `to` specifications of different types. You can specify a variable containing a WSDL message, optionally a selection of a WSDL part, or a variable containing an XML document. Alternatively, the from-spec or to-spec can be a partner link and a specification of its role; note however that you cannot modify the "myRole" endpoint reference of a partner link. Additional variants are a variable property, an expression, or a literal value (from-spec only).

Choice

 The choice activity (BPEL `switch` activity) allows you to select exactly one branch of an activity from a given set of choices. For each choice, a condition determines whether a branch is taken. As with other expressions, you can use XPath expressions, inline Java code, or predefined built-in expressions. Only the first branch with a true condition is executed. If no condition evaluates to true, then a default choice can be specified using the `otherwise` branch.

Receive Choice

 The receive choice activity (BPEL `pick` activity) allows you to block and wait for a suitable message to arrive or for a time-out alarm to go off. The `pick` activity can contain `onMessage` elements specifying the `partnerLink` and the WSDL `portType` and `operation` for the Web service to be provided, and a `variable` which holds the request data received from the caller of the Web service. You can also specify an `onAlarm` element for a time-out alarm which is processed at a specified time or after a given time interval. Duration-valued or deadline-valued time-out expressions can be specified in XPath or Java.

Like the `receive` activity, the `pick` activity carries a `createInstance` attribute which determines whether a new process instance can be created upon the receipt of a Web service request message.

While Loop

 The while loop activity (BPEL `while` activity) allows you to specify that an activity is executed repeatedly as long as a given condition evaluates to true.

Sequence

 The `sequence` activity is used to define a collection of activities which is performed sequentially in lexical order.

Parallel Activities

 The parallel activity (BPEL `flow` activity) provides for concurrency and synchronization of nested activities. It contains the `links` for specifying the (partial) execution order of contained activities. To define such a constraint, a link is referred to in the source element of one activity and the target element in another activity.

If the source activity and the target activity are nested in different enclosing activities, then the link crosses the boundary of an enclosing structured activity. In this case, several restrictions apply; links must not cross the boundary of a while activity, an event handler or a compensation handler, and they must not enter a fault handler.

A link can contain a specification of a transition condition, which is evaluated after completion of the source activity, to determine whether the link status will be positive or negative. The target activity contains a join condition, which can refer to the status of one or more inbound links. The join condition is evaluated when the status of all the inbound links is determined. If the result is true then the target activity is executed. If the result is false, a `joinFailure` fault is thrown or dead-path elimination occurs, depending on the setting of the `suppressJoinFailure` attribute. Dead-path elimination causes the target activity to be skipped and the status of all outgoing links to be set to negative.

You define transition conditions and join conditions using an expressionLanguage, which is XPath (the default), Java, or built-in.

Throw

 The `throw` activity is used for explicitly raising a fault (see 3.6 *Fault Handling*). You can associate fault data with the fault by specifying a fault variable that contains the data.

Rethrow

 The `rethrow` activity can be used inside of a fault handler to delegate the handling of a fault to an enclosing scope (see 3.6 *Fault Handling*). The fault is re-thrown exactly as it was caught by the fault handler, that is, any modifications of the associated fault data are ignored.

Compensate

 The `compensate` activity is used for explicitly invoking a compensation handler of one or more directly nested scopes (see 3.7 *Compensation Handling*). Either the compensation handler of a specified scope or, if the scope name is omitted, the compensation handler of all nested scopes is executed.

Terminate

 The `terminate` activity is used to terminate immediately the behavior of a business process instance within which the `terminate` activity is performed. All running activities are terminated without any fault or compensation handling.

Wait

 The `wait` activity is used to wait for a specified time period or until a certain point in time is reached. Again, duration-valued or deadline-valued time-out expressions can be specified in XPath or Java.

Empty Action

 The empty action (BPEL `empty` activity) is used to specify that no action is to be taken, that is, this is the BPEL rendering of a no-op activity. This activity is used for fault handlers that consume a fault without acting on it. Other use cases for the empty activity include synchronization points in a flow, or placeholders for activities that are to be added later.

Scope

 A `scope` allows you to define local variables, fault handlers, event handlers, and a compensation handler (handlers are described in the following sections). Its nested activity has access to the definitions of all of the enclosing scopes, including the outermost scope which is the process root element itself. The visibility rules known from programming languages, such as Java apply.

3.6 Fault Handling

 Faults can be returned from Web service invocations (`invoke`), explicitly raised in the process (`throw`), returned from a process (reply with fault), or recognized by the runtime infrastructure (standard faults). Fault handlers are specified on the process or scope level. When faults are caught by a fault handler, the exceptional situation can be dealt with by regular activities defined in the fault handler, by invoking compensation handlers using the `compensate` activity, or by delegating to a fault handler of an enclosing scope (`rethrow` or `throw`).

A scope can have one or more fault handlers for specific faults (`catch`) or a generic `catchAll` fault handler. If a fault handler is not specified, a default fault handler applies.

This fault handler contains only the `compensate` activity that caused the compensation handlers of the directly nested scopes to be executed.

Standard Faults

BPEL defines a number of standard faults that might be encountered during the execution of a process. In many cases, these faults are caused by modeling errors, for example, `correlationViolation` or `selectionFailure`. Business Process Choreographer recognizes additional error situations, resulting in the faults: `bpc:timeout`, `bpc:serviceTerminated`, or `bpc:runtimeFailure`.

3.7 Compensation Handling

 A scope can have an explicit `compensationHandler` or, if none is specified, a default compensation handler applies which contains only a `compensate` activity. The compensation handler can be considered to be a continuation of the scope's execution; it reverses the effects of activities as part of the "regular" scope behavior when a failure occurs.

For microflows, you can use the `undo` construct for specifying undo logic as an alternative to a `compensationHandler`. This construct provides a shorthand notation for a compensation handler that invokes a single service as its undo logic and it takes a snapshot of the input for the undo operation on successful completion of `invoke` or `scope`. The `undo` element provides a construct that can express the compensation practice known in WebSphere Integrated Server, WebSphere Application Server Extended Edition V5.0, and WebSphere Business Integration Server Foundation V5.1.

