

Model business processes for flexibility and re-use: A component-oriented approach

Skill Level: Intermediate

[Jose A. De Freitas \(dfreitas@uk.ibm.com\)](mailto:dfreitas@uk.ibm.com)
Advisory Software Engineer
IBM

22 Apr 2009

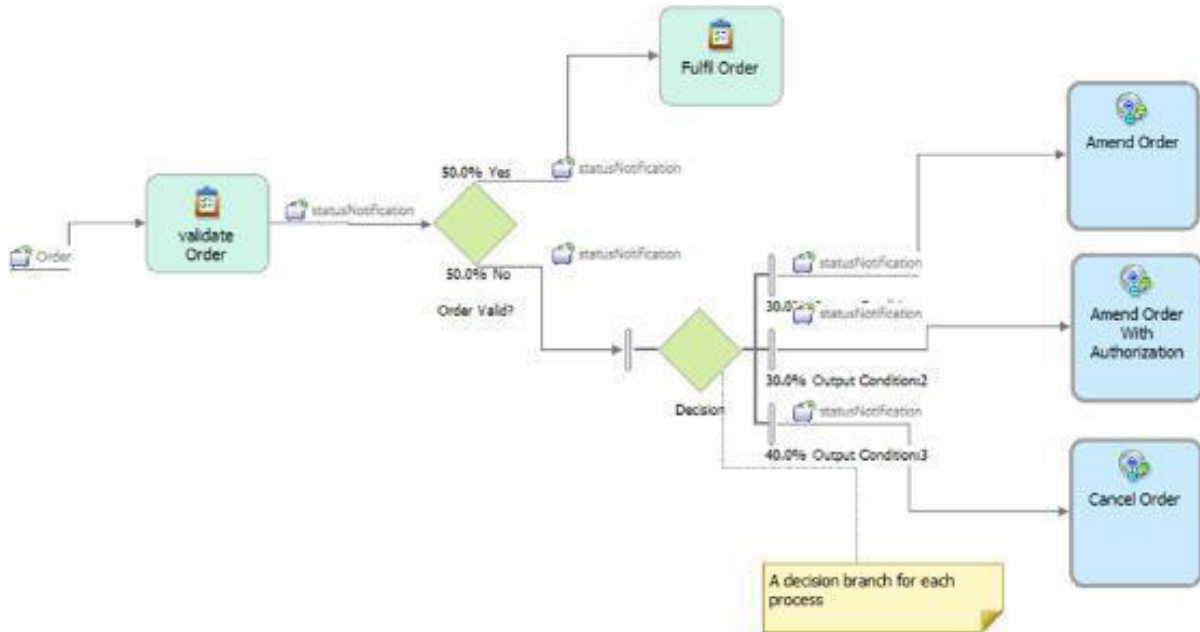
This article describes a component-oriented approach to business process modeling that allows you to capture process variability and ensure that your model is reusable. It describes modeling patterns and practices in WebSphere® Business Modeler that will help you achieve this goal.

Introduction

Today's business dynamics are mandating that business processes be increasingly responsive to change. This makes it critical that business process models be modular and flexible, not only for increased modeling agility but also for the greater robustness and flexibility of executing processes. A traditional approach to business process modeling frequently results in large models that are difficult to change and maintain. Because of their size, these models are not very flexible. They are also not conducive to dynamic process selection — that is, parts of the process model cannot be easily replaced at execution time.

A good example of a scenario where dynamic process selection is justified is the case where different exception handling processes may be invoked depending on the type of error that is encountered. The traditional way to solve this problem is to use a multiple-choice decision. This is illustrated in the trivial process order example shown in Figure 1. If the order is not valid, one of three paths is chosen depending on the outcome of the decision.

