

/THEORY/IN/PRACTICE

Essential Business Process Modeling

Michael Havey

Prescription for a Good BPM Architecture

BPM STANDARDS ABOUND, EACH WITH A DISTINCT DESIGN AND FEATURE SET. This chapter scavenges these approaches in search of a general BPM application architecture that is conceptually comprehensible and meets real-world requirements. A good architecture uses the technique of divide-and-conquer to reduce a difficult problem to smaller, more manageable parts, and where possible, it solves each part not by inventing new technology but by reusing an existing approach. Applying this technique, we ask: in a BPM architecture, what is the problem to be solved, what are its parts, and which standards, if any, solve them?

The architecture presented here is intended as a reference model—with similarities to the WfMC's model (see Chapter 7) and the proposed stack of the BPMP (see Chapter 6)—targeted at product and services architects alike. The appeal for product architects is obvious: the model, though lacking the level of detail for a micro design, has the same form as a BPM product, and can help guide the overall construction. Services architects, who typically advocate buying a good vendor solution and customizing it rather than building the entire solution from scratch, need to comprehend the essential nature of the base product. Rather than treating it as a black box, these architects should have a sense of curiosity analogous to that of a driver who lifts the hood of a car and seeks to grasp the basic mechanics of all those belts and gears. There are a number of reasons why:

- BPM is an emerging subject, and in the current morass of vendors, standards, hype, and theory, having a crystal-clear notion of a good solution (whether built or bought) gives an architect a competitive advantage over the many who are still learning.
- Customers know the essential architecture of some of their IT products and technologies (e.g., database, operating system, network router), but they probably have a fuzzier notion of BPM, and they expect the services architect to help explain it to them and guide them through adoption.
- Vendor documentation seldom distinguishes clearly between standard and proprietary features. (The architecture in this chapter tries to avoid the second of these.) If the customer has been sold on BPEL, the XYZ foundation classes, BAM, and the Intel process optimizer, the services architect needs to know that BPEL and BAM are standard solution-building blocks, and to avoid the proprietary XYZ and optimizer if the customer foresees migrating to another vendor implementation someday. The customer probably does not know the difference.
- The architecture for the customer's application must use and extend the out-of-the-box features of the stack. To know how to do this, the architect needs to understand those out-of-the-box features. For example, what tools are provided to support process development and deployment? What is the strategy to patch or upgrade live production processes? What is the data model, and how can customer data entities be worked into it? How are external systems interfaces built, and how are they called from processes? How can the monitoring subsystem be customized to serve the customer's business and operational requirements?

Designing a Solution

To design a good BPM solution, you must first step back and examine the project's environment: understanding the problem, noting the local and larger-scale perspectives, and only then creating a design and testing your solution.

Understanding the Problem

To understand what a good solution looks like, you must first understand the scope of the problem to solve. The main requirement of a BPM application is the ability to design, run, and monitor and administer business processes that incorporate human and system interactions, described as follows:

Design

The design of a business process is intuitively a flowchart that outlines the steps performed over time in the resolution of a business problem. Unlike most object-oriented designs, whose audience is the technical team of a project, a process design is crafted and comprehended by both business and technical analysts. Business analysts are involved because they understand the business aspects of the process best; the design is simply a rigorous expression of what they frequently draw on paper or on a whiteboard. The level of rigor, plus the anticipation of implementing a software solution to

the design, draws in technical analysts. Thus, business and technical designers require a common design notation that is at once business-oriented and amenable to computer processing. They also require a graphical editor in which to sketch their design.

Run

The early workflow engines that were actually able to run designed processes, and thus to automate the execution of formerly manual procedures, must have been perceived as miraculous. Today, this miracle is a core requirement: “Run what I just drew!” But conceived more dispassionately, the executability of a process is a matter of having a runtime engine and an executable language that the engine knows how to run. Moreover, a mapping is required between the design notation and the executable language, enabling the automated generation of executable code from the design. The manual coding of a process is undesirable because it slows the development cycle and ensures gaps between the design and the execution.

Monitor and administer

Monitoring, the ability to watch the progress of running processes, is crucial for production applications for two reasons: the detection of exceptions (ensuring processes are progressing as expected and not getting stuck), and real-time ad hoc querying (e.g., finding all active processes for customers in a particular account range). Administration is the management of, and the ability to effect changes on, processes on the runtime engine. It includes the ability to install, shut off, or turn on process designs, and to suspend, resume, or terminate running processes. Monitoring and administration require a management language and a graphical management console that can watch and modify the BPM system.

The model architecture must also smoothly accommodate both human and computer interaction:

Human interaction

A manual task is a step in a process that is meant to be executed by either a specific person, or a person in a specific role, within the company. For example, in an insurance claims process, the approval of a claim is a manual step that can be completed by any claims adjuster. To support manual processing, the architecture requires a role-based security model (which maps users to roles), a graphical worklist console (to show the user a list of assigned manual activities and enable completion of an activity), and a corresponding worklist programming interface for the exchange of information between the console and the runtime engine.

System interaction

Other steps in a process must be able to call or be called by software components that are not built into the engine. These components can be external participants (e.g., a request for price quote from another’s company’s process in a buyer-seller relationship), internal systems (e.g., send email, query database, perform mainframe transaction), or simple inline code snippets (e.g., perform calculation, parse data). The architecture should support incoming and outgoing integration interfaces for a variety

of technologies—web services, MQ/Series, JDBC, J2EE (EJB, JMS), Java, C#, and COM. The architecture should also be conversant in XML, the dominant message language of enterprise computing.

The Local Perspective Guided by the Global Contract

The job of a BPM architecture is to describe the local perspective of one company, not the global view of all partner interactions. True, the designers must be aware of the nature of partner interactions, but only to ensure that the company is compliant with interprocess protocols. As Figure 2-1 illustrates, a company that is a Seller must know the rules of interaction with Buyer, Inventory, and Credit Card Company partners, but its main concern is the construction of its own Seller processes.