3.8 Event Handling

 A scope can have event handlers which deal with message events or timer events. Timer events are the same `onAlarm` elements as in `pick` activities (see the section on receive choice activities). However, in event handlers, these events can be processed multiple times. Message events defined by the `onEvent` element contain activities that are executed concurrently to the other activities of a scope. They also represent Web services provided by a process, and refer to a `partnerLink` and the WSDL `portType` and `operation` for the Web service, and a `variable` which holds the request data received from the caller of the Web service. Event handlers are active as long as the enclosing scope is executed. Multiple event handler instances are processed concurrently.

4 Human Task Programming Model

A human task is a component that involves a person interacting with a service. The Task Execution Language (TEL) is used to define a task. TEL defines a model and a grammar for tasks. The following sections describe the elements and attributes of a task definition.

4.1 Task

 Tasks have a set of attributes and related elements. Table 2 explains the task attributes. Subsequent sections cover custom properties, localized descriptions, staff settings, escalations, and UI settings for tasks.

Attribute	Description
allowClaimWhenSuspended	Determines if a task that has been suspended can be claimed; by default this is not the case.
applicationDefaultsComponentName	Defines the name of the application component that specifies defaults for this task.
containmentContextComponentName	Defines the name of the application component that contains this task. If a value is specified, the life cycle of this task is coupled to a specific application component.
autoClaim	Determines whether a task for which the staff resolution resolves to a single person is automatically put into the claimed state
businessRelevance	Identifies tasks that are relevant to the business thus allowing you to specifically query for these tasks.
contextAuthorizationForOwner	Defines access rights for a task to its surrounding context. By default, this attribute is set to NONE. If you set the value to READER, the task owner gets read access to the surrounding context.
defaultLocale	Specifies the default locale of the task. Use this property to specify the locale for the display name, description, and documentation that is specified as part of the task.
durationUntilDeleted	Defines how long a task that has ended is kept in the system before it is deleted. By default, this attribute is set to “0”, and the task is deleted when it ends. You can set the attribute to infinity so that the task is never deleted automatically. If you set the duration to a specific value, then task deletion is delayed by the specified amount of time; the value is interpreted as a duration string of the default WebSphere calendar, or of the calendar specified by The calendarJNDIName and calendarName properties of the task if set.
durationUntilDue	Used to calculate the due date of a task. The value

Attribute	Description
	of this attribute is interpreted as a duration string of the default WebSphere calendar, or of the calendar specified by the calendarJNDIName and calendarName properties of the task if set.
durationUntilExpires	Defines how long a task is allowed to be active before it is moved to the expired state. By default, this attribute is set to infinity so that the task never expires.You can explicitly set the expiration duration to let the task expire. The value of this attribute is interpreted as a duration string of the default WebSphere calendar, or of the calendar specified by the calendarJNDIName and calendarName properties of the task if set.
calendarJNDIName calendarName	WebSphere calendar used to specify durations. You can set either both, or neither of these attributes. If these attributes are not set, the default WebSphere calendar is used.
jndiNameStaffPluginProvider	Determinies on a per task basis which staff plug-in provider is used to interpret staff definitions. Because there is a global definition of the staff plug-in provider at runtime, setting this property is usually not required.
eventHandlerName	Defines the name of the event handler used for escalation notifications and API events.
kind	Specifies if the task is an originating task, a participating task, a purely human task, or an administrative task. Originating tasks are tasks used by humans to initiate automatic work. Participating tasks are used to involve people as the “human” implementation in service choreographies and business processes; by that an automated service can schedule work for a person or for a group of people. Purely human tasks are used by humans to schedule work for other people. Administrative tasks are used to perform administrative operations on business processes.
name targetNamespace	Identifies the task definition.
priority	Defines the priority of a task; the default value of priority is 5; the highest priority is 0.

Attribute	Description
supportsDelegation	Defines whether work items related to a task can be transferred to other people; by default, this is the case.
type	Categorizes tasks according to the business needs of a customer. The type property can be used to filter queries for tasks of a certain type.
validFrom	Defines from when the runtime should allow instantiating tasks for a certain task definition. If a value is not specified, then validFrom is set to the current time during deployment and tasks can be instantiated immediately. You can achieve basic versioning of tasks by defining tasks with the same name and targetNameSpace, but different values for validFrom.

Table 2: Attributes of a Human Task Root Element

Custom Properties

Tasks can also have any number of custom properties. Custom properties are key-value pairs; `key` and `value` are both strings. You can use custom properties to specify customer-specific information for a task. This information can be used at runtime to filter for specific criteria. For example, a company might define a custom property named “business unit”, which is assigned the value “accounting”, “sales”, or “HR”, depending on the business unit the task belongs to. You can use the information in the “business unit” custom property at runtime to filter task lists for people in each of the business units.

Localized Descriptions

Tasks provide a way to define a display name, a description, and documentation for a task. The task definition allows you to specify one display name per locale, thus providing multilingual support. The same is true for description and documentation.

Staff Settings

 Staff settings are used to define who is to do what with a certain task. From a programming model perspective, staff settings are an element of the task; they act as a container for the different staff definitions.

Staff settings can contain the staff definition for the administrators, editors, potential instance creators, potential owners, potential starters, and readers of a task. They can also contain staff definitions for queries that are used to define contact persons for a task for certain categories.

Staff settings consist of staff verbs that have a `name` and a set of `parameters`. The number of parameters depends on the staff verb. Not all staff verbs are supported for all staff directories because not all staff directories offer the same functional richness. For a detailed description of the different staff verbs and their parameters, refer to the section “Predefined staff verbs and their parameters” in the WebSphere Process Server information center (see [WPS Info Center]).