Figure 1. Hard-wired choices limit flexibility and increase process size

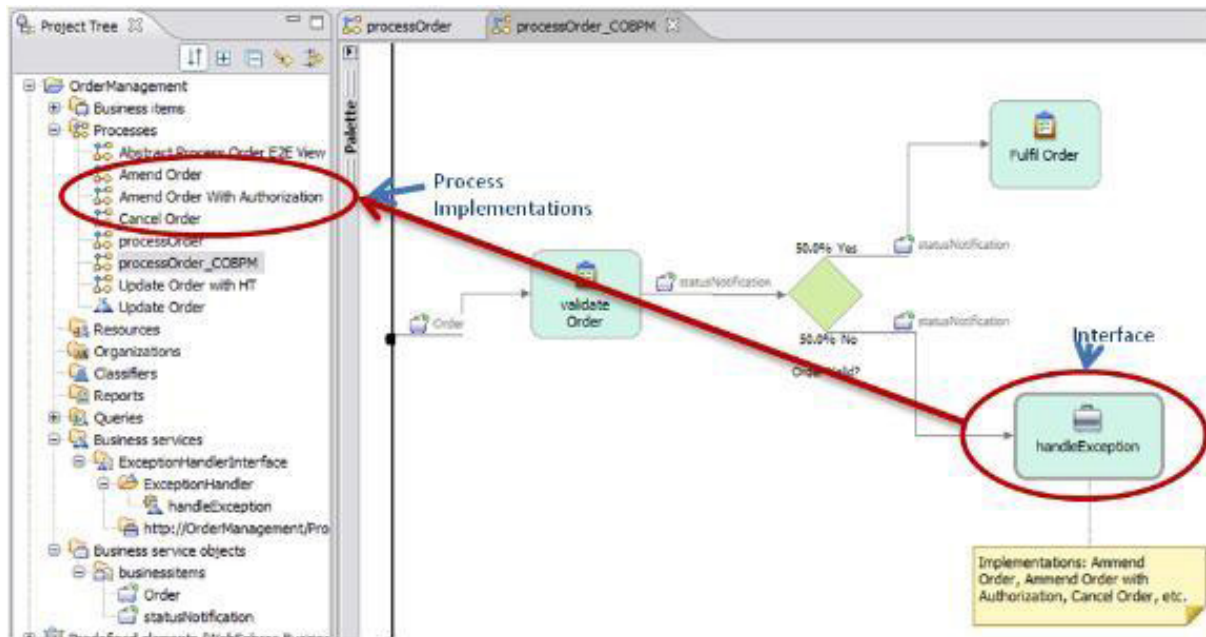


An obvious limitation of this approach is that for each new error handling process a new decision branch must be created. Therefore, the process needs to be changed and all the related documentation, deployment, and testing must be repeated. It also means that you could end up with a very large process.

Business processes as components

An alternative, more modular approach to solving the problem described above is one where a business service interface "stands in" for one of many compatible processes (compatible, meaning that they have the same interfaces). A single process can then be dynamically selected at execution time. See Figure 2.

Figure 2. An interface “stands in” for one of many possible process implementations