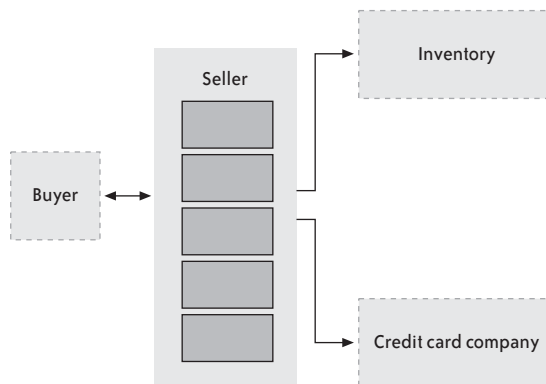


FIGURE 2-1. *Isolate the client company from its partners*

On the other hand, how can the designers be certain that their local processes interact exactly as they should with partners? Several years ago, the “global contract” governing this communication would have been documented in writing by a third-party body. As they modeled their processes, the designers would have kept a close eye on the documentation to ensure that their communications were compliant. But with the advent of web services choreography languages, notably WS-CDL, the contract can be coded in a formal language, around which a tool (considered in greater detail in the section “Local View of Choreography: WS-CDL Toolkit”) can be built to generate, or to validate the compliance of, local processes.

The first cut of the Seller process, for example, can begin as code generated by a tool that interprets the choreography covering Seller, Buyer, Inventory, and Credit Card Company. Or, supposing the first cut had been written manually, at any subsequent stage of development, the code can be run through the tool to validate that Seller interfaces correctly with Buyer, Inventory, and Credit Card Company. Such a tool simplifies process design, and eliminates the eyeball approach.

Components of a Good Design

Figure 2-2 shows the main pieces of the BPM architecture and their relationships.

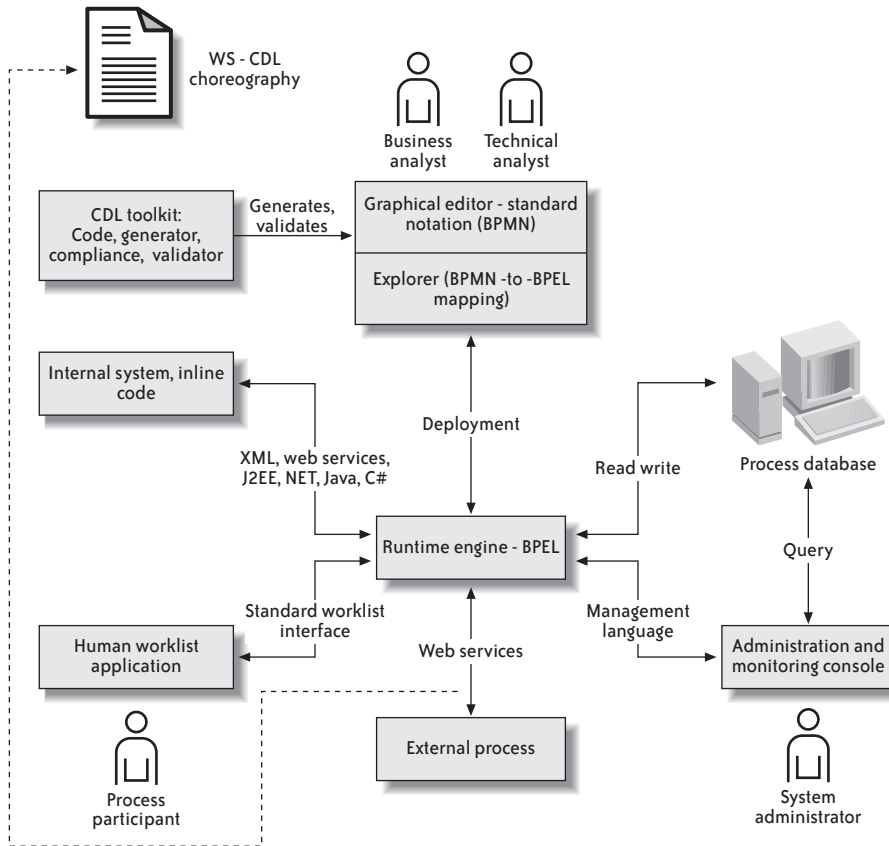


FIGURE 2-2. A good BPM architecture

In the center—and at the heart—of the system is the runtime engine, which executes processes written in BPEL. Business and technical analysts design the processes using a graphical editor that supports BPMN. The editor includes an exporter that generates BPEL XML code from BPMN diagrams.

Human and computer interactions drive the execution of processes in the engine. The people who participate in the process use a human workflow graphical application that connects to the engine through a programmatic worklist interface. The interface allows the user to view and execute pending manual activities. There are two types of computer interactions: internal and external. *Internal interactions*, which reside on the company’s network but are outside of the engine’s address space, are accessed by integration technologies such as web services, J2EE, or COM, with XML as the probable message format; internal interactions can also be more lightweight inline code snippets written in programming languages such as Java or C#. *External interactions* are typically web service-based communications, governed by choreographies or business-to-business collaborations, with the processes of other companies. BPM system administrators use a graphical administration and monitoring console to maintain and track the state of the

engine's processes. The console uses a management language to interface with the engine. The runtime engine persists process state to a database; the console hits the database directly, rather than using the management language, to perform ad hoc process queries.

For applications involving complex interactions with external participants (e.g., a B2B process), a WS-CDL choreography toolkit generates a basic BPMN model that captures the communications required of the local process; it can perform a validation, or choreography compliance check, of that model.

Run It on an Application Server

Never build a BPM application from scratch. Instead, run it on an application server, to leverage built-in application server facilities such as security, transactions, system management, and the pooling of client connections and resources. Table 2-1 summarizes a possible J2EE application server implementation of the BPM architecture.