Client Settings

 You can specify client settings for tasks. Client settings are used to render the task input message or result on a certain client user interface. By default, client settings for BPC Explorer and for WebSphere Portal are supported. However, other clients can also be supported. The task definition has an extensibility mechanism that allows you to specify custom-client settings, consisting of key-value pairs, for these clients.

Escalations

Escalations provide a way to specify what should happen if a task is not progressing as expected. A task can have one or more chains of escalations. Escalation chains have one of the following activation states: ready, claimed, or waiting-for-subtask. If the task reaches a certain state, the first escalation in all of the escalation chains with this state set as the activation state is activated.

Like tasks, escalations have localized descriptions, custom properties, and staff settings to define who receives an escalation. In addition, escalations have the following attributes:

`name` identifies the escalation definition.

`atLeastExpectedState` and `durationUntilEscalation` define the basic escalation behavior of the escalation. `durationUntilEscalation` specifies the time that occurs after an escalation has been activated before the escalation fires. If in the meantime the task has progressed as expected, then it will have a state that is at least `atLeastExpectedState`. If this is the case, the escalation becomes superfluous. The escalation fires if this is not the case.

`escalationAction` defines the action that occurs when an escalation fires. This action can be creating a work item, sending an e-mail, or triggering an event on a registered task notification event handler.

`autoRepeatDuration` allows you to specify the behavior of recurring escalations. If this attribute is set, the escalation is repeated continuously after the specified duration until the escalation becomes superfluous.

`increasePriority` defines if the priority of a task should be increased on escalation. Depending on the value of `increasePriority`, the priority of the task is not increased, increased once, or increased with every repetition of the escalation. The latter applies only to tasks that have the `autoRepeatDuration` attribute set. Note that because zero is the

highest priority, to increase the priority, you must decrease the value of the `increasePriority` attribute.

4.2 Task Notification Event Handlers

Task notification event handlers are used to send escalation events and API events. The Human Task Manager offers two interfaces (`NotificationEventHandlerPlugin` and `APIEventHandlerPlugin`) that an application can implement and a mechanism to register the plug-ins with the runtime. The plug-ins are visible under a given name. You use this name to specify the `eventHandlerName` property of a task introduced before.

4.3 Application Component

Application components are a generic construct that you can use to define a task's containment context, specify defaults for a task, or define the parent context of a task at runtime. Application components are created, managed, and destroyed using an administrative interface. They don't have a representation in the Task Execution Language (TEL).

5 Interfaces to Business Processes and Tasks

WebSphere Process Server V6.0 provides multiple interfaces to business processes and human tasks. From an external client perspective, you can use interfaces provided for service component architecture components or generic BPC interfaces. Additional interfaces are provided for Java code within business processes. Users of services provided by Business Process Choreographer include:

- Integration developers (SCA programming model)
- Application developers (J2EE/Java programming model)
- Administrators (WebSphere administration model)

5.1 Service Component Architecture Client Interfaces

The Service Component Architecture (SCA) provides a common client interface for services provided by SCA components, regardless of the component's implementation, which may be a BPEL process or human task. The services provided by a process component or task component are described using WSDL port types.

The client uses the SCA service manager to locate a service, creates data objects exchanged with the service, invokes methods on the service, and finally processes output or exceptions returned from the service.

Clients can interact with services described by WSDL interfaces through the generic SCA dynamic invocation interface (DII). An SCA metadata interface is provided to allow the client to introspect the signature of the invoked service and the types of data objects exchanged with the service.

Service Component Architecture Exports

You can export SCA components to allow remote SCA clients or non-SCA clients to invoke the services provided. Non-SCA clients can be Web service applications, clients defined by the J2EE Connector Architecture (JCA), or JMS clients. SCA Web service exports are realized with the IBM Web Services infrastructure, and the services provided can be invoked using SOAP bindings. SCA exports for JCA or JMS contain specifications of data bindings that determine the wire format of the corresponding protocol.

5.2 Generic API for Processes

The generic API for processes allows you to develop applications that interact with BPEL processes. It is implemented as an enterprise bean and offered in the following renderings:

- `BusinessFlowManager` provides a remote Enterprise JavaBeans (EJB) interface
- `LocalBusinessFlowManager` interface provides a local EJB interface

Both interfaces provide the same functionality, including methods for:

- Process templates – access installed process models
- Process instances – interact with process instances
- Process life cycle – control the life cycle of process instances
- Activities – control the life cycle of activities in a process
- Variables and custom properties – access to process data

The reference to the remote home interface for process applications is shown in the following example:

```
<ejb-ref>
      <ejb-ref-name>ejb/BusinessFlowManagerHome</ejb-ref-name>
      <ejb-ref-type>Session</ejb-ref-type>
      <home>com.ibm.bpe.api.BusinessFlowManagerHome</home>
      <remote>com.ibm.bpe.api.BusinessFlowManager</remote>
</ejb-ref>
```

The following code snippet shows how to get started with the generic business process API:

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the BusinessFlowManager
Object result = initialContext.lookup(
                   "java:comp/env/ejb/BusinessFlowManagerHome");

// Convert the lookup result to the proper type
BusinessFlowManagerHome processHome = (BusinessFlowManagerHome)
```

```
                javax.rmi.PortableRemoteObject.narrow(
                   result,
                   BusinessFlowManagerHome.class);

// Access the remote interface
BusinessFlowManager process = processHome.create();

// Prepare input data for the process ...
DataObject poMessage = ...;

