WebSphere software

IBM

# WebSphere Process Server V6.1 Business Process Choreographer: Concepts and Architecture

Michael Friess, Erich Fussi, Dieter König, Gerhard Pfau, Markus Reichart, Stefan Rüttinger, Claudia Zentner

IBM Development Lab Böblingen, Germany

June 2008

**Abstract**

Business Process Choreographer is the component in IBM® WebSphere® Process Server that provides support for business processes and human tasks. It allows you to model your business process using the Web Services Business Process Execution Language (WS-BPEL), and to model interactions that involve humans using human tasks. Both business processes and human tasks are exposed as services in the Service Component Architecture (SCA). This whitepaper introduces the concepts and architecture for business processes and human tasks as provided by Business Process Choreographer.

**Table of Contents**

# 1 Business Processes and Human Tasks in Today's Business Environment

## 1.1 Business Processes in a SOA

Service-Oriented Architecture (SOA) is everywhere these days [SysJrnl SOA]. SOA allows for business flexibility – the ability to respond whenever necessary to changing market dynamics. SOA implementations create the foundation for an on demand environment built on reusable components.

By transforming monolithic applications into business processes composed of services, SOA allows you to quickly adapt applications to changing business needs – you can add new services, combine services, or disengage services you don't need to address changing requirements.

Business logic and application data are scattered throughout the organization across a multitude of software assets. Much of it resides in databases, packaged applications (such as enterprise resource planning (ERP) systems) and other back-end systems (such as IBM CICS). Other business logic can be found in existing Java and J2EE applications.

Business Process Management (BPM) offers a service-oriented approach to application development, based on orchestration of services. You can take existing software assets and quickly define how those assets are used in a new integration application. For example, you might want to combine customer information from a packaged CRM solution and J2EE components from an existing customer-facing application with new business logic to create a new Web-based order entry application. You could then extend the availability of this application by exposing it as a Web service for your business partners, enabling its incorporation into their processes.

You can visually orchestrate the interactions between various service components. You can work more efficiently because this is a standard way of representing and interacting with virtually any service component and do not, therefore, need to spend time working with different interfaces and low-level APIs. Graphical editors allow defining the sequence and flow of information between service components. Individual service components become building blocks that you can reuse in developing other integration applications.

## 1.2 Involving People through Human Tasks

Many enterprises are looking at ways to automate their business processes. Fully-automated processes without human interaction can be executed at low cost, with high throughput and short response times.

However, many real-world processes cannot be fully automated. For example, special approvals might have to be given by an authorized manager, rather than automatically by a machine, or people need to be involved in cases where an exception condition is detected that cannot be resolved by the process automatically. Therefore, it is common for business processes to consist of a combination of automated steps and steps with human interaction.

*Human Tasks* allow for the incorporation of humans in arbitrary service-based applications. Human tasks are components that allow people and services to interact with each other. A Human tasks can be included in service based applications or business processes just like any other service.

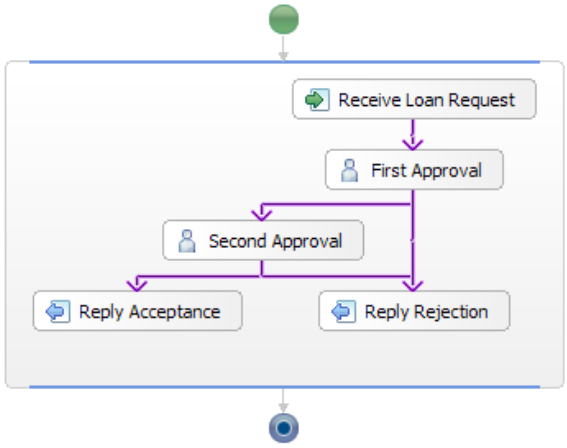


**Figure 1: Human workflow**

Within business processes human tasks can be included as activity implementations. Figure 1 shows an example of a loan approval process that includes human steps. The process consists of two subsequent approval steps. First approval and second approval are performed by different people.

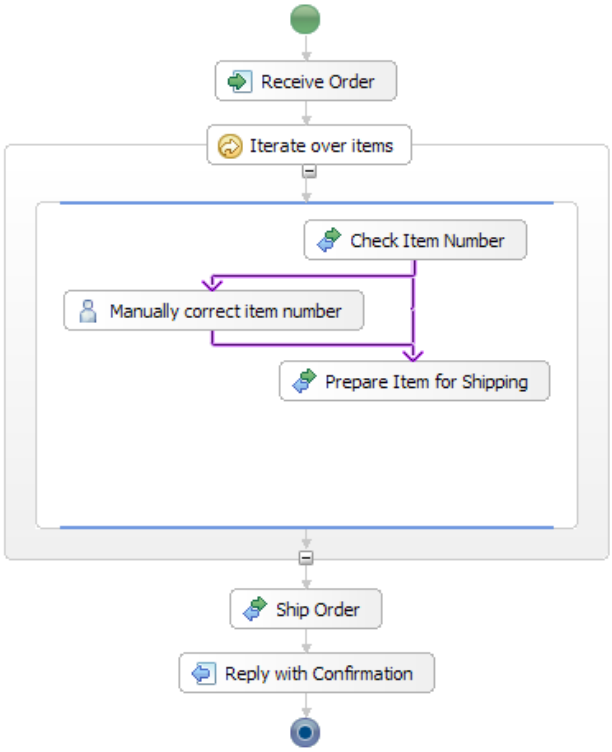


**Figure 2: Human-based exception handling**

Figure 2 shows an automated order process that uses human interaction for exception handling. Assume that in most cases orders entered into the system are correct and processed fully automatically. There is a chance though that the order contains items where the item number is incorrect. Instead of excluding the item from the order, or rejecting the entire order, a human step is introduced. A knowledgeable person gets assigned a task to manually correct the item number. That person does that by using her expertise about available items, possibly calling the sender of the order by phone and clarifying with them.

Both processes involve humans with business processes. While in the first example the entire process is based on the actions humans perform, the second example shows a fully automated process with human-based exception handling.

## 1.3 WebSphere Process Server and WebSphere Integration Developer

WebSphere Process Server (WPS) provides the runtime infrastructure to execute business processes and human tasks. WebSphere Integration Developer (WID) is the modeling tool to author business processes, human tasks, various other service components, and entire applications built from these components. This paper focuses on business processes and human tasks, which are provided by *Business Process Choreographer*, a part of WPS (see the red ellipse in Figure 3). WPS and WID support additional service components and supporting services to allow you building your integration applications, an overview of which you can see in the following picture:

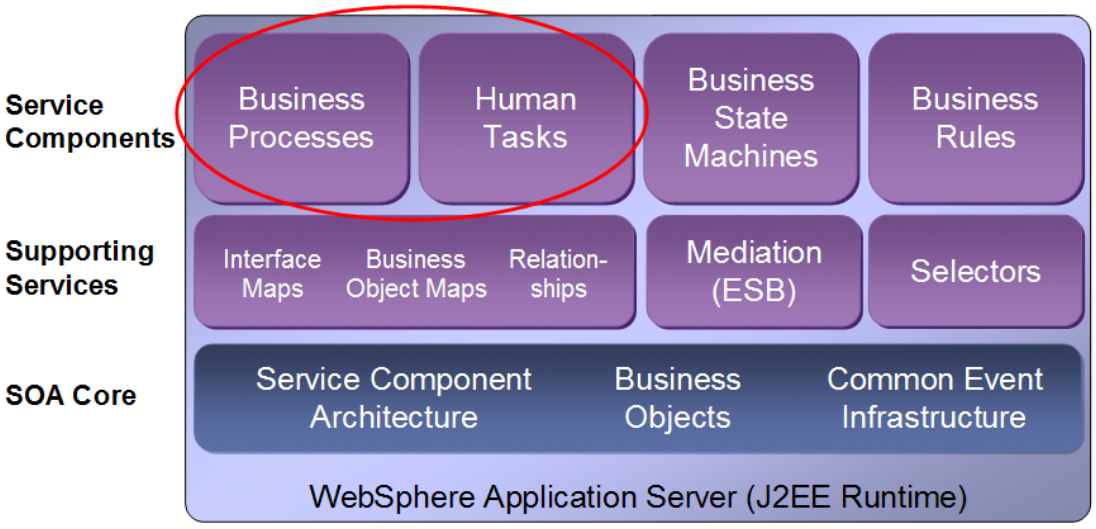


**Figure 3: WebSphere Process Server components**

WebSphere Integration Developer has been designed as an integration development environment for developers building integrated applications. To simplify and accelerate the development of integrated applications, this environment provides a layer of abstraction that separates the visually-presented artifacts you work with, e.g. the business process, from the underlying implementation, e.g. the WS-BPEL definition.

# 2 Service Components

Service Component Architecture (SCA) is a specification for modeling business services that consume and/or produce business data in a Service Oriented Architecture.

Business Objects based on Service Data Objects (SDO) provide the unified application programming model for accessing and manipulating business data.

## 2.1 Service Component Architecture

In SCA, a business application is assembled from components that implement business logic. These components offer their capabilities through *interfaces* and consume operations offered by other components through *references*.

SCA has two major purposes: The implementation of components which provide services (and typically consume other services as part of the implementation) and the assembly of sets of components into business applications through *wiring* which describes usage of components by each other.

SCA decouples service implementation and service assemblies from the details of infrastructure capabilities and from the details of the access methods used to invoke services.

SCA provides a model for implementing service components. It introduces the notion of a *module* containing one or more components as well as *imports* and *exports* for interactions with entities outside of the module. References of components are wired to interfaces of other components. Business processes and human tasks are represented as components and can be used as part of the assembly of a module, together with other component types. See also chapter “Assembly Diagram” on page 31.

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

### 2.1.1 Components

Components consist of interfaces, references, and an implementation. Interfaces can be specified as either WSDL port types or Java interfaces and describe the operations a component provides. References are also typed by WSDL port types or Java Interfaces and describe what services a component is dependent on. Within an assembly, references are wired to interfaces provided by other components or imports. Different component implementation types are supported, such as Java objects, BPEL processes, human tasks, or many others.

Component implementations are concrete implementations of business function which provide services and/or consume services. SCA allows you to choose from any one of a wide range of implementation technologies, including Java, BPEL processes, or human tasks.

Components with implementation type “ProcessImplementation” represent business processes. These process components implement one or more interfaces, specified using WSDL port types. These port types define the Web service operations provided by a process. Furthermore, process components have references, also 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.

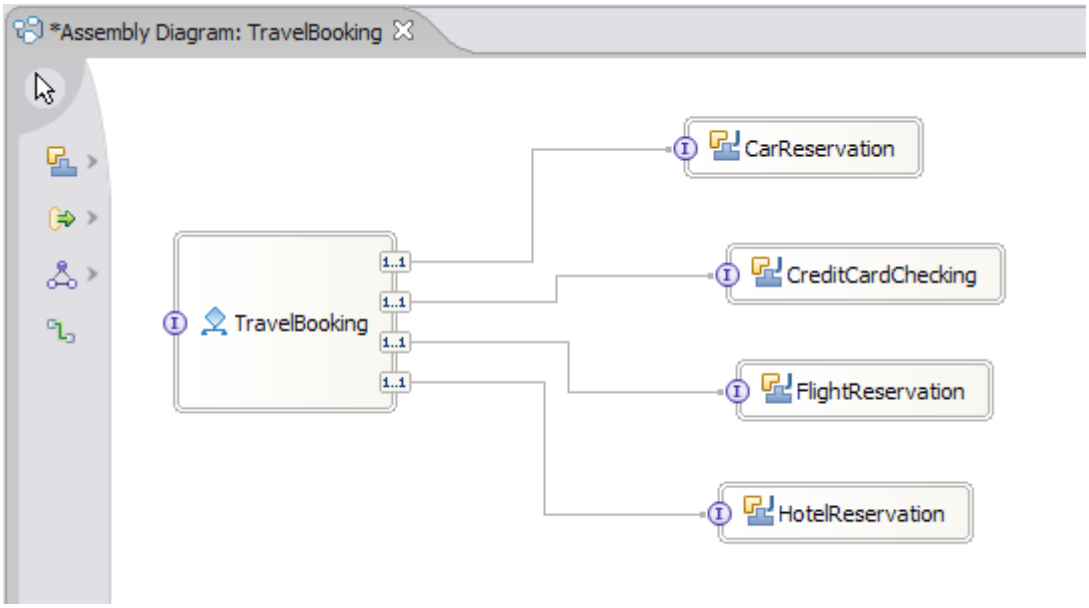


**Figure 4: Components of a travel booking application and their assembly**

BPEL process definitions contain *partner links* which represent peer-to-peer relationships with other partners the process interacts with, either one-way or two-way. Before an outbound interaction can take place, a partner link must be associated with a concrete service endpoint. From the SCA perspective, each outbound relationship (the "partner role" side of a partner link) is represented by a reference that must be wired to another service component or import. Likewise, each inbound relationship (the "my role" side of a partner link) is represented by an interface that other component's references can be wired to. The wiring of a process component binds a BPEL partner link to the concrete endpoint of the service offered by another component, which may be another module-local component or a Web service.