TABLE 2-1. *BPM on J2EE*

Component	J2EE implementation
Engine	Enterprise JavaBeans
Human worklist application	Java Server Pages™ (JSP) web application
Worklist interface	EJB or web service façade to engine
Internal interactions	EJB, Java Messaging Service™ (JMS), Java Connector Architecture™ (JCA, also known as J2C), Java Database Connectivity™ (JDBC), web services via servlets
External interactions	Web services via servlets
Administration and monitoring console	JSP web application
Management interface	Java Management Extensions™ (JMX)
Process database	JDBC access
Graphical editor	Offline Windows program
Exporter	Offline utility
WS-CDL toolkit	Offline utility

This solution is a combination of out-of-the-box product functionality and custom development, as shown in Figure 2-3. The application server layer, for example, might require custom EJBs to provide functions to call from processes and custom JSPs to use in custom management consoles. The BPM layer might require extensions to the process data model and the development of reusable processes.

BPM - Product	BPM - Custom
App server - Product	App server - Custom

FIGURE 2-3. *BPM on an application server*

Components of the Design

Having glimpsed the BPM architecture “from 30,000 feet,” we now descend to a lower altitude to get a closer look at each of its major components. This section examines the requirements and proposes designs for the notations and graphical tool, the runtime engine, the human and system interaction interfaces, the administration and monitoring facilities, and the CDL toolkit.

Notation and Graphical Tool

The need for a graphical process modeling language is often overlooked in the BPM standards race. Most BPM languages are XML-based and can be relatively difficult to compose or read. Design is best communicated with diagrams. Scan through a typical object-oriented component design document, for example, and you will discover that a single UML class diagram can convey most of the intended meaning, making many of the surrounding words redundant. If a picture is sufficiently rich, it is worth *more* than a thousand words! Standardization adds even more value. A diagram that is drawn to a standard specification is familiar to a wide audience, and its semantics are also clearly understood. Readers get the gist of a haphazardly drawn assemblage of boxes and arrows, but the lack of precision in representation makes it less intelligible.

BPM has two good graphical modeling notations—BPMN (Chapter 6) and the UML activity diagram (Chapter 3)—but BPMN is the preferred choice for our architecture because it is more expressive (it supports most of the “P4” patterns introduced in Chapter 2 and elaborated on in Chapter 4) and has a mapping to BPEL.

ITpearls’ Process Modeler tool, for example, is a BPMN drawing tool with BPEL export capabilities. As Figure 2-4 shows, ITpearls provides a set of stencils that are imported into Microsoft Visio.

A BPMN diagram is drawn by dragging shapes from one of the stencils onto a canvas sheet. The ITpearls tool also embeds a Process menu in the Visio menu bar. As shown in the figure, the designer can export the diagram to a file; at the time of writing, this function was planned for BPEL and BPML formats but was not yet available.

The ability to manage the technical implementation detail is one of the greatest shortcomings of modeling languages. A BPMN drawing dragged onto a Visio canvas captures perhaps 75% of the required processing, but the remaining 25% requires development tool capabilities, such as a source code editor and property sheets. For example, the following are not easily modeled using drag-and-drop tools:

- A decision node that is backed by a Java code snippet that performs a complex bit of decision logic.
- An activity that sends a message to an internal system includes a property sheet that prompts for the address of the target system, the content of the message and message header information.

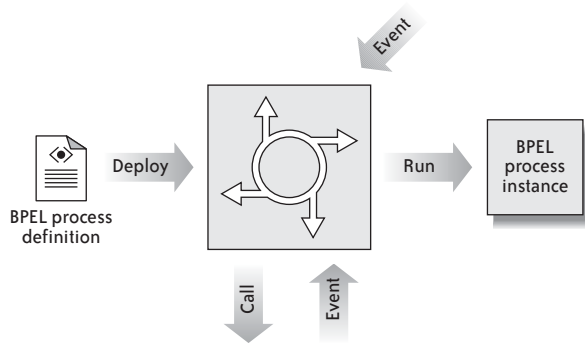


FIGURE 2-5. BPM runtime engine

NOTE

For Java programmers keen to see a good implementation of a BPEL engine, the open source ActiveBPEL is a good bet. The web site <http://www.activebpel.org> provides source code, binaries, and documentation, including a high-level description of the engine's architecture.

Human Interaction

The application's support for human participation in a process requires a worklist interface that allows the process to *coordinate*—that is, initiate, then wait for completion of—manual work tasks. Users view and execute pending tasks in a user-friendly graphical console designed for usability and productivity. Figure 2-6 illustrates a web-based worklist console, known as the “Travel Agent Portal,” for a travel agency.

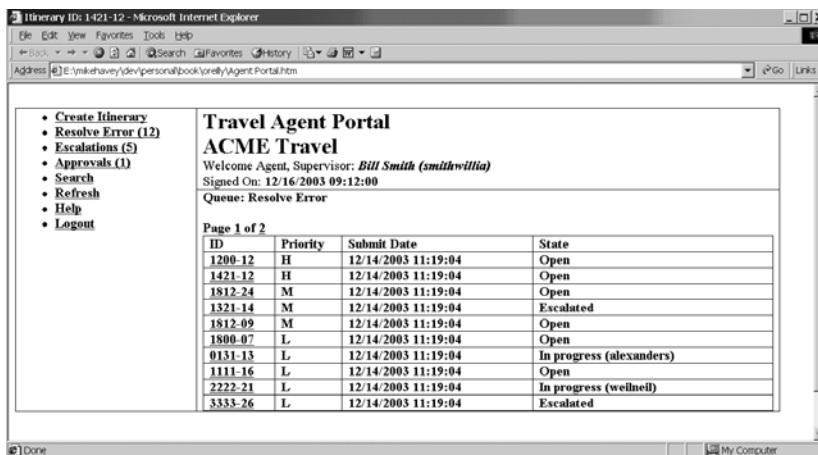


FIGURE 2-6. Human worklist console for a travel agency

The user, named “Bill Smith,” sees three panels. The banner at the top of the second panel welcomes him to the site and reports that he performs the roles of Agent and

Supervisor. The left panel is a command menu that, among other things, presents him with links to view three lists of tasks: Resolve Error, Escalations, and Approvals. (The number in parentheses indicates the size of the list.) When one of these links is clicked, a summary of the task list is populated into the main panel in the bottom right. Currently, the panel shows the Resolve Error list. When the user clicks on one of its items, a separate window pops up showing information about the task and providing an editable form to complete it and send it back to the waiting process.