// Do business ...
process.sendMessage("purchaseOrderProcess",
                    "http://manufacturing.org/wsdl/purchase",
                    "purchaseOrderPT",
                    "sendPurchaseOrder",
                    poMessage);
```

For more information on the generic API for processes, refer to the Javadoc for the `BusinessFlowManagerService` or `LocalBusinessFlowManagerService` interfaces.

5.3 Generic API for Tasks

The generic API for tasks provides a way to work with any kind of task. It is also implemented as an enterprise bean and offered in the following renderings:

- `HumanTaskManager` provides a remote EJB interface
- `LocalHumanTaskManager` provides a local EJB interface
- `HumanTaskManagerDelegate` provides an interface for clients that abstracts from the use of local or remote interface use.

All three interfaces conceptually provide the same API functionality. The API allows you to create and start tasks, to claim and un-claim tasks, and to complete or fail them.

The API allows you to perform ad-hoc queries for tasks and other related objects. This capability can be used to obtain the list of tasks a person is to work on – their work list. Besides executing ad-hoc queries the API allows administrators to define, store, execute, and delete pre-defined queries. These predefined queries provide a convenient way for users to perform queries that do appropriate filtering without having to specify all of the details each time the query is run.

The API also allows you to get the input message, output message, and fault messages of a task, and to set the output or fault messages. It provides for retrieving a task's custom properties, and to get information about who acts in a certain role on a particular task. For administration purposes, the API provides functions to suspend, resume, restart, terminate, and delete tasks. Finally, the API provides methods to dynamically change the assignment of tasks to people.

The following code snippet shows how to get started with the generic API for tasks:

```
// Obtain the default initial JNDI context
InitialContext initialContext = new InitialContext();

// Lookup the remote home interface of the HumanTaskManager
Object result = initialContext.lookup(
                   "java:comp/env/ejb/HumanTaskManagerHome");

// Convert the lookup result to the proper type
HumanTaskManagerHome htmHome = (HumanTaskManagerHome)
                javax.rmi.PortableRemoteObject.narrow(
                   result,
                   HumanTaskManagerHome.class);

// Access the remote interface
HumanTaskManager htm = htmHome.create();

// Do business ...
htm.claim(tkiid);
```

For more information on the generic API for tasks, refer to the Javadoc for the `HumanTaskManager`, `LocalHumanTaskManager`, or `HumanTaskManagerDelegate` interfaces.

5.4 Administrative Interface to Processes

The WebSphere administrative console is a browser-based interface for monitoring, updating, stopping, and starting a wide variety of applications, services, and resources. You can use it to perform the following administrative tasks:

- Administering the compensation service for a server
- Querying and replaying failed messages
- Refreshing staff queries
- Enabling Common Base Events and the audit trail

In addition, the WebSphere administrative (wsadmin) scripting program is a command-line interface that enables you to run administrative commands in a scripting language and to submit scripting language programs for execution. It supports the same tasks as the administrative console. It is intended for production environments and unattended operations. You can use scripts for the following administrative tasks:

- Querying and replaying failed messages
- Refreshing staff queries
- Deleting audit log entries
- Removing unused staff queries

- Deleting process templates and task templates that are no longer valid

5.5 Administrative Interface to Tasks

Tasks can be administered using either the WebSphere administrative console or one of the administrative scripts. The administrative console and the administrative scripts are both based on the Human Task Manager JMX MBean interface. The `HumanTaskManager` MBean allows you to add and remove state observers, start and stop task templates, manage application components, refresh cached staff queries, and replay messages that have been put in the hold queue.

5.6 Java Snippet Programming Model

Java snippets in a BPEL process are either activities or expressions that contain inline Java code. In both cases, the Java code can access objects defined in the enclosing BPEL process, such as variables and variable properties, partner links, correlation sets, custom properties, and process state information. In WebSphere Process Server V6.0, these objects are either data objects or Java objects that represent simple types.

BPEL variables are used in Java snippets in the same way as if they were declared as local Java variables in the enclosing Java method. The mapping from the XML schema type to the corresponding Java type is determined by mapping rules defined in [SDO].

The following example shows a Java snippet condition of a BPEL `while` activity. The inline Java code accesses a Boolean attribute “response” of a data object that represents the BPEL CarReservationOutput variable.

```
<bpel:condition>
   <![CDATA[
      boolean condition = false;
      if(CarReservationOutput != null) {
         condition = CarReservationOutput.getBoolean("response");
      }
      return !condition;
   ]]>
</bpel:condition>
```

Access to BPEL partner links is provided through the `getServiceRefFromPartnerLink` and `setServiceRefToPartnerLink` generic methods that return and accept service references for a specified partner link. Service references are wrappers containing endpoint references (EPRs; see [WS-Addressing]). The Java type of the EPR, `com.ibm.websphere.sca.addressing.EndpointReference`, is provided by SCA. The code in the following Java snippet activity retrieves a service reference which contains the endpoint reference associated with the myPL partner link.