Within a BPEL process definition, each side's responsibility of a partner link is described by WSDL port type. However, this does not imply a particular Web service protocol must be used for the actual service invocation. Services that are wired together inside of an SCA module can call each other directly using synchronous or asynchronous calls. Services outside of an SCA module can use other protocols including Web services (see next section).

Components with implementation type "TaskImplementation" represent human tasks. Participating tasks (see section 4.1 on page 24) have SCA interfaces only and no references. Conversely, originating tasks have only SCA references. Interfaces and references of task components can be wired together with other components in the same way as for every other component implementation.

### 2.1.2 Imports and Exports

Imports and exports represent the external access points from and to an module. Imports allow consuming external services that are not part of the same module. These imported services can be accessed by clients inside of the module like any other component.

Each import contains a binding section that specifies the concrete protocol used to communicate with the external service, which may be a service exported by another SCA module, a Web service, or a service invoked using a J2EE connector or a JMS message.

Exports are used to publish services provided by a module, so that they are addressable outside the boundaries of a module. The service published from the module can be a service of a component defined in the module, or an import defined in the module. The latter case allows the republication of an external service with a new address and/or new bindings. An export publishes services provided by the module.

Similar to imports, each export has a binding specification that determines how the service can be used by external callers. For example, a Web service export binding causes the exported component to be deployed and exposed as a Web service and an SCA export binding causes the exported component to be callable from other SCA modules or SCA clients.

### 2.1.3 SCA and BPEL – Complementary Technologies

Both BPEL and SCA are service composition models, described in a formal language based on XML. They describe business services implemented by composing other business services. Finally, both describe inbound and outbound service interactions using WSDL port types. However, the two languages serve different purposes, and it is important to understand how they complement each other.

SCA describes the structure of an application. The elements of an SCA assembly model include components, services offered by these components and service references components depend on, and connections between components. Moreover, SCA describes endpoint addresses and communication methods used for the connections and policies applied to components and to the connections between them.

On the other hand, a BPEL process describes the business logic, that is, sequences of operations which are performed to execute an individual business process. Services are provided and consumed through partnerLinks, that is, abstract interfaces that must be connected to actual endpoints and communication methods through configuration.

## 2.2 Business Objects

Business Objects represent business entities that flow between components within the WebSphere Process Server runtime. Business objects are based on Service Data Objects [SDO]. The key concept introduced by SDO is the data object. A data object provides a unified way to access data. 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.

On top of the SDO DataObject concept, Business Objects provide services such as create, copy, equality, and serialization. Business objects are described using XML schema complex types and elements defined with a complex type. Simple-typed data is directly represented by a corresponding Java type – the mapping between XML schema simple types and the Java base type is defined by the SDO specification.

# 3 Business Process Concepts

This section introduces the main concepts of business processes as supported by Business Process Choreographer in WebSphere Process Server V6.1.

## 3.1 Two-Level Programming Model

In a service-oriented architecture, services may be either simple, monolithic applications or composite services implemented as business processes. Processes define the order of service interactions and provide business logic between such interactions.

In the two-level programming paradigm, applications consist of process models and service implementations. *Programming in the small* is the implementation of elemental service implementations in a traditional programming language. *Programming in the large* is the development of business processes, using graphical tools for flexible service composition, without the need for directly dealing with programming languages directly. The model allows for recursive aggregation at different levels of abstraction, i.e., a composite service at one level can be used as an atomic service at a higher level of abstraction.

## 3.2 Business Processes

A business process orchestrates a set of services to fulfill a particular business goal. For instance, the processing of a purchase order can be defined as a business process that specifies all the steps involved in dealing with a specific order. Business processes may be short-running or long-running. They may involve services of all kinds such as human tasks or fully automated services.

Business processes allow aggregating services into new services as each business process itself is yet another service. That is, a business process acts as service requestor as well as service provider. Looking at a purchase order process, it may offer services to submit and cancel an order. On the other hand, to fulfill its processing, the process needs to interact with another service provided by a supplier.

The interfaces of a business process are specified using Web Services Definition Language (WSDL). A business process has inbound interfaces and outbound interfaces: Inbound interfaces define what services the business process offers, outbound interfaces the services it requires to be provided by other services providers. Through these interfaces a business process interacts with the outside world.

You define the implementation of a business process, its business logic, using the *Web Services Business Process Execution Language* (WS-BPEL). WS-BPEL provides a standardized way of defining business processes, based on WSDL and other XML standards like XML Schema and XPath. Moreover, WS-BPEL is an extensible language. Process Choreographer extends WS-BPEL through:

Staff extensions to allow for human interaction specifications

Java extensions to support Java to express parts of your process logic

Extended flow support enabling arbitrary cycles through back links

Extension allowing for integrating with IBM Information Server

Support for queries based on particular properties of a business process

Quality of service extensions

XPath extensions functions allowing to deal with business objects, or access custom properties

The next section introduces the overall structure of a business process.

### 3.2.1 Overall business process structure

Let's look at the graphical representation of business processes in WebSphere Integration Developer. It depicts the main ingredients that make up a business process definition.

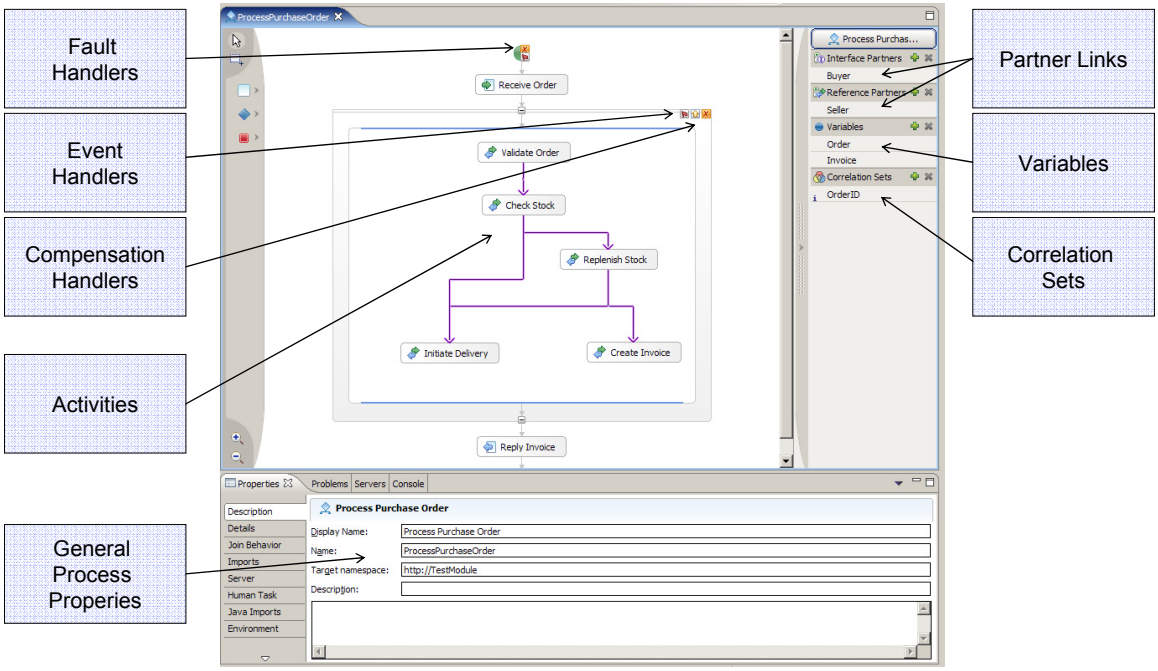


**Figure 5: Overall structure of a business process**

*General process properties* define attributes of the process such as its name or target namespace. As part of those properties, you also define the type of process that is whether it is a long-running process or a short-running microflow (see chapter "Microflows and Long-running Processes" for more details).

*Partner Links* define the interfaces through which the process interacts with external services. They also specify the role the process takes in such service interactions, whether the process acts as service provider, as service requestor, or in both roles.

*Variables* are the construct in WS-BPEL to store data – data that the process exchanges during service interactions as well as data that is needed for internal processing.

*Correlation Sets* allow for identifying a particular process instance through a set of properties passed as part of the message data when interacting with the process instance. This enables multiple interactions with the same process instance.

You may specify different types of handlers at various levels of a process definition:

*Fault handlers* deal with exceptional conditions. They can be defined to deal with specific faults or to handle of arbitrary faults. Typical uses of fault handlers include notifying other services about unexpected behavior, creating log entries, or reversing work by invoking compensation handlers.

*Compensation handlers* are used to undo already successfully completed work upon the occurrence of faults in the process.

*Event handlers* are used to deal with unsolicited external events or timer events in parallel to the overall process execution.

Finally, there are *activities*. They define the actual implementation of a business process, its business logic. The following section discusses activities in more detail.

## 3.3 Activities

Activities are the composition primitives that implement a business process. WS-BPEL distinguishes *basic activities* and *structured activities*. Structured activities allow specifying collections of nested activities, and their execution order. Basic activities represent atomic operations within a business process.

Looking at *basic activities*, a basic activity can be one of the following:

*Receive* – waits to receive an external request

*Reply* – provides a response to a request received via a *Receive* activity

*Invoke* – calls an operation of another service

*Human Task* – invokes a service provided by a human being

*Assign* – manipulates data of variables and partner links

*Java Snippet* – allows inline Java for simple computations and data manipulation

*Throw* – throws a fault from within the process

*Compensate* – triggers compensation for a successfully completed process logic

*Wait* – waits for a period of time or until a specified point in time

*Terminate* – forces the termination of the entire process

*Empty* – provides a no-op instruction

*Information service* – interacts with services provided by the Information Server platform, or directly accesses a relational database using SQLBasic activities can be aggregated into more complex structures using *structured activities*. In the following, you see the structured activities supported in V6.1:

*Sequence* – groups activities that are executed sequentially

*Parallel* (or *Flow*) – specifies a collection of activities that may execute in parallel. You may use links to define the execution order of those activities including the synchronization of parallel paths of that flow.

*Scope* – defines a collection of activities with their own set of variables, as well as fault handlers, a compensation handler, and event handlers.

*While* – specifies a loop

*Choice* (or *Switch*) – selects one out of several branches

*Receive Choice* (or *Pick*) – executes one out of several branches as soon as a matching request arrives, or upon timeout

*ForEach* – allows processing a dynamic number of multiple branches – in parallel or serially.

*Cyclic Flow* – enables flow-oriented modeling including arbitrary cycles (aka back links).

You can compose structured activities recursively to express arbitrarily complex aggregates of activities.

## 3.4 Microflows and Long-running Processes

Business Process Choreographer supports two types of business processes: *long-running processes* and *microflows*. Each type has its own characteristics with regard to its quality of service and transactional characteristics. The type of a process determines what modeling constructs the process is able to support, as well as how the process is being executed at runtime. The combination of long-running processes and microflows allows for building powerful business process compositions. Let's look at the differences of the two in some more detail.

A *long-running process* may run for hours, days or even years. It is of interruptible nature thus allowing for asynchronous interactions with other services, such as waiting for unsolicited events, involving human interactions, or realizing complex business-to-business interaction with other stateful services. Such a process is realized as a series of global transactions that may be executed in parallel to support truly parallel branches within the process. Its state is stored in a database.

A *microflow*, on the other hand, is of short-running, non-interruptible nature. It is transient, that is, its state only exists in memory. A microflow is executed within a single unit of work (either a global transaction or an activity session). It typically orchestrates synchronous, short-running services; in particular it cannot drive truly asynchronous service interactions or human tasks.

### 3.4.1 Transactional characteristics of long-running processes and microflows

As stated before, one of the main differences between long-running processes and microflows is their transactional characteristics:

A *microflow* is executed within a single unit of work. The unit of work may either be a global transaction, or an activity session (in the following, we focus on the first).