Behind the scenes, a web service called the Worklist offers two operations:

`initiateTask`

An asynchronous method called by the process to commence the task. The process passes its identity (e.g., process ID, correlated data, service endpoint), a unique identified of the task (e.g., Resolve Error), the intended user or role (e.g., the Agent role), and any other data required by the user to work the task.

`taskComplete`

A callback to the process indicating completion of the process, passing back any associated data.

The state of the task is persisted to a system management database (described in detail in the later section “Administration and Monitoring”). On `initiateTask`, the Worklist web service writes the details of the task to a table; the worklist console pulls data from that table for display to the user, and updates it as the task progresses. The normal sequence of events is shown in Figure 2-7.

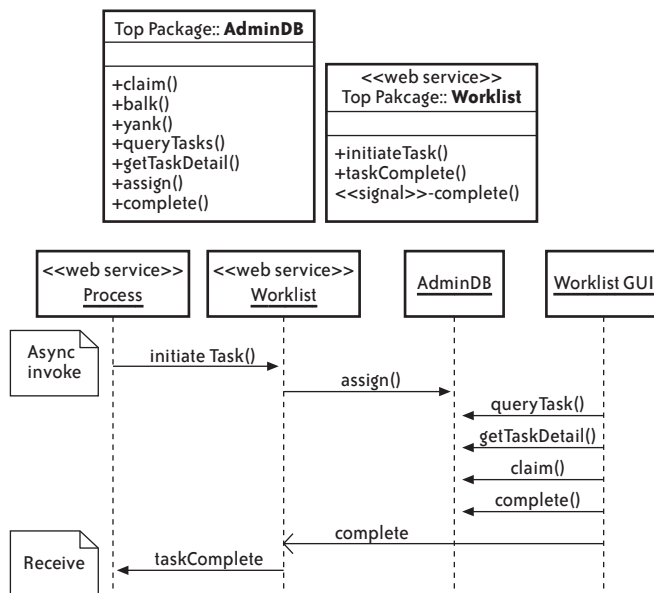


FIGURE 2-7. *Worklist interface*

The figure shows two classes: Worklist, stereotyped as a web service, and AdminDB (typically a set of stored procedures or a Java or C# data access object), representing the typical accesses to the database for task management. In the sequence diagram, when the process invokes the Worklist's `initiateTask()`, Worklist persists to the database, via `assign()`, a record of the initial assignment (e.g., `Resolve Error` is assigned to role `Agent`). The worklist console, when the user clicks on a worklist link, performs the query `queryTasks()` to draw a summary of tasks, and when the user selects one of the tasks, calls `getTaskDetail()` to show the full set of task data. The user then *claims* the task (more on this next) and completes it (`claim()` and `complete()`, respectively), at which point the console signals to the Worklist that the job is finished; the Worklist in turn invokes the `taskComplete()` callback, which awakens the BPEL process.*

NOTE

Several BPM products (e.g., IBM MQ Workflow) have worklist interfaces. Oracle's implementation stands out because it is designed as a web service callable from a BPEL process.

In most cases, from the perspective of the process, the identity of the person who actually performs a task is irrelevant. What is important is that person's *type* or *role*. Any travel agent, for example, can work a "resolve error" case. But in the end, a particular person does end up doing the work, and a good role-based security model is required to make it happen. "Bill Smith" is allowed to execute a "resolve error" task, for example, because he has the role of "travel agent." The security infrastructure leveraged by the BPM architecture should provide two role-based functions:

- The ability to lookup a user's roles. When Bill Smith logs into the console, the console should show him his current roles.
- Secure access control. The architecture should not allow a user who is NOT a travel agent to complete a "resolve error" task.

During its lifetime, a task can change hands several times: from role to user, from user to role, or from user to user within a role. In the simplest case, described in the previous scenario, a task is initially assigned to a role, and then claimed by a user having that role. Other possibilities are summarized in the following table.

TABLE 2-2. *User-role actions*

Type	Description
Assign	The task is tagged to either a particular user or to a role.
Claim	The task is tagged to a particular user in a role. In other words, a user interested in working the task shared with other users in the role, claims it.

* The Worklist design is inspired by the TaskManager component of the Oracle BPEL Process Manager. The use of the Oracle worklist is demonstrated in Chapter 10.

TABLE 2-2. *User-role actions (continued)*

Type	Description
Yank	A claimed task is moved to a different user in the role. In other words, a user confiscates a task from its claimed user. E.g., task is escalated.
Balk	A claimed task is untagged and assigned back to the role. In other words, the user decides not to work the task.

System Interaction

No process runs in isolation. In addition to its human dependencies, a process will invariably call or be called by various types of software components. There are four possible modes of interaction:

TABLE 2-3. *Interaction modes*

Mode	Description
Receive	Process receives a message from another system.
Receive-Respond	Process receives a message from another system and sends back a response message.
Send	Process sends a message to another system.
Send-Receive	Process sends a message to another system and waits for the response.

Components can be divided into those that are external and internal to the company. External interactions are web service-based and follow choreography or collaboration protocols. Internal software interactions can be inline code snippets or client/server interfaces to other systems running on the corporate network.