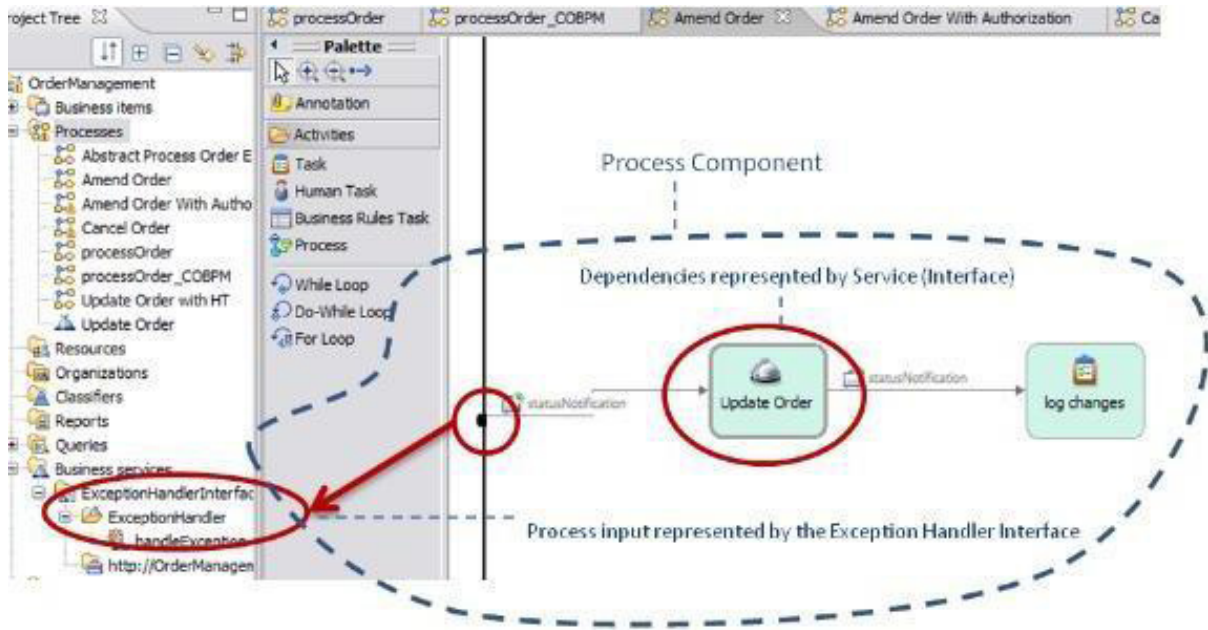


At first glance this may appear to be only a subtle change in modeling style, but the consequences are far-reaching. First, the process interfaces are clearly defined through their inputs and outputs (Figure 3). But most importantly, the process dependencies are also clearly exposed through interfaces (business services in Business Modeler). This effectively means that a process modeled in this way is a *component* in the sense that it embodies the most fundamental property of components: *a component declares the services it provides and those that it depends on*.

Cohesion is another property that is typically associated with software components and that is also highly desirable in the context of business process modeling. While the idea behind component cohesion may be simple (keep related functionality together), it does require good analysis skills to achieve. The number and complexity of process dependencies may provide a measure of process cohesiveness; the lower the number of dependencies and the lower the dependency complexity, the more cohesive a process tends to be.

If we agree that business processes are components if they (1) are cohesive and (2) expose services and dependencies, then "component-oriented business process modeling" is an adequate term to describe the activity of creating such process models. This notion of business processes as components is not one that is exclusive to the technical or implementation domain. A component view of a business is also necessary for greater business agility. Thus if your goal is to better align business with technology, and to easily respond to changing business requirements, then the use of components in one domain necessarily implies a component view in the other.

Figure 3. A process component exposes its services and its dependencies



It is trivially true that components may be composed of other components, but the way in which they are composed may differ. For greater flexibility, we propose that a business process component not be statically bound to its constituent components. Instead dependencies should be *injected* at run time based on the outcome of the execution of business rules. The use of the term *injected* is used here to draw an analogy to the more technical notion of "dependency injection" (used in the popular Spring framework and the Service Component Architecture (SCA) that has delivered increased reuse potential, robustness and flexibility to solutions built using this approach). See [Resources](#) for details on Spring and SCA.

Component-oriented process modeling using WebSphere Business Modeler

The terms *service* and *interface* are used interchangeably because, in Business Modeler, a service typically represents an interface to some external service implementation. Business Modeler distinguishes between those services that are defined in Business Modeler (sometimes called global services) and those that are imported into Business Modeler as Web Services Definition Language (WSDL) services. Both types of services can be used as interfaces in the component-oriented approach, the former in a top-down modeling approach and the latter in a bottom-up approach (please refer to [Component-oriented process modeling and SOA](#)). Web services definitions can be imported from the file system, from WebSphere Services Registry and Repository, or from Rational® Asset Manager.