A *long-running process* comprises multiple global transactions. Each navigation step of a long-running process is performed in its own transaction. A navigation step may span multiple activities.Invoked services may or may not participate in the transaction of the process. Services themselves can be transactional or not.

The following figure depicts the transaction of a *microflow*, and services that participate and do not participate in the microflow’s transaction.

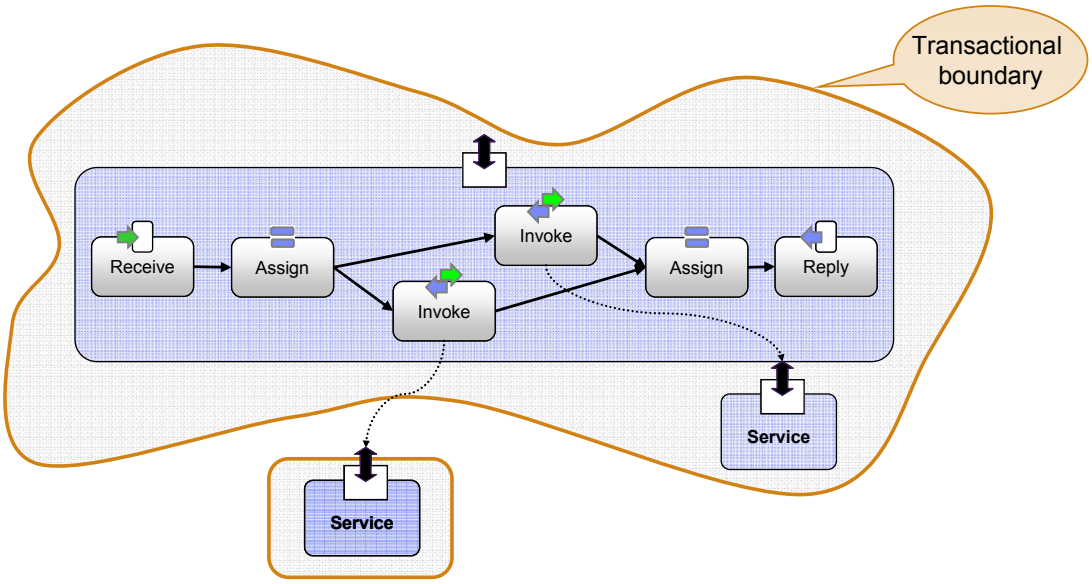


**Figure 6: Microflow and its transactional boundary**

If a transaction rollback occurs, the transaction manager rolls back all the services that participated in the microflow's transaction. Others remain untouched. For undoing those too, compensation needs to be modeled (see chapter "Compensation" for details).

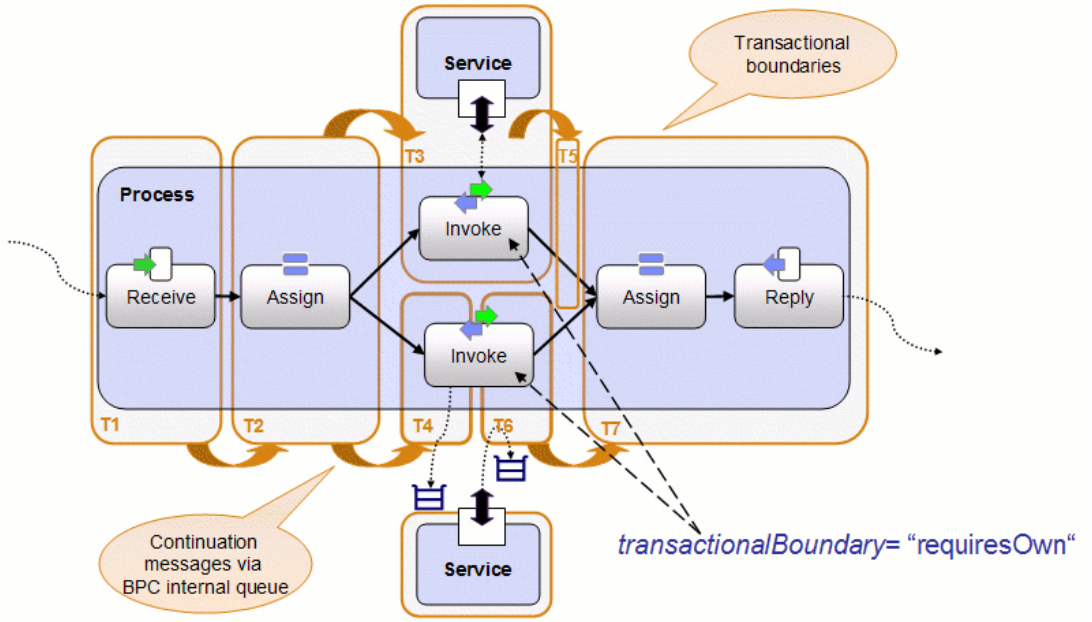


**Figure 7: Long-running process and its transactional boundaries**

Looking at a *long-running process*, its navigation is performed as a series of transactions. Each single transaction is triggered either by an external request, an internal continuation message, a timer or a response from an asynchronous service. The transaction's execution comprises the navigation of one or more activities of the process including the respective process state changes in the state database. The last step of such a transaction may be the sending of a continuation message to ensure follow-on navigation of the process.

Long-running processes are forward recoverable. If there is a failure, the transaction that was in flight rolls back and is retried, causing navigation to continue from there. Navigation state is kept in the runtime database in form of process instance state. In addition, the messaging system holds navigation state in form of external requests, responses, and internal continuation messages to drive the navigation of the process. The database, the messaging system, and transactional resources used by invoked services all participate in a two-phase-commit protocol.

You may influence the transactional boundaries of a long-running process using the so-called *transactional behavior* attribute. It can be specified for invoke activities, human task activities, information service activities and Java snippet activities as follows:

The transaction should *commit before* your activity is started.

The transaction should *commit after* the activity is completed.

The activity should *participate* in the transaction of the previous navigation.

The activity *requires* its *own* transaction.

Thus, you can tune your process with regard to the number of global transactions involved in the overall process navigation. The process engine tries to honor those specifications, however, may need to overrule them where necessary.

## 3.5 Compensation

Applications typically require transactional integrity to guarantee that a complex request is executed either in its entirety, or not at all. For traditional transactions, this is described by the ACID properties (Atomicity, Consistency, Isolation, and Durability). They are achieved by a transaction manager, resource managers, and back-end systems working together according to a two-phase commit protocol. This cooperation ensures that the operations performed on behalf of a transaction are either all committed or all rolled back.

Often, however, a complex request cannot be run as an ACID transaction for a number of reasons:

Back-end systems or resource managers might not be able to participate in the XA protocol. Updates to these systems are performed immediately and do not participate in the overall transaction. If the transaction fails, these updates are not undone; the overall system is in an inconsistent state.

As long as a transaction has not been committed, none of the changes done on its behalf are visible to the outside world. The isolation property guarantees that they only become visible when the transaction's final state has been reached. This works well for short-running processes that invoke synchronous operations. However, when a process involves asynchronous steps, for example, steps implemented by a back-end system driven by JMS messages or steps involving human interaction, intermediate results of the process must be made visible. This means that the JMS message must be sent, the information about the task the person has to work on must be made available, and so on. Therefore, the intermediate state of the process has to be committed.

There are actions in a business process that are inherently non-transactional. Sending a letter to a customer is such an operation – as soon as the letter has left the sender, there is no way to undo the operation. In all of these examples, intermediate results of the business process are made available, and they cannot be undone by simply rolling back an ACID transaction with the help of the transaction manager. Instead, another operation must be invoked that explicitly reverses the original operation: If a back-end system has been called to update data, for example, to increment a value, the system must be called again to decrement the value. If a letter has been sent in error to a customer, another letter must be sent to apologize for the error.

These operations might need to be cascaded if other operations have already started that use the results that have been wrongly made available. To help you defining undo logic, you may use *compensation* to appropriately handle failures situations of your process. WS-BPEL introduces the notion of compensation that lets you define compensation logic as part of the process logic through the following constructs:

*Compensation handlers* allow specifying compensation logic to undo successfully completed activities in the case of a fault. Compensation handlers may be defined for scopes, i.e., groups of activities, and for single activities that invoke services.

*Compensate* activities trigger such compensation handlers. This construct can be used as part of your fault handling logic in fault handlers and in compensation handlers.

By default, if you do not specify fault and compensation handlers explicitly, compensation will still be triggered for you by applying a default implementation that is calling compensation handlers in the reverse order of the invocation of their respective activities.

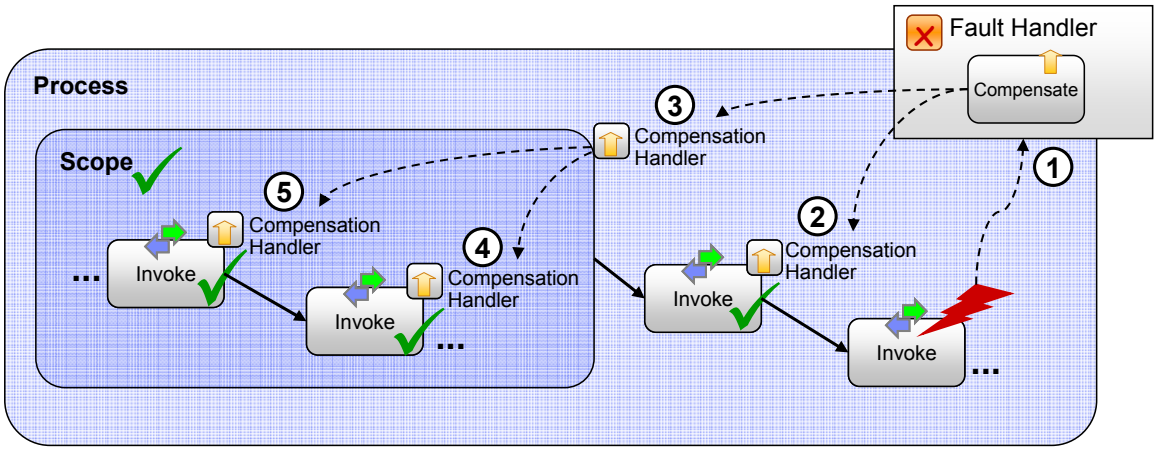


**Figure 8: Compensation of a BPEL process**

Let's now visualize how compensation works looking at the following schematic example: The sample process is composed of various activities with compensation logic specified as part of their compensation handlers. The overall fault handler of the process is modeled to trigger compensation in order to reverse the changes caused by the process in the case a fault occurs.

What does this look like at runtime? After successfully processing the scope's activities, the scope itself completes successfully. Eventually, a fault occurs that triggers the process-level fault handler (1). Its implementation, a *compensate* activity, causes compensation to kick in. Compensation is to ensure that all successfully completed work prior to the occurrence of the fault is undone in the reverse order of its execution. Compensation starts with calling the compensation handlers of the directly enclosed, successfully completed activities of the process – the invoke activity (2) and the scope (3). The scope has its own compensation logic provided as part of its compensation handler. It in turn triggers compensation of the activities it comprises – (4) and (5). Thus, for each of the successfully completed service invocations, the defined compensation handlers are triggered in the right order, resulting in undoing the visible effects of the process.

The example depicted a scenario that undoes the entire process. BPEL compensation also allows you to partially reverse results of particular units of the process and continue with normal processing after that. That is, compensation can happen on and off while a process is being navigated – whenever there is a need to reverse certain logical units of the business process.