Adapters

The BPM architecture should aim to support the widest variety of system interfaces. A naïve design would be provide individual hooks for every conceivable technology: the MQ hook, the JDBC hook, the Tuxedo hook, the Java hook, the EJB hook, the COM hook, the web service hook, the JMS hook, and so on. A cleaner, more extensible approach is to provide built-in support for the typical process technologies such as web services and XML, inline code capabilities for Java or C#, and an adapter plug-in model for anything else. The adapter plug-in model requires the following:

- The adapter must be coded to a generic interface that the runtime engine understands. On a J2EE platform, for example, a particular EJB remote interface or a Java Connector Architecture (JCA, or J2C) adapter might be required.
- The runtime engine must provide a directory into which adapters can be registered. On J2EE, this might be the Java Naming and Directory Interface (JNDI).

In Figure 2-8, adapters for SAP, MQ, mainframe and database are deployed to the engine, sitting alongside natively supported XML, web service, and Java modules.

The Process Is a Web Service. In many contemporary BPM approaches, including BPEL, the process not only interacts with other web services, *it is itself a web service*. Every

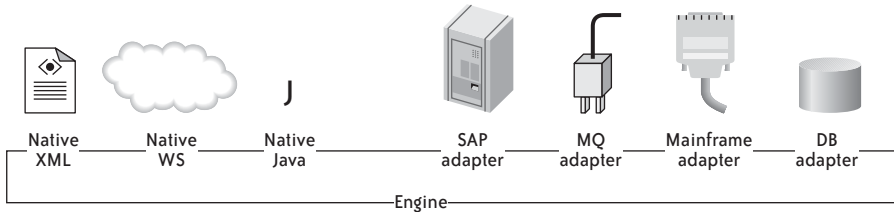


FIGURE 2-8. System interaction: native and adapter

“receive” node, in which the process listens for inbound messages, appears to the outside world as a web service operation. Calling that operation injects an event into the process and causes it to perform a burst of activity before either completing or waiting for the next event.

For BPM architects, this feature means that the runtime engine must include a special web services listener that knows how to accept an inbound Simple Object Access Protocol (SOAP) message, inject it into the engine, obtain the response, if any, and send it back out as a SOAP message. On a J2EE platform, because SOAP is typically transported over HTTP, this component could be an HTTP servlet that would handle inbound SOAP using HTTP POST and GET methods. The design is depicted in Figure 2-9.

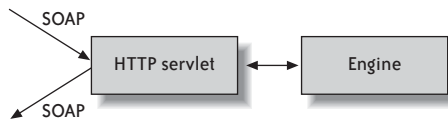


FIGURE 2-9. Servlet as web services listener for engine

Administration and Monitoring

The mark of a successful production application is a well-conceived and sophisticated set of administration and monitoring tools. The main tool is a graphical console that lets system administrators watch, tune, and perform management actions; in most cases, the console uses (under the covers) a standard management API to converse with the running system. A BPM console should allow operators to monitor running processes, kill hanging processes, install new process definitions, and decommissioning processes that are no longer required. Internally, the BPM application should support SNMP or JMX management infrastructures to facilitate these operational capabilities. With these interfaces, custom consoles can be developed to meet requirements that are not provided out of the box (e.g., sending an email to a specified group list when an error occurs while deactivating a process definition).

The first step in designing a system management interface for a particular system is to identify the “managed objects” and their services. Table 2-4 summarizes the BPM managed objects and required services.

TABLE 2-4. *Managed objects and required services*

Managed Objects	Required Services
Process definitions	- Find process definitions known to the engine. Possible filter: activated or deactivated. - Deploy: Add a process definition to the list. - Remove. - Activate: Switch on a process definition so that it can be instantiated. - Deactivate: Switch off a process definition so that it cannot be instantiated.
Processes	- Find processes known to the engine. Possible filters: process definition, start date, completion date, process variable values, state. - Suspend. - Resume. - Terminate.
Activities and worklist tasks	- Find activities and tasks known to the engine. Possible filters: type of role, parent process, start date, completion date, state, incoming or outgoing message values. - Suspend. - Resume. - Terminate.
Users and roles	- Add, modify or remove users from the application. - Add or remove roles from the application. - Add or remove users from roles.
Applications	Configure connections for applications (e.g., web services, database, MQ, J2EE) that can call or be called by processes. For example, deploy a servlet to listen for inbound web service calls, and create a JAX RPC connection to invoke external web services.

Figure 2-10 illustrates a typical BPM console that provides these services.

The screenshot shows a web browser window titled "Microsoft Internet Explorer" displaying the "BPM Administrative Console". The address bar shows the URL "C:\msk\haver\dev\personal\book\preilly\BPMAdmin.html". The console has a left sidebar with a menu: "Process Definitions", "Processes", "Activities", "Users and Role", and "Applications". The main content area is titled "Processes - Search" and contains a search form. The form has a "Definition:" field with the value "HandleClaim.bpel". Below it are "Start Date (YYYY-MM-DD)" and "Completion Date (YYYY-MM-DD)" fields, with the start date set to "2002-01-01". There are four "Variable:" sections, each with "Name:" and "Value:" fields. The first variable has "Name: claimID" and "Value: 12*". The other three variables have empty fields. At the bottom of the form are "Submit" and "Clear" buttons.

FIGURE 2-10. *Admin console: process search*

The left panel contains a set of links representing the core managed objects. When an administrator clicks on a link, the right panel shows administrative options for the selected object. For example, clicking on “Processes” brings up the panel “Processes – Search.” The search shown in the diagram will find all processes with definition “HandleClaim.bpel” that started on or after 2002-01-01 and have a variable called “claimID” that starts with the text “12.” The search is executed by clicking on the “Search” link. The list of matching processes is shown in Figure 2-11.