With component-oriented process modeling, components are represented as

disjointed global processes in Business Modeler. It is therefore necessary to:

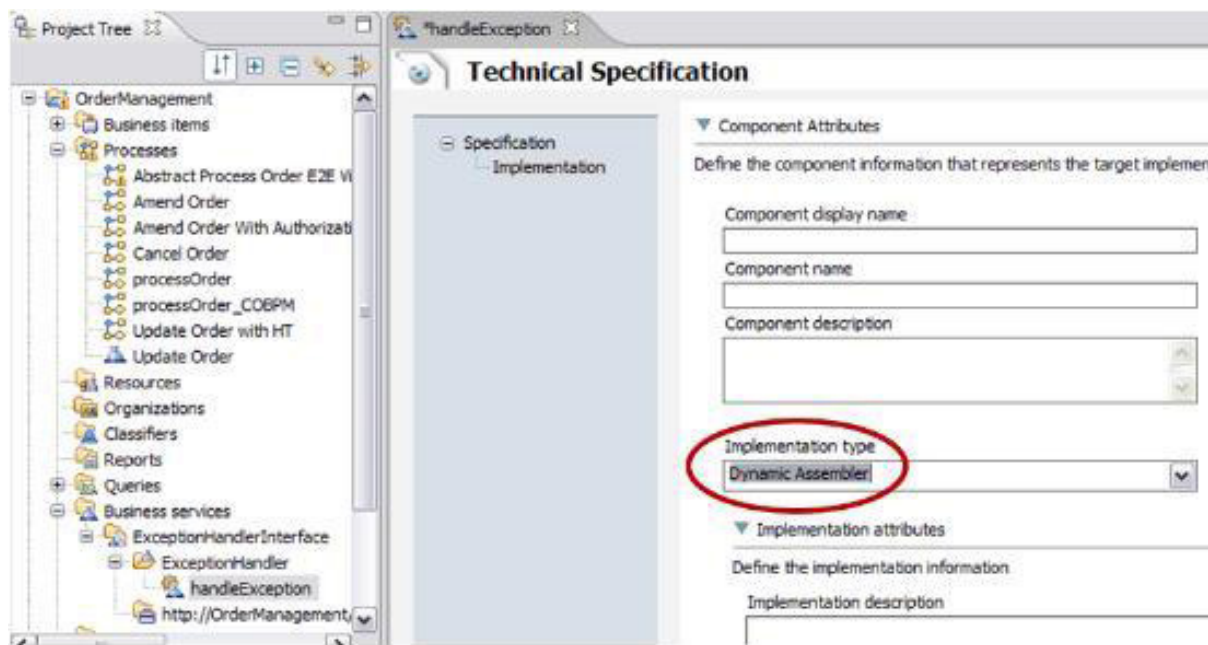
1. Define the “glue” that holds them together in a containing business process
2. Define a modeling pattern that enables an end-to-end view of this process

These two steps are outlined in the following sections.

Business rules serve as the “glue”