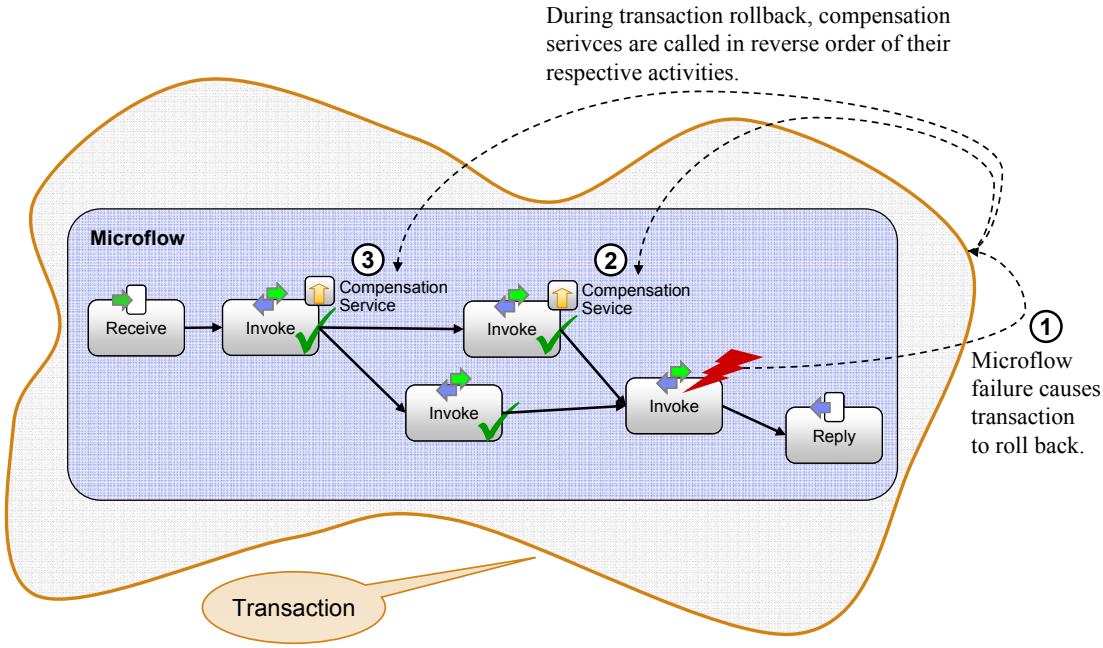


**Figure 9: Microflow Compensation**

The compensation approach described so far applies to long-running business processes. For microflows (shown in the next figure), compensation differs in some regard due to their difference in transactional behavior. A microflow is performed within a single unit of work, i.e., transaction or activity session. Consequently, compensation is triggered upon rollback of that unit of work, that is by the infrastructure rather than a compensate activity (1). Compensation logic is typically specified for those activities that cannot be reversed by means of rolling back the unit of work, for instance a non-transactional web service. During the execution of the microflow, the compensation logic for compensable activities is registered with the enclosing unit of work. Dependent on the outcome of this unit of work – rollback or commit – compensation might happen or not. You specify compensation logic in form of single compensation services for invoke activities only (2).

Microflows are considered atomic; there is no support for "partial rollback" – it is either all or nothing. Consequently, scopes within a microflow cannot be compensated during navigation of a microflow.

## 3.6 Event Handling

WS-BPEL processes can deal with events that occur concurrently to the execution of the business process. *Event handlers* are provided to receive such unsolicited events; they support two types of events – message events and timer events.

*Event handlers for message events* implement operations provided by the business process. On arrival of a corresponding request message, the event handler is executed. If the event handler has a reply activity to respond to the inbound request then it implements a request-response operation.

*Event handlers for timer events* are executed when a certain point in time has been reached or a specified time interval has expired. They can be executed repeatedly; in this case, each timer event occurs when specified duration has passed since the previous event handler was started.

An event handler is associated with a scope (including the business process itself as the outermost scope). It is enabled as part of the initialization of the scope. When the scope has finished execution, its event handlers are disabled. Running event handlers are allowed to complete before the scope ends.

When multiple message or timer events occur, the corresponding event handler instances run in parallel to the scope and in parallel with each other. Variables defined within an event handler are private to the event handler instance. Variables defined outside the scope the event handler is declared in may be accessed.

Event handlers cannot be used to create a new process instance. Therefore, they always represent inbound requests into a running process instance, and require the use of correlation sets (see below) in order to ensure the delivery of the request to the correct process instance.

Unlike its predecessor specifications, WS-BPEL 2.0 only allows scopes as primary activity of event handlers. Even though not enforced in WPS V6, it is recommended to comply with this rule right away when modeling new processes.

## 3.7 Lifecycle

The business process definition specified in WS-BPEL serves as a template from which process instances are created and started during runtime. To allow doing so, the business process definition and related authoring artifacts are deployed and installed as part of an enterprise application. After installation of that application, a so-called *process template* is available to the runtime. Process templates are used to create and start *process instances* from.

As part of the BPEL process definition, you specified operations of the process' interface that create instances of that process. When invoking one of these operations, an instance comes to life. Invocation such operations may happen via different APIs offered by WebSphere Process Server, for example, using the Service Component Architecture (SCA) client interface, or using the Business Process Choreographer API.

Once a process instance has come into existence, the process engine navigates this instance as prescribed by the corresponding process template. That is, the process engine determines the activities of the business process it performs, and the order of their execution. An instance ends upon completion of the business logic that implements the process, or upon termination of the instance. Termination of a process can be initiated through an unhandled fault, upon a terminate activity as part of the process logic, or using the Business Process Choreographer API that offers methods to terminate a process instance.

While a process instance is running, you can put its navigation on hold temporarily, for example, to repair or maintain your system. The Business Process Choreographer API offers a suspend operation to do so. Resumption of the suspended process instance may happen automatically at a certain point in time or after a specified duration. Alternatively, the resume operation allows you to continue the navigation of a process instance from the point in execution where it was suspended before.

Upon completion of a processes instance, there is also a means to restart the process instance in order to rerun the instance based on data it originally started with. This is particularly useful when trying to recover from a failure during the prior execution of the process instance, that is, when its completion was unsuccessful or forced. The restart operation is part of the life-cycle operations the Business Process Choreographer API offers. Finally, the life-cycle of a process instance concludes with the deletion of the process instance that may happen automatically or be triggered explicitly using the Business Process Choreographer API.

### 3.7.1 Subprocesses

For modularity or reuse, you may not only want to model a single business process to contain all your business logic, but you may rather want to define multiple business processes that invoke each other. A process that is invoked by another process is known as a *subprocess*. Business Process Choreographer distinguishes *child processes*, and autonomous subprocesses called *peer processes*.

For child processes, the parent process manages their life-cycles. That is, life-cycle operations applied to the parent process are propagated to its child processes. Those life-cycle operations comprise suspend, resume, terminate, delete, and compensation.

For peer processes, the parent process does not manage the life cycle. In particular, the parent process may end before a peer process it spawned.

### 3.7.2 Process instance states and transitions

Life-cycle operations result in state changes of the process instance. For long-running processes, those states become visible. The following state diagram depicts the states and the respective state transitions that a process instance might undergo during its life-cycle. Some of the transitions are triggered explicitly using the afore-mentioned APIs, others are driven by the process engine, e. g., in case of child processes (see also section above).

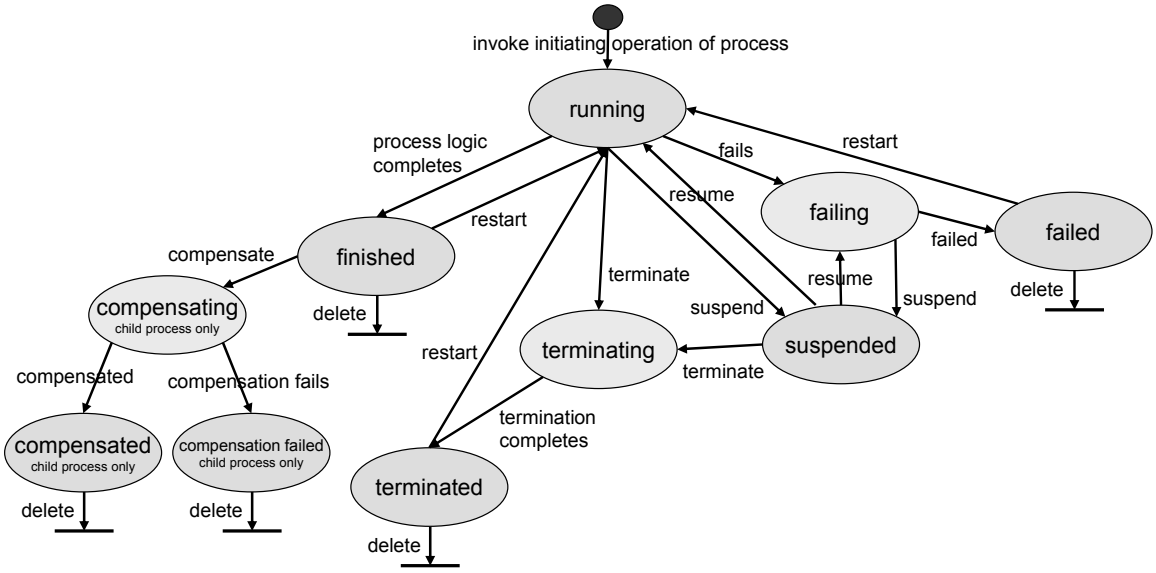


**Figure 10: Process instance states and transitions**

Having introduced process instances, let's now, as part of the next section, look at how to identify single process instances.

### 3.7.3 Correlation Sets – Identifying Long-running Instances

There might not only be a single interaction involved when dealing with a particular business process instance, but multiple interactions with the same process instance. For example, you want to cancel the process instance you created, which handles a specific purchase order. This requires a means to identify that instance. WS-BPEL introduces the notion of *correlation sets* for that purpose.

A correlation set is composed of a set of properties passed as part of a message exchanged with the process instance through one of its service interfaces (such as order ID and customer name). The process instance stores correlation sets as part of its state. Once exchanged and thus initialized, you may use such a correlation set as part of subsequent interaction with the same instance to uniquely identify that process instance. A single process instance may have multiple correlation sets, for example, may define different identifications for different service interactions.

Correlation sets are a means to identify process instances as part of the application data that is exchanged with a process instance. In addition to this user defined way of dealing with instance identification, Business Process Choreographer also associates a unique process instance identifier with each process instance. The system generated identifier can be used as a handle when operating through the Business Process Choreographer API.

## 3.8 Dynamic Partner Assignments

The services with which a business process interacts are modeled as partner links in BPEL. Before operations on a partner's service can be invoked via a partner link, the binding and communication data for the partner service must be available. The relevant information about a partner service is usually set as part of business process deployment.

As an alternative to the static association of partner links with services endpoints, BPEL allows establishing the relationship dynamically and provides the mechanisms to do so via assignment of endpoint references (EPRs; see [WS-Addressing]) to partner links.

If a BPEL process definition contains assignment activities that initialize partner links with EPRs before the partner link is used for outbound interactions, then the SCA reference associated with the partner link (see chapter “Service Component Architecture” on page 8) does not need to be statically wired to another service. It is not always possible to detect that a partner link is used before it is initialized; therefore, a warning message is issued during validation of the SCA module when references are left unwired.

Another dynamic endpoint resolution scenario is the invocation of a subprocess. When modeling the invocation of a subprocess, one may only know the name of the subprocess. The concrete version of the subprocess is determined dynamically using its name.

## 3.9 Monitoring

The state changes that happen on behalf of the execution of a business process can be observed by other applications using the monitoring capabilities of the process engine. The process engine reports these state changes in form of Common Base Events (CBEs) that are emitted using the Common Event Infrastructure (CEI), or as entries in a database table, the so-called database audit log.

### 3.9.1 Defining Monitoring for a business process

Business processes externalize state changes by default for certain elements like, for instance, the process itself and invoke activities. Furthermore, a fine grained monitoring specification can be provided that contains specifications about what element of the business process definition is supposed to externalize which type of a state change. The monitoring specification can be defined and manipulated using the Monitor pane of BPEL constructs when designing the business process with WID. For example, it can be specified that the start of an invoke activity shall be externalized as a record in the audit log database table, while the completion of the process is supposed to be emitted as a Common Base Event.

### 3.9.2 Using Monitoring information

CEI provides two ways to consume records based on its API. The first kind of API calls allows using XPath based queries to retrieve events of the BPEL process execution. The second option is to subscribe to certain event types and to use topics. Thus it is possible to have multiple subscribers that get informed automatically when state changes occur.

CBE Event Browser, BPC Observer and WebSphere Business Monitor are three applications that use the generated CBEs. CBE Event Browser is used to view all the events occurring in WPS. BPC Observer allows observing aggregated information of business processes in BPC, e. g. the average execution time. With WebSphere Business Monitor you can monitor your business processes track their key performance indicators.

If the database audit log was selected to externalize state changes, then the AUDIT_LOG view in the BPC database can be used to access the data. That means that SQL SELECT statements can be used to retrieve the records. The audit log table is an “append only” table. Thus, it will grow over time unless records are explicitly deleted from that table.

# 4 Human Task Concepts

## 4.1 Human Tasks

A human task is a component that involves a person interacting with a service. Human tasks can be of different kinds, depending on the kind of interaction they are used for. The most common scenario is to use human tasks to involve people in service-based applications to perform a service. Such human tasks are called *to-do tasks* as they represent a to-do for people participating in service-based applications.