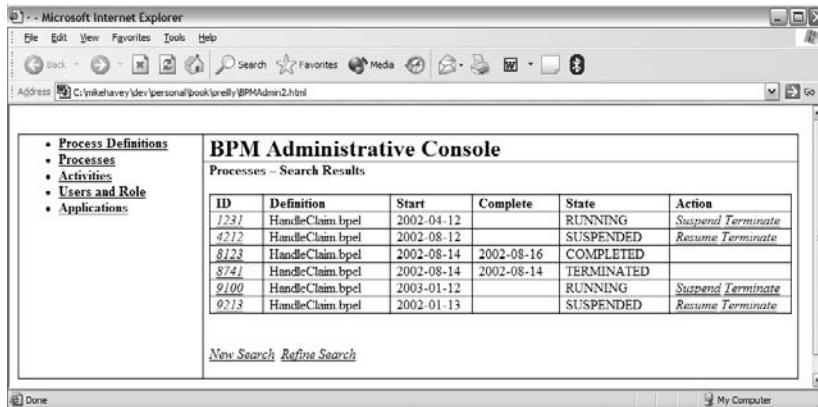


FIGURE 2-11. Admin console: process search results

The list provides links (the “ID” value) to a page that displays the full detail of the process. For convenience, the links under the “Action” column allow process maintenance without the need to delve into the details. The options listed in this column depend on the state. The first item (ID “1231”), whose state is “RUNNING,” can be suspended or terminated; the second item (ID “4212”), which is already “SUSPENDED,” can be resumed or terminated.

Persistence

Because most processes are stateful and must survive a restart of the runtime engine, the BPM implementation requires a persistent data store where process state is kept current in case of engine recovery. The data store is best modeled as a normalized relational database schema. Not only is this approach an accepted practice in contemporary enterprise architecture, but it also benefits customers by giving them the ability to perform ad hoc process queries. Customers might also build reporting databases or data warehouses based on the “runtime” process persistence model, and institute purge and replication jobs that, respectively, cleanup the runtime model and synchronize the reporting model.

Figure 2-12 shows a good BPM runtime data model.

The tables in the model include the following:

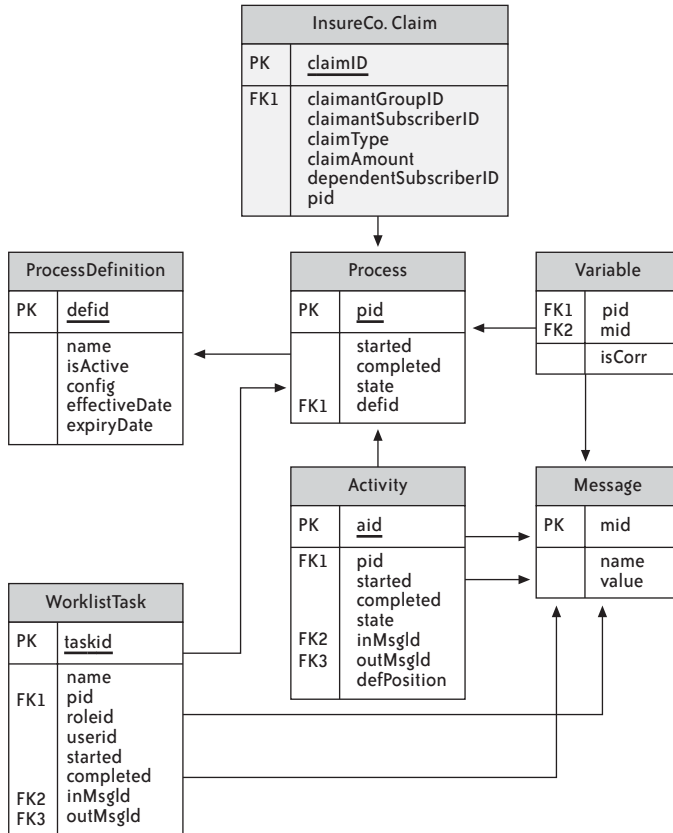


FIGURE 2-12. Process data model

ProcessDefinition

The process definition keeps a list of definitions deployed and known to the engine. Each definition maps a unique numeric id (*defId*) to a descriptive name (*name*), which (for simplicity) is the name of the XML process definition file (e.g., *processClaim.bpel*). The optional field *effectiveDate* specifies the date and time when instances can begin to be created for the definition; if it is left blank, instances can be created any time. Similarly, *expiryDate* specifies the date and time after which no instances can be created; if it is left blank, the definition never expires. *isActive* field is a Boolean (actually ‘Y’ or ‘N’) that, when set to ‘N’ prevents the creation of new instances, regardless of the effective and expiry dates. The field *config* holds deployment settings such as the URLs of the WSDLs of the process and of each of its partners.

Process

A process, which is a running *instance* of a process definition, is represented by a unique numeric identifier called a “pid” (probably based on a database sequence), and has fields representing its “state” (e.g., RUNNING, SUSPENDED, COMPLETED, ABORTED), timestamps for its start and completion (“started” and “completed,” respectively), and a pointer to its process definition (“defId”).

Message

A message is simply a piece of data with a name and a value. The value is often an entire XML document or a piece of it. A message is identified by a unique numeric field called a “mid.”

Variable

A process can have zero or more variables. The table Variable, which has a many-to-one relationship to Process, holds the variable values for a given instance. Variable joins to Process on the field “pid.” The field “mid” references a record in Message, where the actual name and value are held. Finally isCorr is a Boolean (actually ‘Y’ or ‘N’) that indicates whether the variable is a correlating field.

Activity

An activity is a step in a process. Its numeric database ID “aid” maps to defPosition, which for simplicity is the name of the activity found in the definition (e.g., adjustClaim). Activity has a many-to-one relationship to Process, joining it on its “pid” field. The fields started and completed represent the activity’s start and completion times, respectively. The field state represents current progress (e.g., STARTED, ASSIGNED_TO_ROLE, WAITING_MSG, RUNNING, COMPLETED, ABORTED). An activity can have an input message and an output message, represented by isMsgId and outMsgId respectively, each of which references the Message table.

WorklistTask

A worklist task is similar in structure to an activity, having a process reference (“pid”), inbound and outbound messages (inMsg and outMsg, respectively), and start and completion dates. But, as mentioned earlier, the lifetime of a task spans several process activities (in BPEL, the entire sequence from “invoke” to “receive”), and a task is assigned to a particular role (roleId) or user (userid). Despite similarities to Activity, WorklistTask merits its own table.