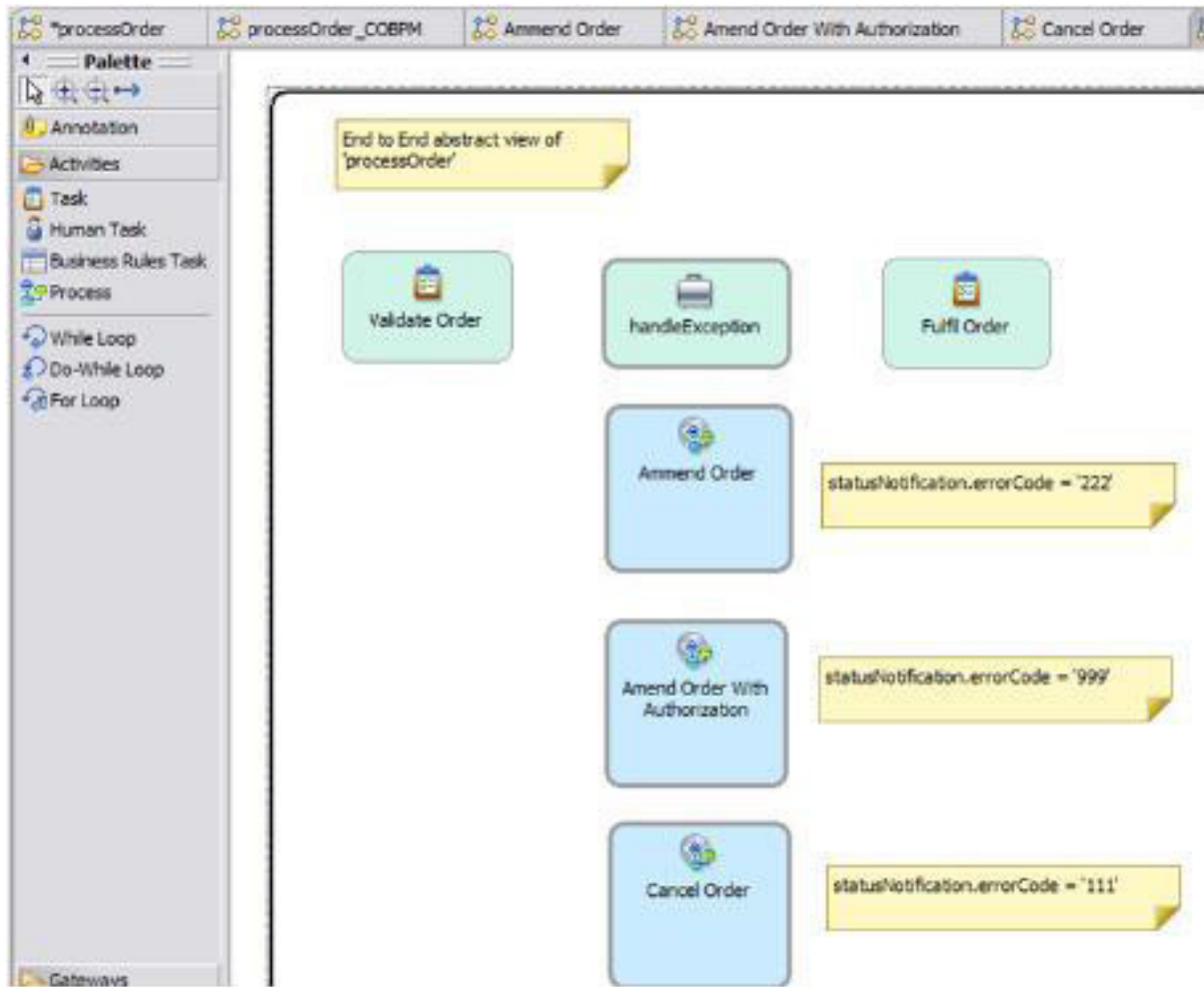
The most flexible way to govern the dynamic composition of processes and services is through the use of business rules. In a typical implementation, business rules are used to inspect the data that flows through a particular node or point in the process flow (known as a *point of variability*) and to make a decision that determines the next path in the flow. In our example, a simple rule could be “if the outcome of the validation is an error code of ‘999’ then the corresponding amendment to the order must be authorised by the supervisor”. The phrase “must be authorised by the supervisor” simply means that the process that implements this capability (*Amend Order With Authorization*, in the example) is dynamically invoked. WebSphere Business Services Fabric (hereafter referred to as Business Services Fabric) is ideally suited for defining, managing and executing such rules. In Business Modeler, you can designate a service interface to be a *Dynamic Assembler* (Figure 4) to indicate that it is a Business Services Fabric point of variability. Then, you can define rules and policies for the component within Business Services Fabric. Refer to [Resources](#) for details on points of variability.

Figure 4. Point of variability identified as a Business Services Fabric Dynamic Assembler



While Business Modeler supports the definition of business rules to control the outcome of decisions, business rules that govern the dynamic selection of endpoints must be defined elsewhere, for example, in Business Services Fabric. For this reason, we propose that the rules for dynamic processing be documented within Business Modeler as annotations (shown in Figure 5).

Figure 5. Abstract representation of process variability



Proposed process for end-to-end representation

Because a high-level business process (one that contains other process components) has flows that will dynamically vary, there isn't one single end-to-end view of the process. Such a view can, however, be depicted in an abstract way. We propose the representation pattern shown in Figure 5 to enable the visualisation of process variability.