An application invokes the human-based service and gets a response once the person working on the task has completed his work. The business process scenario where a human task activity is used to include a human step in a process is a typical form of this scenario.

As to-do tasks are just another service, replacing them with an automated service does not require changing the calling application. For example, in case of a business process, this allows to start with human based service realizations, and increase the degree of automation by subsequently replacing human tasks with automated services. The figure below shows the wiring diagram of a translation process that uses one service to translate a document and one service to verify that the translation has been done correctly. The process is wired to use human tasks for both, the initial translation and the verification.

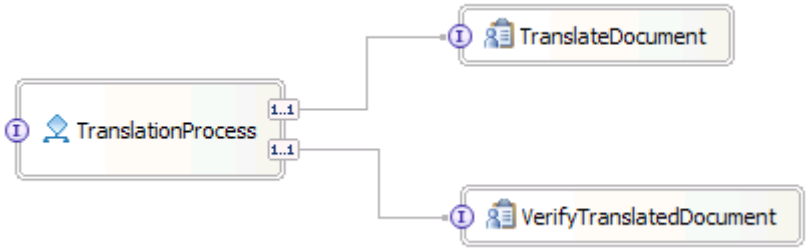


**Figure 11: Human task components**

Assuming that a service provider starts to offer a cheap and powerful automatic translation service, the application could be changed by simple rewiring: Changing the wire between *TranslationProcess* and the human task component *TranslateDocument* to now point to the new *AutomaticTranslationService* component is all that is required. The picture below shows the modified wiring diagram.

Besides the scenario where tasks are used to involve humans to perform services, there are scenarios where tasks provide a way for humans to invoke a certain automatic operation, or to schedule work for another human. Tasks that allow humans to invoke an automatic operation are called *invocation tasks*. The person who creates and starts the task becomes the originator of the work that has been initiated. This person will also receive the result once the task completes, that is, when the service that provides the implementation of this task returns its result.

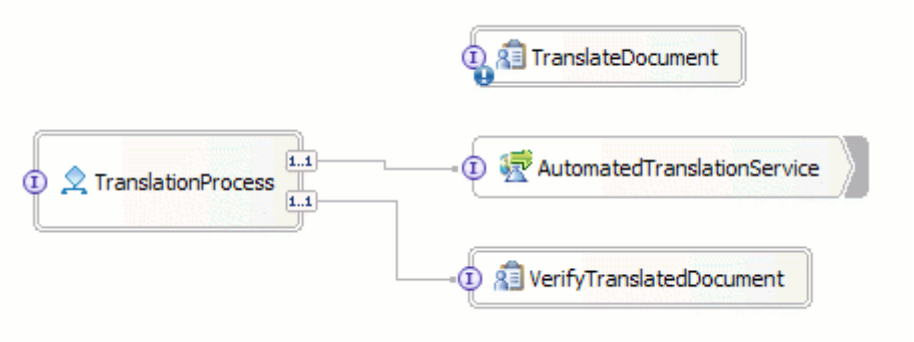


**Figure 12: Human task component replaced with automated service**

Tasks that allow humans to schedule work for other people are called *collaboration tasks* as they allow people to collaborate. The receiver of the work performs the job defined by the task in the same way the person would have done if the task would have been created by an automated service. Conceptually, collaboration tasks can be seen as a combination of an invocation task that describes the interface of the task and the set of people that can start the task, and a to-do task that defines the potential owners of the task.

The figure below summarizes the different kinds of human tasks and their interfaces.

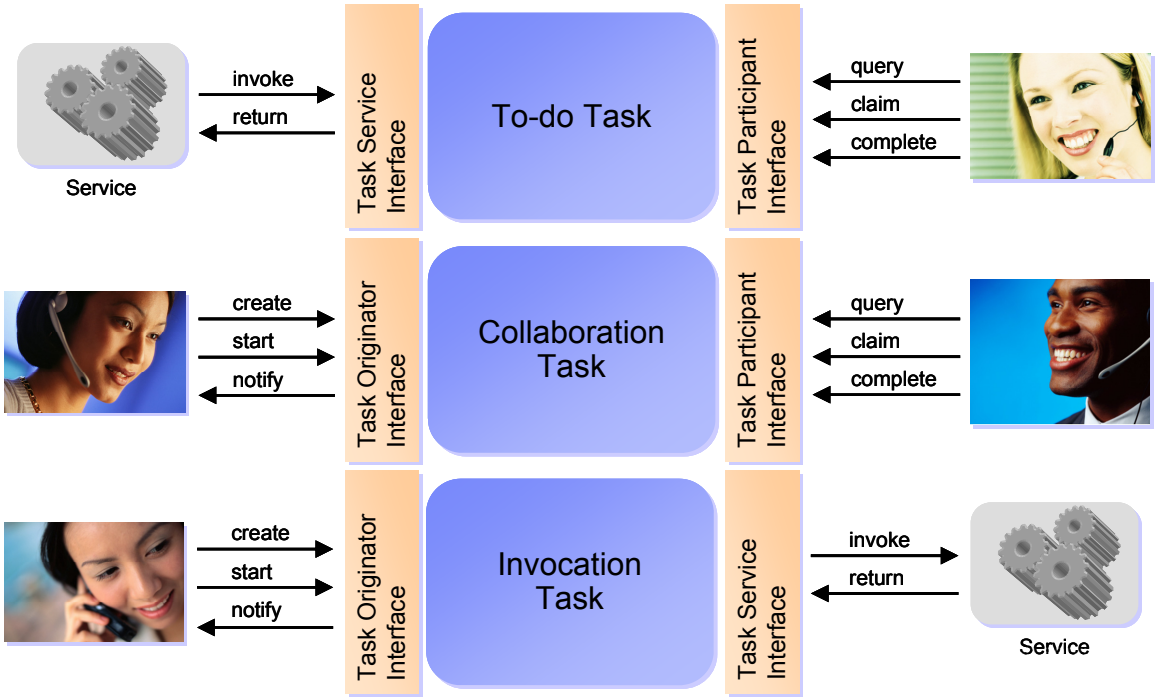


**Figure 13: The different kinds of human tasks**

## 4.2 Assigning People to Tasks

People work on tasks according to the role the task defines for them. Examples for such roles are potential owner, reader, or administrator. People assignment can either assign individual people or groups. The latter is preferable when using with large groups of people, especially in high volume scenarios.

The criteria to find people to act on a task in a certain role are specified by defining a *people query*, formerly also known as staff query. People queries are executed at runtime retrieving the set of qualifying users from a certain people directory. Supported people directories are Virtual Member Manager (VMM), WebSphere User Registry and native LDAP. As VMM provides a uniform interface across one or many underlying people repositories incl. LDAP using VMM is recommended. Some features like substitution (see section 4.4) can only be used when VMM is used as the people directory.

The result of people queries can be altered at runtime. For example, the administrator of a task can decide to transfer the task to a certain person, or to add additional people to the list of potential owners to make more people eligible for working on a task.

## 4.3 Escalation and Notification

Human tasks can have certain time constraints. The most common time constraint is the due date of a task. If the task has not been completed until its due date, it becomes overdue, requiring special attention. To cater for these scenarios, human tasks allow for the specification of a due date. The due date can be used as a sort criteria when visualizing lists of human tasks. In addition to that *escalations* are offered. Escalations allow the modeler of a human task to express timing expectations for a task. A typical usage scenario for an escalation would be to monitor how long a task remains unclaimed, and notify somebody when a certain time period is exceeded. Another typical usage scenario would be to track the time it takes for a person to complete a task, and to take appropriate action if a certain duration is exceeded.

Escalations can be chained. The first escalation in an escalation chain is activated when the task reaches the activation state of the chain. Once activated, the escalation waits until its modeled escalation time is reached. When that's the case, and in the meantime the task has not progressed to the expected state, a certain escalation action is triggered. Escalation actions are used to inform the escalation receivers that the progress of the task is behind. Supported escalation actions are giving the escalation receivers access to the escalated task by creating a work item for them, sending them an e-mail, or sending an event to a registered event listener. The figure below shows an example of an approval task using escalation.

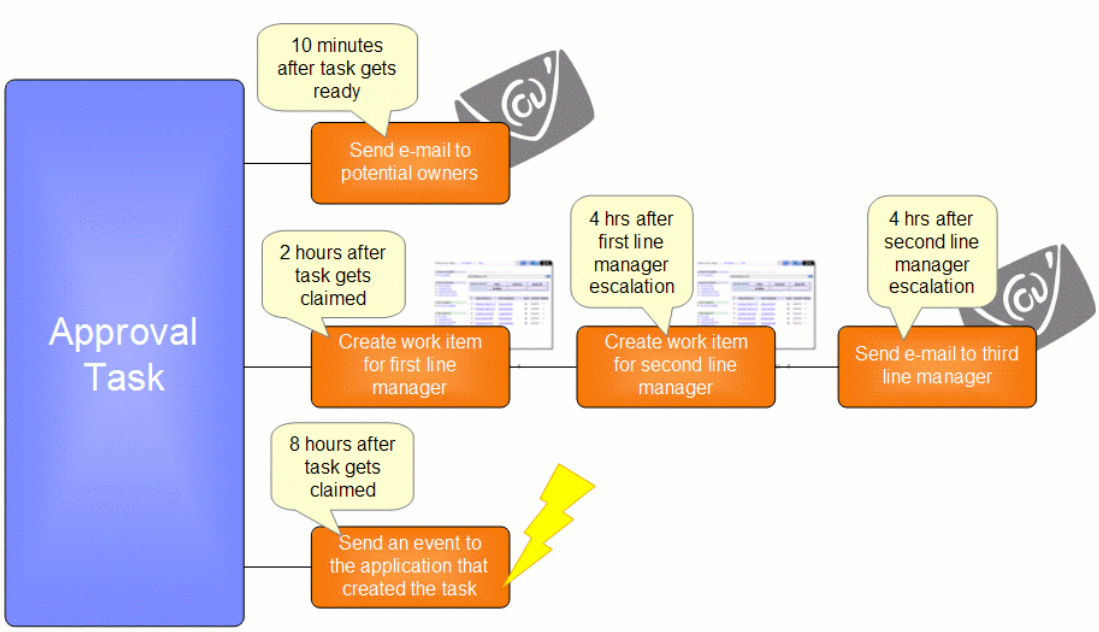


**Figure 14: Various escalations of a human task**

The approval task has three escalation chains. The first escalation chain starts when the task gets ready. It only has one escalation, which sends an e-mail to the potential owners of the task if the task has not been claimed within 10 minutes after it has been started.

The second and the third escalation chain are started in parallel when the task gets claimed. The second escalation chain informs the managers responsible for approvals if the person working on the task requires more time than expected. The first escalation goes to the first line manager of the employee that has claimed the task, if nothing happens the second line manager is notified, and finally an e-mail is sent to the third line manager.

The third escalation chain allows automatic handling of an escalation by registering an event handler that receives notification events. The event handler can for example pass these events to an application that automatically re-assigns another user to a task.

## 4.4 Substitution

Substitution allows to automatically route tasks to the substitutes of a person in case a person becomes absent. A list of substitutes is maintained for each person participating in a human task based application. Business users can specify a list of personal substitutes and can notify the system if they are absent or present. Business Administrators can manage lists of substitutes for business users. Substitution is performed during people resolution and is re-evaluated during the scheduled refresh of people assignments.

Substitution policies can be defined for each human task to further specify the substitution behavior. There is the option to not allow substitution, which can be used to ensure that sensitive human tasks are processed by the users originally assigned. When enabling substitution then if a person is absent, this person's work is assigned to the first substitute on this person's list of substitutes that is present. Alternatively a modeler can configure substitution for a human task to just remove absent users. In that case, if none of the users or their substitutes is present, the task is assigned to the original list of users.

## 4.5 Human Tasks and Business Processes

Human tasks can be used together with business processes in two ways. A business process can use a task component just like any other component using the task's service interface. This usage mode allows for the transparent replacement of the human task with an automatic service without the need for changing the process definition.

The second way to use human tasks is to inline them into business processes. These tasks are called *inline human tasks* or just *inline tasks*. Inline tasks offer the advantage that they have access to the context of the business process they belong to. This enables scenarios like the "4-eyes principle", also known as "separation of duties". A usage example for the 4-eyes principle is an approval flow with two approval steps, both assigned to the same group of people, but with the additional rule that the approvals need to be given by two different people. The second approval's staff query is defined such that it explicitly excludes the actual starter of the first approval. This can be done because as inline tasks, both approval tasks know about the enclosing process and its context, and hence the second task can access information from the first task.