Figure 2-13 also demonstrates how the core model can be extended with application-specific data. The shaded table InsureCo.Claim represents an insurance company’s representation of a claim. The field pid links the claim to a particular record of the Process table, which represents a specific process instance. The mapping in this case is one-to-one: each claim is handled by one instance of claims business process. Table 2-5 describes some other possible scenarios; significantly, no changes are required to the core tables.

TABLE 2-5. *Modeling process-to-claim relationships.*

Scenario	Implementation
One claim to one process	Claim table has pid field linking to process table.
Multiple claims are handled by one process	Same.
One claim is handled by multiple processes	Create a table called ClaimProcess with two fields: claimID, which links to the claim table, and pid, which links to the process table.
Multiple claims are handled by multiple processes	Same.

The following examples, based on an Oracle database implementation, illustrate several SQL statements that query the data model. These queries could be used to generate reports or administer the system :

- List suspended processes based on definition `purchaseOrder.bpel` that were started before September 1, 2004

```
Select p.pid from process p, processdefinition d
where p.defid=d.defid and d.name='purchaseOrder.bpel'
and p.state='SUSPENDED' and p.started<to_date('2004-09-01','YYYY-MM-DD');
```

- Find the process based on definition `orderTicket.bpel` whose correlation variable `ticketNumber` has value `A12345`

```
Select p.pid from process p, processdefinition d, variable v, message m
where p.defid=d.defid and d.name='orderTicket.bpel' and
v.pid=p.pid and v.isCorr='Y' and v.mid=m.mid and m.name='ticketNumber'
and m.value is not null and m.value='A12345';
```

- List all the activities and their input and output variables, of the process with ID 1012

```
Select a.defposition, a.started, a.completed, a.state, inmsg.name, inmsg.value,
ormsg.name, ormsg.value
from activity a, message inmsg, message ormsg
where a.inmsgid=inmsg.mid(+) and a.outmsgid=ormsg.mid(+) and a.pid=1012;
```

- (Using the custom data extension) List all claims in the order they were started

```
Select c.* from insureco.claim c, process p where c.pid=p.pid
order by p.started desc;
```

BAM and process mining

Business activity monitoring (BAM) is an exalted form of process administration and monitoring offered by vendors such as Pegasystems. BAM is targeted at the business manager who requires a sophisticated dashboard with graphical views (often in the form of graphs or charts) of process-related data updated in real time or generated as part of a report. This data can include the state of running processes and their activities, as well as aggregated statistics concerning the business subject matter of processes (e.g., if the process is for loans, view the number of approved loans so far today). Figure 2-13 illustrates a typical dashboard.[†]

BAM can also let the manager make changes in response to emerging conditions. For example, if a given process appears to be stuck waiting for the action of a particular person, the manager can reassign it to someone else. Some BAM implementations can even run on auto-pilot; rules can be defined to automatically perform a specific action when a particular event occurs or a particular condition takes effect (e.g., sending an email to the bank manager when a high-value customer closes an account).

[†] Pegasystems, “PegaRules Process Commander: Business Activity Monitor: Empowering BPM,” Pegasystems white paper, 2003, <http://www.pegasystems.com/downloads>.

PROCESS VERSIONING BASICS

Over time, the definition of a process needs to change to accommodate bug fixes, enhancements, and new public interfaces. But in a production environment, applying these changes to definitions that have active and possibly long-running instances requires exquisite care. The trick is to make the change without compromising the current base.

Depending on the nature of the change, use one of the following strategies:

- If the change is minor, such as a fix or enhancement to a particular activity (e.g., fixing an XPath expression in a data transformation, or switching to a new WSDL port for a process invocation), replace the old code with the new code. To prevent access to instances while this is happening, either shut down the system or queue client requests.
- If the change is major but does not alter the public service interface (e.g., a change of control flow in the internal logic of the process), first, keep the old process definition, but deactivate it by switching its `isActive` flag to 'N.' Then create a new process definition, with a unique `defId`, `isActive` set to 'Y,' and name containing an incremental version (e.g., if the previous version was `processClaim.bpel`, name the new one `processClaim_v2.bpel`). Give the `config` field the same values as in the previous version. The effect is that clients do not change the way they interact with the process, but internally the system supports two versions— one for old instances, and one for new ones.
- If the change is major and represents a new public service interface (e.g., new XML format and acknowledgments now mandatory), follow the same procedure, but modify the settings in the “`config`” field of the new definition: point to a new version of the WSDL that has the updated public interface. The effect is that clients use the new interface to start new instances, and the old one only to finish old instances.

In addition to providing such a rich perspective on the state of the application, BAM—by showing the most commonly traversed paths through a process and by isolating inefficient steps—also helps process engineers and business operations people spot continuous improvements. The dashboard might reveal, for example, that a particular process path has not been run for the past five years, suggesting a redesign in which the path is eliminated and the staff that participates in it reduced or reassigned.

The curious science of *process mining* formalizes this idea, specifying algorithms to discover the behavior of a process from the ordering of events in its logs. For example, if the logs show that, for a given process, activity A is always followed by B and then, in arbitrary order, C and D, we can guess that the process is designed such that A transitions sequentially to B, at which point it forks into two parallel paths headed by C and D, as shown in Figure 2-14.