In this diagram *Validate Order*, *handleException* and *Fulfil Order* constitute the (trivial) end-to-end process, while *Amend Order*, *Amend Order with Authorization* and *Cancel Orders* are the possible implementations of *handleException*. The

annotations beside each implementation document the condition (business rule) that triggers the selection of the corresponding implementation at execution time.

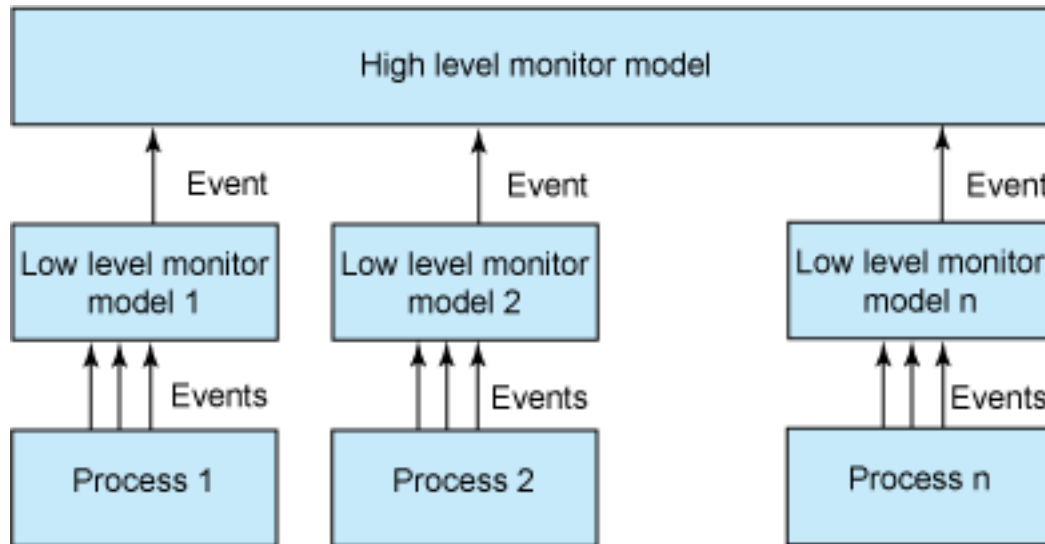
Some additional modeling considerations when using component-oriented process modeling are as follows:

- **Asynchronous processes.** There is nothing special about how asynchronous processes are modeled using component-oriented process modeling. Any process component can issue a “fire-and-forget” request and any component (with the right interface to receive the response) may act as a “listener”. There is, however, one small difference. While “standard” process components are invoked as a result of the evaluation of business rules, an asynchronous process component is triggered by the arrival of an event or message. It is therefore important to designate that a given component is a listener (an annotation may suffice) so that it is implemented accordingly.
- **Sharing context.** Sometimes it is necessary to keep business object states across process component invocations. For example, if the complete Order object is not passed as an input parameter to all process components, it may be necessary to store it somewhere so that each component in the end-to-end order process may have access to it. That “somewhere” may be a database, or it may be some sort of “process context” (unless properly managed by the underlying software, the latter should be avoided because of the scalability implications). How the process state is kept across process component invocations should not be a concern of business process modeling. However, the fact that the process is using (or producing output from) data that does not derive from its inputs should be recorded in the process model. In Business Modeler a process and its (connected) sub-processes can share data through the use of a global repository, but with process components (that are not connected to the parent process) this is no longer possible. The solution that we propose is centred around the notion that context interactions are dependencies that the process has on a “context service”. As such, they are represented by a service interface and should not be treated any differently than any other dependencies.
- **Process simulation implications.** Because business rules and policies for dynamic process assembly cannot be directly defined in Business Modeler, its simulation engine is not able to simulate processes that are composed of other decoupled processes. Each process can be simulated individually but, if the intent is to improve the process through successive simulations, it is obvious that the optimisation of the parts may not result in the optimisation of the whole process. If this is the goal then interfaces have to be replaced by their respective process implementations as sub-processes.