```
<bpel:script>
   <![CDATA[
```

```
      DataObject mySRef =
         getServiceRefFromPartnerLink( "myPL",
                                       PARTNER_LINK_MY_ROLE);
      ...
   ]]>
</bpel:script>
```

Additional generic getter and setter methods are provided for variable properties, correlation set properties, and custom properties, again, following the type mapping rules in [SDO].

5.7 Business Process Choreographer Explorer Components for JavaServer Faces

Business Process Choreographer Explorer is the generic user interface for managing and administering process instances and task instances. Because the application is generic, there is a demand for developing customized clients to support customer-specific business processes better.

In WebSphere Process Server, V6 the BPC Explorer was designed for "dual use": the first is the application itself as the process administration client and the other as a framework to create custom business process clients. For this purpose, BPC Explorer is based on the JavaServer Faces (JSF) framework, which allows the use and reuse of JSF components. The BPC Explorer is built on specialized components that application developers can reuse to build their own customized JSF applications. Figure 5 shows how these components are used in the BPC Explorer interface.

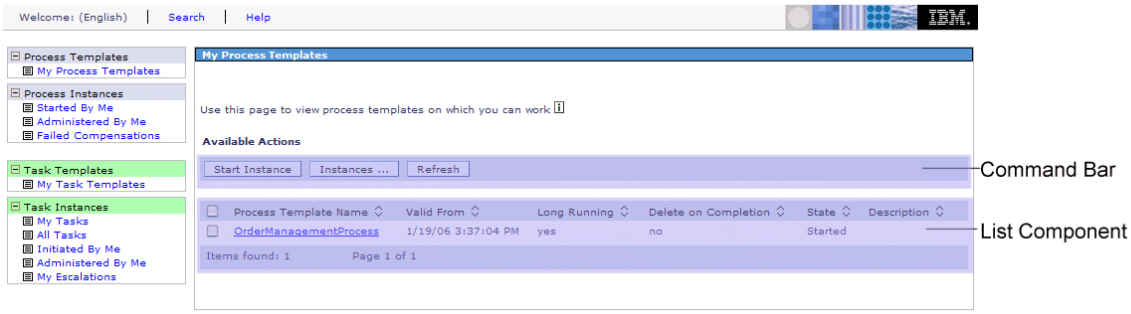

Figure 5: Business Process Choreographer Explorer Components

The components include:

- List component. This component displays a list of BPC application objects, such as tasks and processes. It has national language support, and support for custom converters to display the application objects. You can configure this list with different queries.

- Details component. This component displays the properties of an application object, for example, a process activity. By default it supports national languages, for example, through default labels for fields and converters for particular properties.
- Command-bar component. This component displays a set of command buttons that operate on the selected objects in either the list or details component. Application developers can add commands to, or remove commands from their applications to integrate specialized functions that are not part of BPC, for example, the integration with other back ends.
- Message component. This component displays the input and output business objects messages for task instances, process instances, and activities. The message component renders the `commonj.sdo.DataObject` parts and primitive types, such as integers and strings in a JSF application. If the message type is primitive, a label and an input field are rendered. If the message type is a `DataObject`, the component traverses through the `DataObject` and can render the following elements:
- primitive elements (or leaf elements)
- nested elements
- arrays
- sequences

Application developers can embed these components in their JSF pages and integrate their own functions to work on the selected objects, for example, add command buttons with customer-specific functions. The reuse of the BPC Explorer components cuts the development time, because it leaves all of the integration issues to the components.

5.8 Monitoring and Auditing

Business Process Choreographer can be divided into two subcomponents: the Business Flow Manager (BFM) that copes with business processes and the Human Task Manager (HTM) that handles the interaction with human participants. BFM and HTM manage the state changes of the objects they host. These objects are processes, activities, and variables for BFM, and tasks and escalations for HTM. Both components provide state observers that externalize the occurrence of these state changes.

Two state observers are available with WebSphere Process Server:

The *audit state observer* writes audit trail records persistently to the underlying relational database (see database views). The records are written under the same transactional protection as the state changes inside the component. Thus, it is guaranteed that the records in the audit trail are consistent with the execution steps in the component.

The *CEI state observer* generates events in the Common Base Event format that are emitted using the Common Event Infrastructure (CEI).

Enabling Monitoring

The externalization of state changes is controlled by a separate monitoring specification in a `.mon` file. A monitoring specification can distinguish between the state observers to allow separate specifications for each of them. By specifying references to the elements in the BPEL or TEL file (*EventPoints*), you can define for which elements state changes are to be externalized. Based on predefined state changes of an element (*EventNature*), you can also focus on specific state changes rather than externalizing all of them. For business processes, certain defaults are defined for elements that result in the externalization of state changes even if the corresponding *EventPoint* is not contained in the monitoring specification.

Consuming State Changes

There are two ways to consume externalized state changes depending on the state observer that is used. If you use the audit state observer to externalize the information, then the state changes are written to the underlying relational database. To access the records, you can use SQL select statements to access the audit log view (see 5.9, Database Views).

If you use the CEI state observer, use the CEI API to access event data. There are two modes offered by CEI to consume events. The query mode is based on XPATH queries. An application can use the API with SQL-like queries to retrieve events. The subscription mode of CEI allows multiple subscribers to register with certain events. Thus, the events are pushed to a subscribing application as soon as they arrive.

5.9 Database Views

BPC uses a relational database system to store runtime information. A number of database views are published as part of the BPC programming model to access the data stored in the underlying tables. Reasons why you want to use these views include:

- You don’t want to grant specific access rights, such as the J2EE monitoring role or a process reader role, to users of your application and thus the users cannot access certain data within their context
- You want to include information from your application’s database tables, such as a side table, and join them efficiently with process information in BPC
- You want to run enhanced aggregation functions or functions which are not supported by the `query()` and `queryAll()` API functions
- You want to invoke database system specific functions or call stored procedures together with BPC table information

Note that the published database views also serve as the basis for the `query()` and `queryAll()` functions. Table 3 shows the process-related database views and 4 shows the task-related database views.

Database view name	Information per row

Database view name	Information per row
PROCESS_TEMPLATE	Process template (definition)
PROCESS_INSTANCE	Process instance (based on a template)
PROCESS_ATTRIBUTE	Attribute of a process instance
ACTIVITY	Activity instance in a process
ACTIVITY_ATTRIBUTE	Attribute of an activity instance
ACTIVITY_SERVICE	Service for an activity instance in a process which is waiting for a message or an event
AUDIT_LOG_B	Audit log event for a process, if enabled

Table 3: Process-Related Database Views

Database view name	Information per row
TASK_TEMPL	Task template (definition)
TASK_TEMPL_CPROP	Custom property for a task template
TASK_TEMPL_DESC	Localized description for a task definition
TASK	Task instance
TASK_CPROP	Custom property for a task instance
TASK_DESC	Localized description for a task instance
ESC_TEMPL	Escalation template (definition)
ESC_TEMPL_CPROP	Custom property for an escalation template
ESC_TEMPL_DESC	Localized description for an escalation definition
ESCALATION	Escalation instance
ESCALATION_CPROP	Custom property for an escalation instance
ESCALATION_DESC	Localized description for an escalation instance
APPLICATION_COMP	Registered application component
TASK_AUDIT_LOG	Audit log for tasks, if enabled
WORK_ITEM	Information about the assignment of a task or the authorization to a user

Table 4: Task-Related Database Views

To access one or more database views in your J2EE application, look up the data source defined for BPC (define and use a resource reference, if appropriate), obtain a database connection, and use JDBC statements:

```
InitialContext ctxt = new InitialContext();
// Use resource ref here, if appropriate
// Note that JNDI lookup name is different when running
// in an ND environment
DataSource ds = (DataSource) ctxt.lookup(“jdbc/BPEDB”);
Connection con = ds.getConnection();
Statement stmt = con.createStatement();
ResultSet result = stmt.executeStatement(
                    “SELECT NAME FROM PROCESS_TEMPLATE”);
while( result.next() ) {
   System.out.println( result.getString(1) );
}
result.close();
stmt.close();
con.close();
```

Use join predicates if you want to combine information from multiple views. A typical join condition includes a where clause for the ID columns..

The `query()` and `queryAll()` API functions provide convenient conversion functions for binary data types and formats for timestamps and dates. If you access the views using JDBC, note that timestamp and date information is stored as UTC and that IDs, such as the `PIID` column for process instances, are stored in a binary format. In order to get the required byte array for a JDBC SQL statement, you use the `toByteArray()` method which is available for all ID Java objects.

Whenever possible, use a less restrictive isolation level, such as uncommitted read, to run concurrent SQL statements for the database views, because locks and lock waits might impact overall system performance and they can also affect process navigation.

Further details of the BPC database tables are not published and you must not access them directly because they are subject to change without notice in future releases. Furthermore, changing data in the tables directly can lead to unpredictable results.

6 Business Process Development Tools

WebSphere Integration Developer (WID) is the graphical front end for creating integrated applications containing business processes and human tasks that run in WebSphere Process Server. WID is based on Rational Application Developer (RAD), which itself is based on the WebSphere Studio Workbench, powered by Eclipse technology.

WID extends the WebSphere Studio Workbench with a set of editors and tools, including an assembly editor, human task editor, process editor, process debugger, and an integration test client.

WID uses two special kinds of Eclipse projects to store the artifacts, called “module” and “library”. Modules can contain all kinds of artifacts and result in an enterprise archive (EAR)

file for installing on the server. Libraries contain reusable artifacts, such as business objects and interfaces. They end up as utility Java archive (JAR) files in the module EAR file.

6.1 Assembly Editor

You can use the assembly editor to build applications by assembling the Service Component Architecture (SCA) components. The assembly editor is the graphical front end for the SCA programming model, and displays and edits the SCDL (.component) files for each of the components of the module.

When you open a module assembly with the assembly editor, you can visually compose the integrated application by adding components and connecting them with wires in the editor view. Both business processes and human tasks are kinds of these components.

You can either drag and drop existing processes and tasks from the Business Integration view into the diagram (bottom-up approach), or you can create new components from the palette, wire them, and then generate the corresponding implementations (top-down approach).

6.2 Business Process Editor

The process editor is a visual tool that lets you model a business process. You can add nodes to control the sequence of the execution and nodes to invoke services and human tasks. You can specify definitions to handle external events, faults, and compensation, the so-called *handlers*. You can also define the data used within the business process, which can be based on a WSDL message and an XSD schema.

Figure 6 shows the Business Process Editor. On the right side of the editor, you find the process [see 3.1], the partner links [see 3.2], the variables [see 3.3] and the correlation sets [see 3.4]. The palette on the left side of the editor contains all the activities that you can use within the process [see 3.5], including human tasks [see 4.1]. To add fault handlers [see 3.6], a compensation handler [see 3.7] and event handler [see 3.8] to the process, you should use the hover help icons for nodes where you want to add them.

Both a bottom-up and a top-down approach to modeling the process are possible. You can start either by creating placeholders (empty action nodes) in the process and refine them, or you can use existing Web service definitions (WSDL) in your business process.

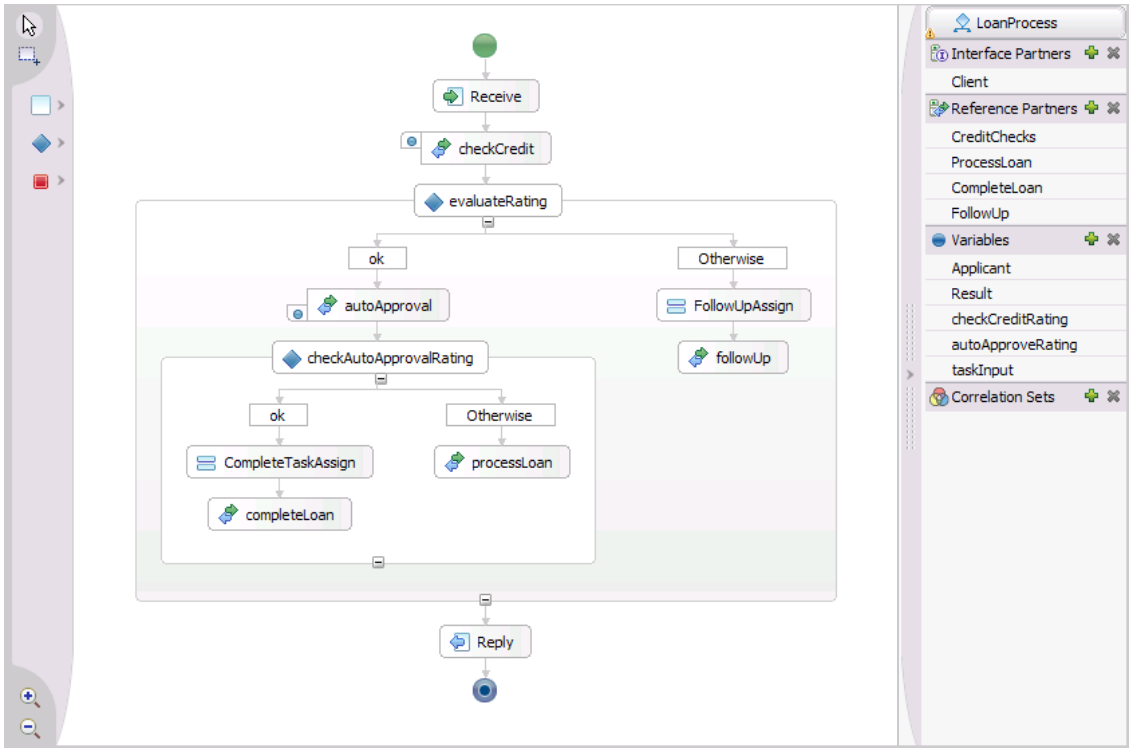