Having the ability to access process context also allows assigning tasks directly to the process starter – something very useful for example in employee self service applications.

Another scenario where inline tasks are beneficial is the *server controlled page flow* scenario. Server controlled page flows are used if a single person has to work on subsequent steps of a business process. By default, a person works on a single human task at a time, and once completed goes back to the task list to get another one. When using server controlled page flows the person gets presented with a sequence of dialogs, without having to go back to the task list. Please note that to present the user a sequence of dialog pages client side technologies like web forms or JSF pages could be used, which have a lower footprint than server controlled page flows. The use of server controlled page flows makes sense though if you need to invoke services between two UI steps to, e.g., retrieve or update data, or if you have auditing requirements that require CEI events to be written after a UI interaction has been completed.

## 4.6 Ad-hoc Creation of Human Tasks

The nature of the different usage scenarios for human tasks demands that tasks are made available not only in a static, predefined form, but that tasks can also be created and modified dynamically at runtime. To enable these scenarios tasks provide ad-hoc support: Tasks as well as task definitions can be created programmatically on the fly using appropriate APIs.

To support ad-hoc collaboration scenarios, it is possible for the owner of a given task to create tasks for other people on the fly. A scenario where ad-hoc created tasks are used is if a person gets a task assigned, but needs to collaborate with a few other people to get the job done. In that case, he creates ad-hoc tasks for the people he wants to involve, assigning to them pieces of the original task he needs for completing his task. A concrete example is a document review process where the main reviewer assigns the review of certain chapters of the document to other reviewers, based on their expertise.

## 4.7 Ad-hoc Collaboration using Human Tasks

In addition to the ad-hoc capabilities introduced in the previous chapter human tasks offer another capability that falls into the ad-hoc space. At runtime client applications can offer people the ability to dynamically create and start human tasks, and to dynamically "wire" these human tasks with existing human tasks.

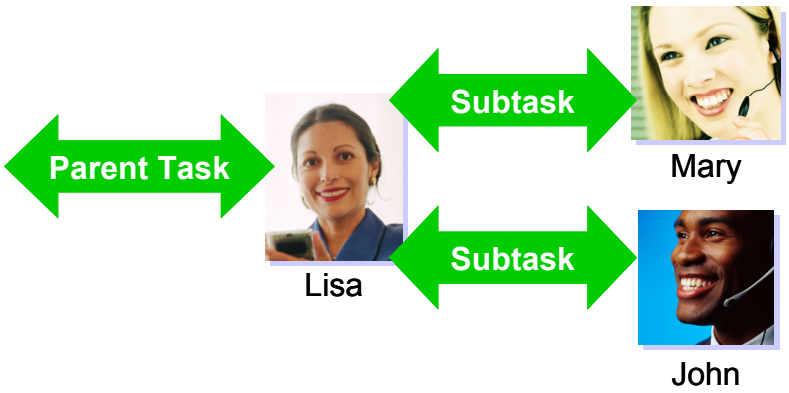


**Figure 15: Creating sub-tasks**

Figure 15 shows an example where Lisa has accepted a task assigned to her. While working on the task she realizes that she needs information from Mary and John to complete her task. Therefore she creates a human task to request information for both Mary and John as sub-tasks to her own task. By that she can easily keep track of the tasks she created. An alternative way for ad-hoc collaborations would be to get the as much work done as possible, and then to create a follow-on task for Mary, asking her to complete the work.

## 4.8 Monitoring

The state changes that happen on behalf of the execution of a task can be observed by other applications using the monitoring capabilities of the "Human Task Manager" (HTM). The HTM reports these state changes in form of Common Base Events (CBEs) that are emitted using the Common Event Infrastructure (CEI) or as entries in a database table, the so-called database audit log.

### 4.8.1 Defining Monitoring for a task

Tasks do not externalize state changes by default. However, a fine grained monitoring specification can be provided that contains specifications about what element of a task is supposed to externalize which type of a state change. This monitoring specification can be defined and manipulated using the Monitor pane of task elements when designing the task with WID. For example, it can be specified that the start of a task shall be externalized as a record in the audit log database table, while the completion of an escalation item is supposed to be emitted as a Common Base Event.

### 4.8.2 Using Monitoring information

Monitoring of human tasks is done in the same way as for business processes. The same CEI APIs can be used, see “Using Monitoring information” on page 22.

If the database audit log was selected to externalize state changes, then the HTM_AUDIT_LOG view in the BPC database can be used to access the data. That means that SQL SELECT statements can be used to retrieve the records. The audit table is an “append only” table. Thus, it will grow over time unless records are explicitly deleted from that table.

# 5 Developing Business Processes and Human Tasks

WebSphere Integration Developer (WID) is an integrated development environment for creating integrated applications containing business processes and human tasks (and other artifacts not focused on in this paper). WebSphere Integration Developer is based on Rational Application Developer (RAD), which itself is based on Eclipse.

Among other editors and tools, WID provides an assembly diagram, business process editor, human task 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 is the entity that is later on installed into the server. Libraries contain reusable artifacts like Business Objects and Interfaces. They will be included into each module that references this library.

## 5.1 Assembly Diagram

The WebSphere Integration Developer's assembly diagram lets you build applications by assembling components (see chapter “Service Component Architecture” on page 8) - it’s the graphical front end over the SCA programming model. It displays and edits the component definition SCDL (.component) files for each of the components of the application.

When you create or open a module with the assembly diagram, you can visually compose the integrated application by adding components and connecting them with wires in the editor view. Both business processes and (stand-alone) human tasks are – among others – kinds of components.

You can use drag and drop already modeled business processes and human 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 according implementations (top-down approach).

## 5.2 Business Process Editor

The process editor is a visual tool that lets you create a BPEL business process. You can add nodes to control the sequence of the execution as well as nodes that perform a particular action such as invoking services and human tasks by using the palette on the left side of the editor. You can specify definitions to handle external events, errors and compensation, the so called “handlers”. In addition, you define the data used within the business process, which can be based on a WSDL message and an XSD schema (called “Data Type”).

Both a bottom-up and a top-down approach is possible: You can either start by creating “placeholders” (that is,. “Empty Action”) as nodes in the process first and refine them, or you can use existing implementations (for example SOAP based Web Services) in your business process.

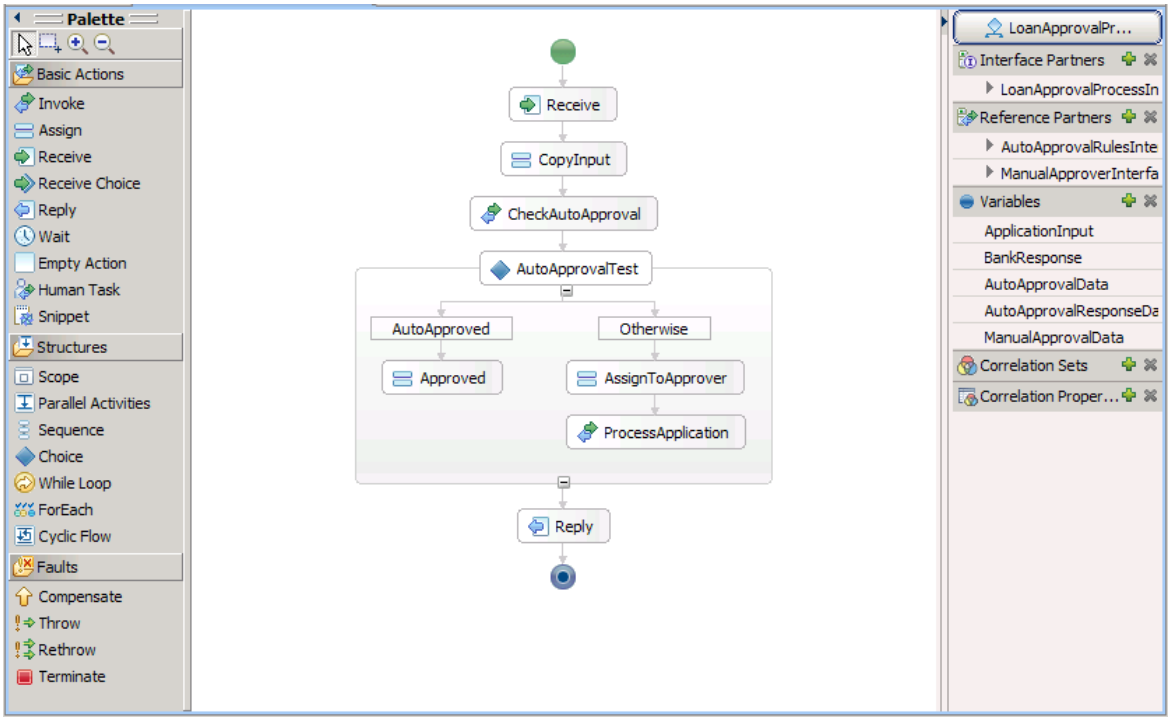


**Figure 16: Process editor**

## 5.3 Human Task Editor

The human task editor is the tool to visually compose services that involve human participation. They can be defined either within a business process (“inline tasks”) or as human task component (“standalone tasks”).

Within the editor, you define three aspects of the human tasks:

Who has access rights to these tasks (“People Assignment”) (see chapter “Assigning People to Tasks” on page 25)? For each of the predefined roles of the task like potential owner, editor, reader or administrator, you can define a staff query defining the set of persons allowed to access the task with certain rights.

How is the task visualized (“User Interface”)? This section defines how this task is presented to the user. For the Business Process Choreographer Explorer, you can define special JSP snippets to show the input and output message of the task, for Portal clients you can specify info about the portlet to be used to show the task and for IBM Lotus Forms clients you can define which form(s) visualize the task data. You can extend the list of clients by adding your own client type using a published client extension point.

What happens when the tasks takes too long (“Escalation”) (see chapter “Escalation and Notification” on page 26)? In the “Escalation” part of the editor, you can define what happens when the task takes longer than expected.

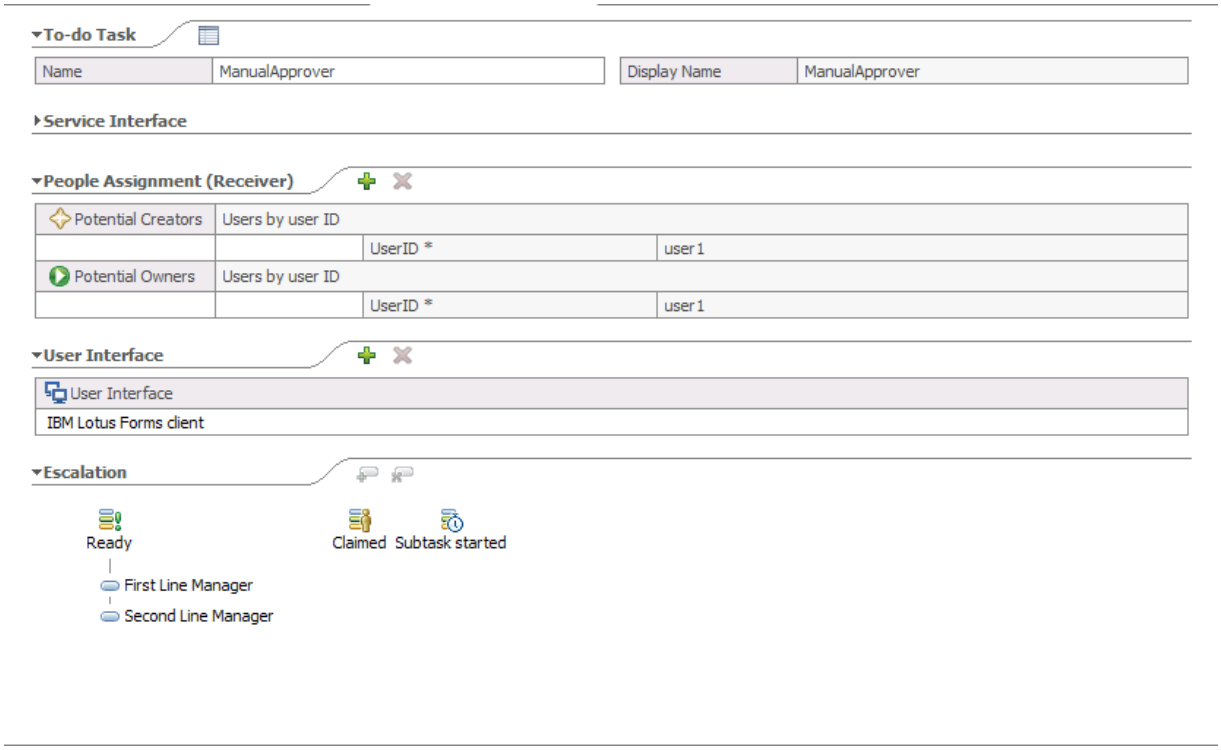


**Figure 17: Human task editor**

## 5.4 Debugging Processes

WID provides a graphical business process debugger which you can use to test and debug your business processes. It allows you to debug the control flow within the business process, view and manipulate data, and step into the code of Java snippets or Java conditional expressions of control links or loops.