- Implications for business process monitoring.** In Business Modeler you can define business measures and key performance indicators (KPIs) that are specific to a particular process. A WebSphere Business Monitor model may be generated from each Business Modeler process. With many decoupled process models forming part of a larger (parent) process, it is no longer possible to generate a single monitor model for the parent process and all its process components. This is by no means a bad thing. In the same way that we now have decoupled business processes, we also end up with corresponding “decoupled” monitor models and, as a by-product, a much more modular approach to monitoring.

The attentive reader will at this point be thinking that this will work well if the intent is to monitor each process component independently. It will not work so well if we wish to monitor the end-to-end process — for example, if we wish to establish metrics that cover the complete life span of an order. The solution is to use an approach that has already been proposed as a best practice for the creation of monitor models. The approach consists of the separation of monitor models into two separate monitoring levels: *high-level monitor model* and *low-level monitor model*. Low-level monitor models receive events from the running processes and in turn propagate (appropriate) events to the high-level monitor model. The original intent of this approach was to establish a certain degree of insulation between the running process and the higher-level monitoring artifacts, such as KPIs and business measures. With decoupled component processes you can use the same approach but in this case we have many low-level monitor models contributing events to a single high-level monitor model that oversees the whole process (Figure 6).

Figure 6. Low-level models contribute to a high-level model for end-to-end monitoring



Business Modeler can already generate high- and low-level monitor models, but the

generation of multiple monitor models that contribute to a single high-level model is not supported. This must be done in the Monitor Model Editor.

Component-oriented process modeling and SOA

In most business environments, process models are created not only to describe new capabilities, but also to define how those capabilities interface to existing systems. In a growing number of cases the interfaces to existing legacy systems are implemented as web services that conform to Services Oriented Architecture (SOA) design principles. These services can be imported into Business Modeler as *business services*. Once in Business Modeler, business services may be referenced by one or more business processes.

This approach requires two types of analysis: top-down, to understand what the new business capabilities are and how they interact with each other and bottom-up, to understand what capabilities already exist and how they can be reused in the new solution. Successive iterations of this analysis process result in the desired goals of asset re-use and better alignment of business processes and technology.

In component-oriented business process modeling, business services are no longer just interfaces to external systems or services, but they can also be interfaces to new or existing process components. Thus the component-oriented approach benefits from (but does not necessarily prescribe) increased rigor and discipline in the way that processes are modeled to ensure that they fully realize their re-use potential. Such reusable process components must then necessarily also become subject to the top-down or bottom-up SOA analysis process.

Summary

This article addresses the issues of model size, re-use, modularity and flexibility in business process models by proposing a component-oriented approach to business process modeling. It defines a process component to be a cohesive business process that exposes its inputs and outputs as interfaces and declares all its dependencies as interfaces. It also maintains that such process components can be dynamically assembled into larger processes or process components.

Resources

Learn

- [Business Process Modeling Notation Specification](#). This describes the BPMN specification of the Open Management Group.
- [Introduction to the Spring framework](#) (developerWorks, 2005). This article provides an introduction to Spring as an IOC (Inversion of Control, also known as Dependency Injection). container.
- [Make composite business services adaptable with points of variability, Part 1: Choosing the right implementation](#) (developerWorks, 2007). First of a series of articles that elaborate on points of variability in business processes.
- [Make composite business services adaptable with points of variability, Part 2: Using dynamic service mediation in WebSphere Business Services Fabric](#) (developerWorks, 2007). Second in the series, this article focuses on the capabilities of WebSphere Business Services Fabric.
- [Introducing SCA](#). This is an excellent SCA introductory paper by David Chappel.
- Browse the [technology bookstore](#) for books on these and other technical topics.

Discuss

- Check out [developerWorks blogs](#) and get involved in the [developerWorks community](#).

About the author

Jose A. De Freitas

Jose is a member of the WebSphere Business Modeler SWAT Team. His current interests are business process management and tools architecture.

Trademarks

IBM, WebSphere, and Rational are trademarks or registered trademarks of IBM Corporation in the United States, other countries, or both.

Microsoft and Windows are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered

trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Other company, product, and service names may be trademarks or service marks of others.