Figure 6: WebSphere Integration Developer – Business Process Editor

6.3 Human Task Editor

You can use the human task editor to visually compose services that interact with human participants. These services can be defined either within a business process (*inline tasks*) or as human task components (*standalone tasks*).

You define the following aspects of human tasks in the editor [see 4.1]:

- Who has access rights to these tasks: in the receiver settings, you can specify a staff query for each of the predefined roles, which defines the set of people that are allowed to access the task and the access rights that this set of people have.
- How the task is visualized: in the client settings, you can define how a task is presented to the user. For the BPC Explorer, you can define a custom JSP to show the input and output message of the task. You can use the published client extension point of WID to extend the list of clients with your own client type.
- What happens when tasks take too long: in the escalation settings, you can define what happens when the task takes longer than expected.

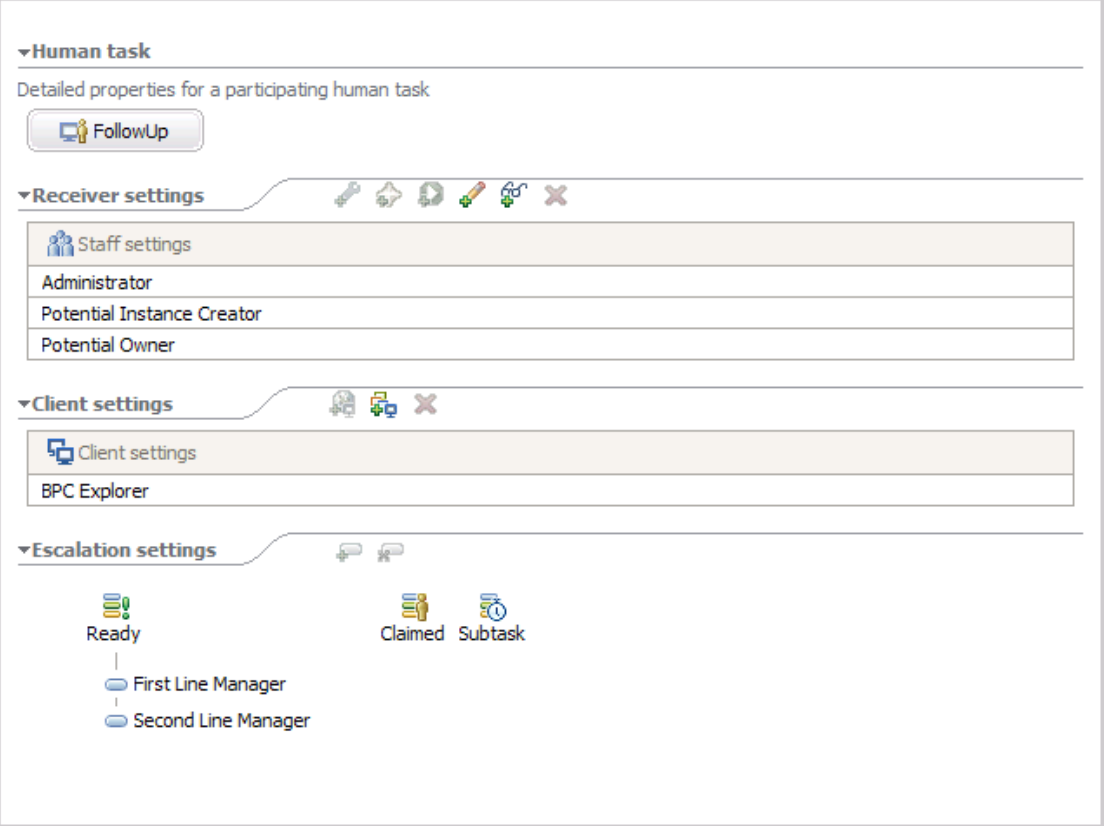

Figure 7: WebSphere Integration Developer – Human Task Editor

6.4 Integration Test Client

You can test your modules and components in WID using the integration test client. It tests the interface operations in your components, which enables you to determine whether your components are correctly implemented and whether your references are correctly wired.

You can test a component, such as a business process or human task, a set of wired components, the complete module, and even a set of connected modules (using imports and exports). The integration test client is integrated into the assembly editor. When you start the test, you are prompted for interface input data for the selected component.

Even if the wiring is not complete, you can test a component by using a manual emulator. This allows to you test a business process, for example, although the implementation is incomplete. You can provide the emulator with the expected response data of these references and continue testing.

To run the integration test client, WebSphere Process Server must be available, either using the test environment (WTE) in WID or as a standalone server.

6.5 Debugging Processes

WID provides a graphical business process debugger which you can use to test and debug your business processes. You can debug the control flow in the business process, view and manipulate data, and step into the Java expressions, for example, control links, loops, and snippets.

To run the debugger, WebSphere Process Server must be available, either using the test environment (WTE) in WID or as a standalone server.

As with a Java debugger, the process debugger allows you to set breakpoints and to view and change data. Breakpoints are always set either before an activity (entry breakpoint) or after an activity (exit breakpoint). The actual values of each variable are shown in a separate view and can be changed interactively while you are debugging a process.

You can step over an activity, step into Java expressions (which will open the Java debugger for you) as well as run to the next breakpoint or to the end of the process.

Appendix A SCA Qualifiers

SCA qualifiers can be associated with interfaces, references, and the implementation of a component. If this component is implemented by a process or a human task then several constraints apply to the values of these qualifiers, depending on the execution mode of the process (microflow vs. long-running) or the type of the task (participating, originating, purely human).

Qualifiers for Business Process Components

Table 5 summarizes the process component qualifier settings. These are initial values if they are not marked as mandatory.

SCA attribute / qualifier type	SCA attribute / qualifier	Long-running process	Microflow
interface attribute	preferredInteractionStyle	async (mandatory)	any (one-way), sync (request-response)
reference attribute	multiplicity	1..1 (mandatory)	1..1 (mandatory)
reference qualifier	deliverAsyncAt	commit (mandatory)	call
interface qualifier	joinTransaction	false (mandatory)	true (mandatory*)
reference qualifier	suspendTransaction	false	false (*)
implementation qualifier	transaction	global (mandatory)	global (mandatory*)

SCA attribute / qualifier type	SCA attribute / qualifier	Long-running process	Microflow
interface qualifier	joinActivitySession	n/a	true (mandatory**)
reference qualifier	suspendActivitySession	n/a	false (**)
implementation qualifier	activitySession	n/a	true (mandatory**)
implementation qualifier	securityIdentity	(any value)	(any value)