The debugger requires a running WebSphere Process Server. This can be a remote server or the test environment directly integrated within WID.

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

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

## 5.5 Programming Interfaces

WebSphere Process Server V6.1 provides multiple interfaces to business processes and human tasks. From an external client perspective, you may use interfaces provided for service component architecture components (that is, strongly typed interfaces) or generic Business Process Choreographer interfaces (that is, un-typed interfaces). Figure 18 summarizes strongly-typed interfaces for business processes and human tasks and Figure 19 shows generic interfaces provided by BFM and HTM.

Additional interfaces are provided for Java code inlined within business processes (Java snippets and Java expressions), to access context of the process. Finally, a number of database views are published as part of the BPC programming model. They allow accessing the data stored in the underlying BPC database tables.

For more details about Business Process Choreographer interfaces, see the WebSphere Process Server V6.1 Business Process Choreographer Programming Model [BPC Prog Mod].

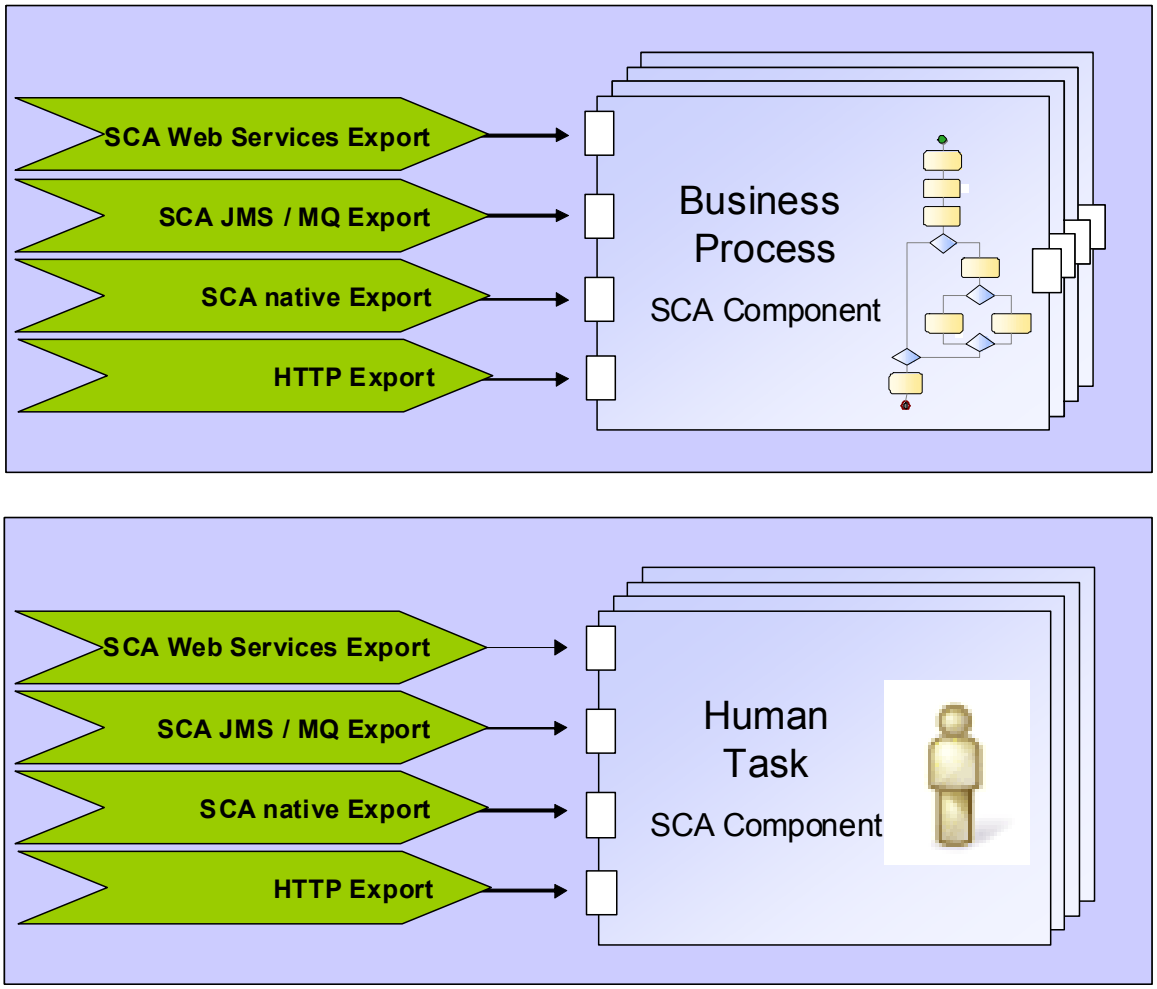


**Figure 18: Type-safe Process / Task Component Programming Interfaces**

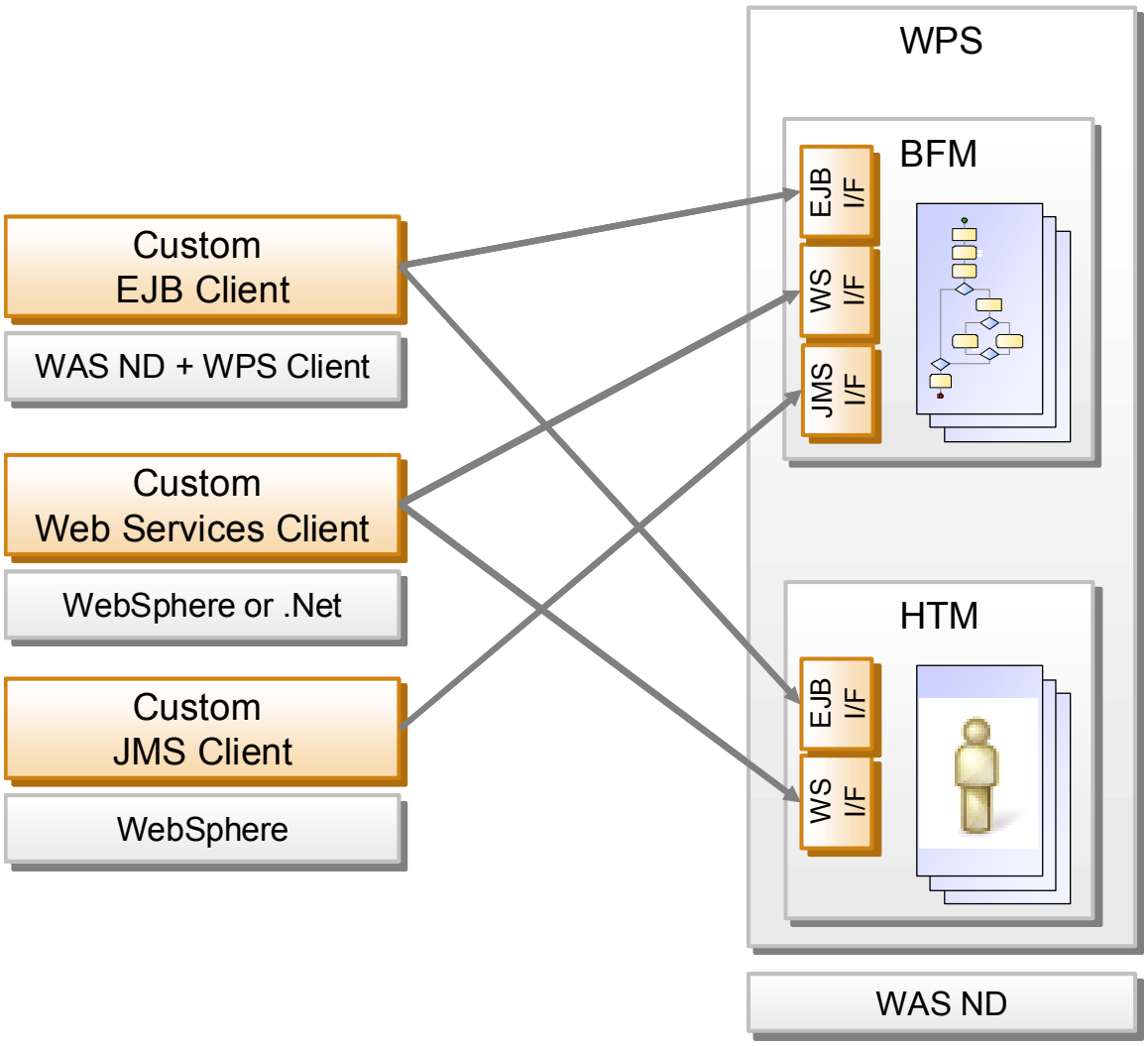


**Figure 19: Generic BFM / HTM Programming Interfaces**

# 6 Using and Administering Business Processes and Human Tasks

## 6.1 Administer Business Process Choreographer

Administering Business Process Choreographer involves different administration tasks and different tools:

The Business Process Choreographer runtime can be administered using the WebSphere administrative console or using scripts.

Business processes and human tasks are deployed and installed as part of an enterprise application. You can use the WebSphere administrative console or scripts to administer process templates and task templates, and Business Process Choreographer Explorer to work with process instances and task instances. For more information on the Business Process Choreographer Explorer, refer to section “Working with and Managing Processes and Tasks” on page 37.

Other administration tasks are not covered in this paper.

### 6.1.1 Business Process Choreographer runtime

The Business Process Choreographer runtime consists of the business process container and the human task container. It can be configured for every WebSphere application server or cluster that has Business Process Choreographer installed. The configuration can be done through wizards in the WebSphere administrative console or through scripts. During configuration, the associations between the business process container, the human task container, their database, and the different message queues they require are created. After the initial configuration step, an administrator can perform the following administration tasks:

Change the location and maximum size of the compensation recovery log.

Change the settings for the human task manager E-Mail service.

Enable the human task manager “group work items” feature.

Enable the human task manager “substitution” feature.

Refresh people queries if the people directory changes, to force the people assignments to be evaluated again.

Remove unused people query results from the database.

Remove completed business process instances from the database.

Remove process templates and human task templates that are not valid from the database.

Enable Business Process Choreographer events to be emitted to the Common Event Infrastructure as Common Base Events and/or stored in the audit trail.

Delete some or all audit log entries.

Change the number of retries for business process container messages which cannot be processed.

Change the maximum number of business process container messages in the retention queue.

Query and replay failed messages for business processes or human tasks that could not be processed and have been put on the hold queue.

Business process container messages are stored in the retention queue if a temporary error condition, such as a database deadlock or a connection failure, is detected. After a configurable timeout, messages from the retention queue are fed back into normal processing. After the maximum number of retries, failed messages are put into the hold queue from where they can be replayed by an administrator.

For the task container, if a message can not be processed, it is immediately stored in the task container's hold queue from where it can be replayed by an administrator.

### 6.1.2 Business processes and human tasks

Business processes and human tasks are part of a module created by WID, which is technically an EJB module of an enterprise application (see "Developing Business Processes and Human Tasks" on page 31). A business process template is a deployed process model, and a human task template is a deployed task model. The WebSphere administrative console supports the following administration tasks:

Start template

Stop template

Get state of a template

Each process template and each task template has a state which is either *started* or *stopped*. Templates can be started and stopped individually. If a process template or task template is stopped, no new process instances or task instances of the corresponding business process or human task can be created, respectively. This means that an administrator can disable certain functions of a workflow application, for example, for maintenance reasons or to avoid that further instances of a template that is to be phased out can be created.

Business process instances and human task instances can be administered using the Business Process Choreographer Explorer (see next section).

## 6.2 Working with and Managing Processes and Human Tasks

Once processes and tasks are deployed, business users work on human tasks and interact with business processes. Process and human task administrators manage the actual processes and human tasks which may encompass checking status, repairing processes, and managing work assignment.

The activities of an administrator are common across definitions of human tasks and business processes. Business Process Choreographer Explorer is the graphical user interface that provides the main administrative capabilities for managing business processes and human tasks [WPS InfoCenter].