interface qualifier	securityPermission	(any value)	(any value)
reference qualifier	requestExpiration	(any value)	(any value)
reference qualifier	responseExpiration	(any value)	(any value)
reference qualifier	reliability	assured	assured

Table 5: Qualifiers for Business Process Components

Notes:

Activity session qualifiers are only applicable to microflows.

Global transaction qualifiers (*) and activity session qualifiers (**) are mutually exclusive.

If activity sessions are used, then the implementation qualifier transaction must be specified with value=”local”, localTransactionBoundary=”activitySession”, and localTransactionResolver=”container”.

Qualifiers for Human Task Components

Task-specific SCA qualifier settings are summarized in Table 6. Again, the values shown are initial values only if they are not marked as mandatory.

SCA attribute / qualifier type	SCA attribute / qualifier	Participating Task	Originating Task	Purely Human Task
interface attribute	preferredInteraction Style	async (mandatory)	n/a	n/a
reference attribute	multiplicity	n/a	1..1 (mandatory)	n/a
reference qualifier	deliverAsyncAt	n/a	commit (mandatory)	n/a
interface qualifier	joinTransaction	false (mandatory)	n/a	n/a

SCA attribute / qualifier type	SCA attribute / qualifier	Participating Task	Originating Task	Purely Human Task
reference qualifier	suspendTransaction	n/a	false	n/a
implementation qualifier	transaction	global (mandatory)	global (mandatory)	global (mandatory)
interface qualifier	joinActivitySession	n/a	n/a	n/a
reference qualifier	suspendActivitySession	n/a	n/a	n/a
implementation qualifier	activitySession	n/a	n/a	n/a
implementation qualifier	securityIdentity	(any value)	(any value)	(any value)
interface qualifier	securityPermission	(any value)	n/a	n/a
reference qualifier	requestExpiration	n/a	(any value)	n/a
reference qualifier	responseExpiration	n/a	(any value)	n/a
reference qualifier	reliability	n/a	assured	n/a

Table 6: Qualifiers for Human Task Components

Appendix B References

[BPC in WebSphere] Business process choreography in WebSphere: Combining the power of BPEL and J2EE, IBM Systems Journal, Volume 43, Number 2, 2004, M. Kloppmann, D. König, F. Leymann, G. Pfau, and D. Roller, available via http://www.research.ibm.com/journal/sj/432/kloppmann.html

[BPC Samples] Business Process Choreographer Samples, available via http://publib.boulder.ibm.com/bpcsamp/index.html

[BPEL4WS 1.1] Business Process Execution Language for Web Services Version 1.1, BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems, May 2003, available via http://www-128.ibm.com/developerworks/library/specification/ws-bpel/, http://ifr.sap.com/bpel4ws/

[BPEL4People] WS-BPEL Extension for People, a joint IBM-SAP whitepaper, August 2005, available via http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

[BPEL-SPE] WS-BPEL Extension for Sub-Processes, a joint IBM-SAP whitepaper, October 2005, available via http://www-128.ibm.com/developerworks/webservices/library/specification/ws-bpelsubproc/

[SDO] Service Data Objects, BEA Systems, IBM, June 2005, available via http://www-128.ibm.com/developerworks/library/specification/j-commonj-sdowmt/index.html

[SOA PM] Introduction to the IBM SOA programming model, D. Ferguson, M. Stockton, available via http://www-128.ibm.com/developerworks/webservices/library/ws-soa-progmodel/index.html

[WPS Info Center] WebSphere Process Server Product Documentation, available via http://www-306.ibm.com/software/integration/wps/library/infocenter/

[WS-Addressing] Web Services Addressing, W3C Specification, August 2004, available via http://www.w3.org/Submission/2004/SUBM-ws-addressing-20040810/

[WS-BPEL 2.0] Web Service Business Process Execution Language Version 2.0, Working Draft, July 2005, OASIS Technical Committee, available via http://www.oasis-open.org/committees/wsbpel

[WSDL 1.1] Web Services Description Language (WSDL) Version 1.1, W3C Note, available via http://www.w3.org/TR/2001/NOTE-wsdl-20010315

[XML Schema Part 1] XML Schema Part 1: Structures, W3C Recommendation, October 2004, available via http://www.w3.org/TR/xmlschema-1/

[XML Schema Part 2] XML Schema Part 2: Datatypes, W3C Recommendation, October 2004, available via http://www.w3.org/TR/xmlschema-2/

[XML] XML Specification, W3C Recommendation, February 1998, available via http://www.w3.org/TR/1998/REC-xml-19980210

[XPATH 1.0] XML Path Language (XPath) Version 1.0, W3C Recommendation, November 1999, available via http://www.w3.org/TR/1999/REC-xpath-19991116

Appendix C Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both: IBM, WebSphere.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.