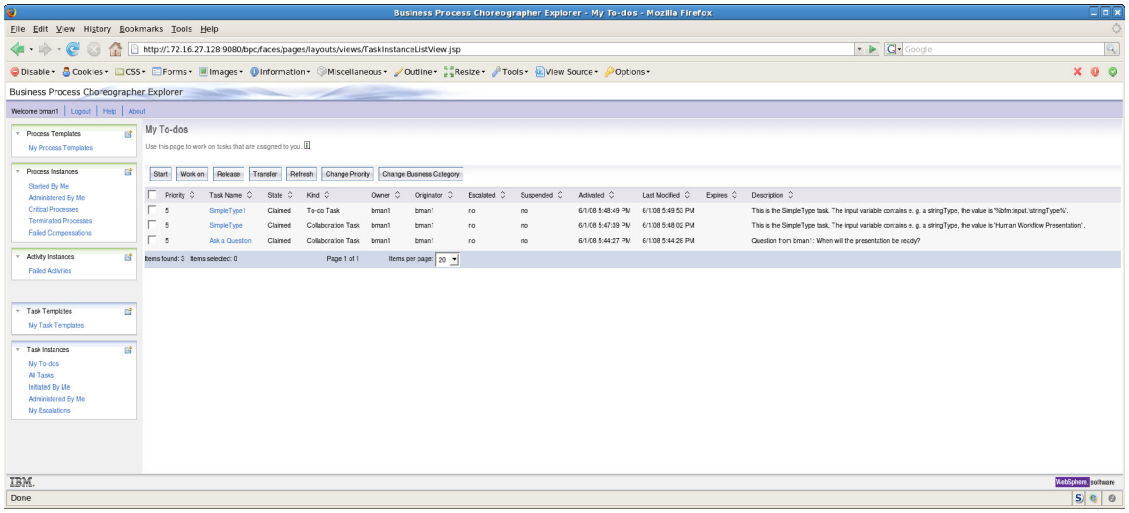


**Figure 20: Business Process Choreographer Explorer with task list**

The user interface provides lists and details of business process human and task templates, their instances and related objects such as process activities, work items and escalations for human tasks. Users can search for these entities using the predefined views or define their own search filters. Graphical views provide information on the template definition of a business process and the current status of an instance of such a process.
To manage and repair processes and manage work assignment, it provides the following capabilities:

Suspend and resume, compensate, restart or terminate process instances as well as delete them.

Restart or force the completion of activities.

Suspend, resume and terminate task instances as well as delete them. Additionaly their priority and business category can be modified.

Create, transfer, delete and cancel work items.

Show the escalations for overdue tasks.

Administrators have access only to those processes and human tasks to which they are assigned as administrators. Dedicated Business Process Choreographer system administrators have access to all entities.

Administrators can work on human tasks assigned to them as potential owners or starters like other business users. Business Process Choreographer Explorer provides generic capabilities to work on human tasks. Lists are provided for tasks administered by the user, tasks initiated by the user, tasks available to work on, or all accessible tasks. The user can initiate tasks and work on tasks, for example claim, cancel and save changes as well as complete an assigned task.

Depending on the target user role and particular requirements of a project, Business Process Choreographer Explorer can be used as-is or customized, or specialized Web clients are developed independently. To support these options, Business Process Choreographer Explorer provides customization capabilities and is built with reusable user interface components.

The input and output forms for tasks can be customized using JavaServer Pages (JSP) fragments which Business Process Choreographer Explorer embeds into the overall Web page. The look and feel can be adapted through Cascading Style Sheets (CSS) definitions. Business Process Choreographer Explorer consistently uses CSS styles for the various elements in the Web pages.

If the views provided out-of-the-box do not meet the needs of the administrators, they can define their own private views. System administrators can define public views for all users and define which of the predefined views to display in an instance of Business Process Choreographer Explorer. By removing all predefined views for a specific kind of Business Process Choreographer object, this kind of object can't be accessed in the client anymore. This allows system administrators to tailor an instance of Business Process Choreographer Explorer to the needs of a group of administrators.
When defining a new view, the administrator selects which columns the view should display, which set of actions to make available – for administrative or processing work – and a set of filter criteria that defines the content of the new view.

Business Process Choreographer Explorer is based on the JavaServer Faces (JSF) framework and a set of JSF components that are available to Web developers to build customized Web clients. The client generator in WebSphere Integration Developer allows to generate a web application for selected tasks based on the JSF components.

The four JSF components can be used to list Business Process Choreographer objects, provide details on them, provide input and output forms for business data and provide commands available in a command bar. The client model is a set of Java classes for each of the Business Process Choreographer entities such as process template or task instance. The JSF components work on these bean-like instances. The list component allows to display the results of any kind of query, if the entities in the result set provide bean-like access methods for the properties to display in the list columns. The command bar component can be used together with the list component to define actions on a single or multiple selected list entry/entries. The message JSF component integrates custom JSPs which the Web developer provides separately. Message and Details component are especially suited for Web clients with the need to handle arbitrary human tasks and business processes.

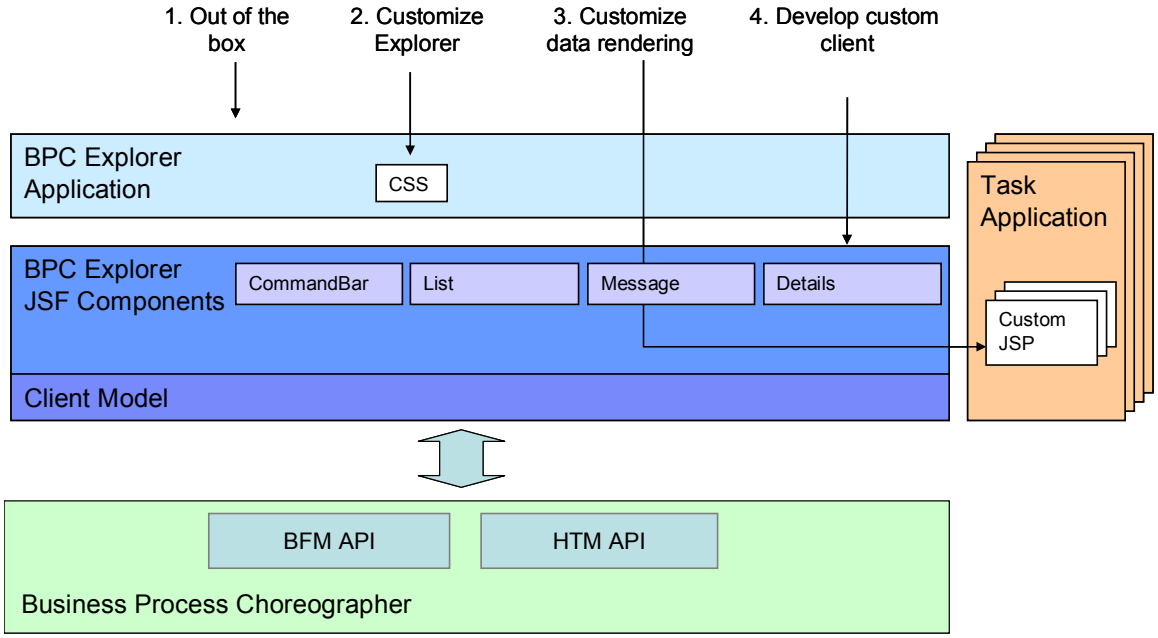


**Figure 21: Structure of Business Process Choreographer Explorer and alternative uses**

The JSF components build on the generic Business Process Choreographer interfaces [WPS InfoCenter].

## 6.3 Versioning

Business Process Choreographer supports versioning of processes and tasks. This allows for scenarios like the following: A company requires a new version of its claims handling process with the beginning of each year. Existing process instances need to continue using the prior version; however, new process instances will employ the new version as soon as it is becoming valid.

Business Process Choreographer's versioning is based on valid-from timestamps. Different versions of a process or task are named equally; however, differ with regard to their valid-from timestamps. You model different versions using the so-called *validFrom* attribute of a process or task definition. At runtime, multiple versions of the same process or task may be installed and in use concurrently.

To exploit versioning, *late binding* is required as the means to allow selecting the currently valid process or task when creating and starting a new process instance or task instance. In a late-binding scenario, the decision on the version to be used happens dynamically at runtime. Business Process Choreographer supports late binding of processes and tasks in various scenarios.

Late binding can be employed using the generic Business Process Choreographer API for processes and tasks. Given the name of the process or task, the API allows you to create a new instance and let Business Process Choreographer determine the currently valid version of the process or task to be used.

A process calling another process may use late binding to resolve the currently valid version of the process to be called. You model this as part of the calling process.

*Early binding* as opposed to late binding enables using a dedicated version of a process or task. In the case of early binding, the valid-from specification for a process or task is irrelevant as you decided to go with a particular version and do not want this to change dynamically whenever a newer version is being introduced.

Early binding can be employed using the generic Business Process Choreographer API: Given a dedicated handle to a particular process version, you can create and start instances from that version.

Early binding always happens between statically wired SCA components, be they processes or tasks. Here the decision on the version is made before deploying your integration application.

Once a process instance or task instance is created, it stays with the version it was created with for its lifetime – it's bound to it.

# 7 References


[BPC Prog Mod] WebSphere Process Server V6.1 Business Process Choreographer Programming Model, available via http://www.ibm.com/support/docview.wss?uid=swg27012602

[BPC zone] Working with Business Process Choreographer – Essential Business Process Choreographer reference material, available via http://www.ibm.com/developerworks/websphere/zones/was/wpc.html

[BPC Cleanup] Process Cleanup Service for Business Process Choreographer, available via http://www-1.ibm.com/support/docview.wss?uid=swg27007816

[BPC Compensation] Using Compensation in Business Processes with Business Process Choreographer, available via http://www3.software.ibm.com/ibmdl/pub/software/dw/wes/pdf/0604_robeller-compensation-bpc.pdf

[BPC Human Tasks] Human Tasks in different Scenarios, available via http://www-1.ibm.com/support/docview.wss?uid=swg27006889&aid=1

[BPC Event Handlers] Event Handlers in Business Process Choreographer, available via http://www-128.ibm.com/developerworks/library/ws-eventhandler/

[BPC PHP] Access WebSphere Process Server V6.0 Business Processes with PHP, available via http://www-128.ibm.com/developerworks/websphere/library/techarticles/0602_smolny/0602_smolny.html

[BPC Reference] Working with Business Process Choreographer, available via http://www-128.ibm.com/developerworks/websphere/zones/was/wpc.html

[BPM Samples] Business Process Management Samples & Tutorials - Version 6.1, available via http://publib.boulder.ibm.com/bpcsamp/index.html

[BPEL4People] WS-BPEL Extension for People specification, v1.0, June 2007, available via http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

[WS-HumanTask] Web Services Human Task specification, v1.0, June 2007, available via http://www.ibm.com/developerworks/webservices/library/specification/ws-bpel4people/

[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

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

[BPELJ] WS-BPEL Extension Java Technology, a joint whitepaper by BEA and IBM, March 2004, available via http://www-106.ibm.com/developerworks/webservices/library/ws-bpelj/

[SCA] Service Component Architecture Programming Model, available via http://www.osoa.org/display/Main/Service+Component+Architecture+Home

[SDO] Service Data Objects Programming Model, available via http://www.osoa.org/display/Main/Service+Data+Objects+Home

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

[SOA Tasks] SOA programming model for implementing Web services, Part 8: Human-based Web services, M. Kloppmann, S. Liesche, G. Pfau, M. Stockton, available via http://www.ibm.com/developerworks/webservices/library/ws-soa-progmodel8/

[SysJrnl SOA] IBM Systems Journal Issue on Service Oriented Architecture, IBM Systems Journal Volume 44, Number 4, 2005, available via http://researchweb.watson.ibm.com/journal/sj44-4.html

[WPS Homepage] WebSphere Process Server Homepage, available via http://www.ibm.com/software/integration/wps/

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

[WPS and WID] WebSphere Process Server and WebSphere Integration Developer, on IBM developerWorks, http://www-128.ibm.com/developerworks/websphere/zones/businessintegration/wps/wps.html

[WS-Addressing] Web Services Addressing, W3C Recommendation, May 2006, available via http://www.w3.org/TR/2006/REC-ws-addr-core-20060509/

[WS-BPEL 2.0] Web Services – Business Process Execution Language Version 2.0, OASIS Standard, April 2007, OASIS Technical Committee, available via http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.pdf

[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

# 8 Trademarks

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

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