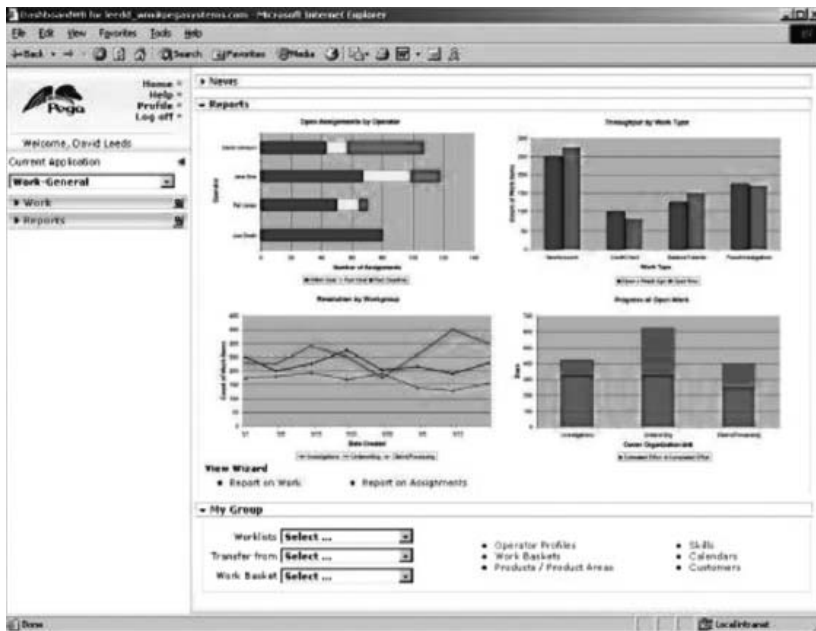


FIGURE 2-13. Pegasystems BAM dashboard

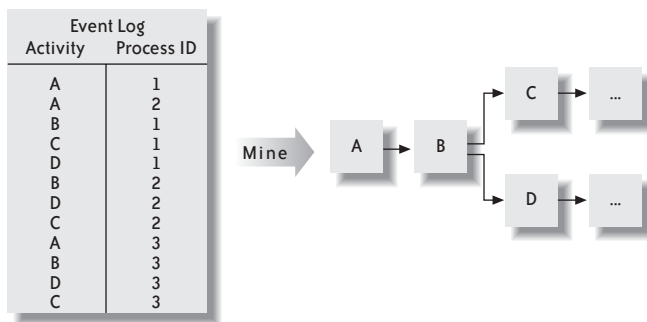


FIGURE 2-14. Process mining: guessing process flow from the event log

The ability to watch processes unfold “in the wild” is one of BAM’s key benefits. Most processes are designed “normatively,” claim the authors of the widely read paper “Workflow Mining: A Survey of Issues and Approaches.”[‡] In other words, they are designed to behave as they *should* behave. But when we watch them run, we discover their *actual* behavior. In effect, by working backwards from the empirical runtime data, we can discover a process design that is potentially better than the one that is currently deployed.

[‡] W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. J. M. M. Weijsters, “Workflow Mining: A Survey of Issues and Approaches,” *Data and Knowledge Engineering*, Volume 47, Issue 2 (Nov 2003),

Local View of Choreography: WS-CDL Toolkit

The most fanciful and speculative component of the architecture is the WS-CDL toolkit, which provides two tools:

The generator

Generates a process model having the required the flow of events and invocations for one participant in a multi-participant choreography.

The validator

Validates a local process model against a multi-participant choreography.

There are several problematic aspects of the architecture to consider:

- No implementation of such a toolkit is available today, though many discussions of WS-CDL highlight the value of a WS-CDL-to-BPEL generator.
- In our architecture, BPMN is a more suitable generation target language than BPEL; BPMN is our design language, and BPEL is generated from it by a separate export tool.
- BPMN is purely graphical, and cannot be generated in a standard way by an automated tool. To generate BPMN code for ITpearls's Visio-based Process Modeler tool, for example, would require understanding, and coding to, the Visio object model. BPMN has no behind-the-scenes open XML representation that the tool can use.

The good news is that BPMN is a sufficiently expressive language to capture the local participant view of a choreography. Further, BPMN's published mapping to BPEL adequately covers the key BPEL participant-related activity types: receive and invoke. In short, WS-CDL-to-BPMN-to-BPEL, when applied to choreography, is conceptually viable, although the first leg of that trip is potentially an uncomfortable ride.

How would the generator work? Taking as input the choreography and the identity of the required participant, the generator would scan through the choreography looking for interactions involving that participant. For each interaction, the tool would generate an activity in the local process. The tool would be required to handle various control constructs (sequential, conditional, parallel) and to detect and encode message correlation (e.g., correlate—a response and an acknowledgement with an earlier request).

Consider the following informally stated choreography:

```
A sends request to B
In parallel, B forwards request and then waits for an ack and then a response from,
respectively, C and D
B combines the two responses and sends to A
```

The local process for B would resemble the following:

```
Receive request from A
Do a parallel split
  Path 1
    Send request to C
    Receive correlated ack from C
```

```

    Receive correlated response from C
Path 2
    Send request to D
    Receive correlated ack from D
    Receive correlated response from D
Combine responses from C and D
Send combined response to A

```

As for the validator, taking as input the choreography, the identity of a participant, and the code of the participant's local process, the validator would be required to check whether the process includes the correct activities in the proper order and does not miss anything. The following code for participant C is invalid, because it does not include the step to send an acknowledgment:

```

Receive request from B
Send response to B

```

Standards

Table 2-6 lists the major BPM standards and where they fall within our model architecture.

TABLE 2-6. *BPM standards: where they fit*

Name	Organization	Type	Chapter in This Book
Workflow Reference Model	WfMC	Architectural Model	Chapter 7
Business Process Modeling Notation (BPMN)	BPML	Notation Language	Chapter 6
UML Activity Diagram	OMG	Notation Language	Chapter 3
Business Process Execution Language (BPEL)	OASIS	Execution Language	Chapter 5
Business Process Modeling Language (BPML)	BPML	Execution Language	Chapter 6
XML Process Definition Language (XPDL)	WfMC	Execution Language	Chapter 7
XLANG	Microsoft	Execution Language	Chapter 9
Web Services Flow Language (WSFL)	IBM	Execution Language	Chapter 9
Business Process Definition Metamodel (BPDM)	OMG	Execution Language and/or Notation Language	Chapter 9
Business Process Runtime Interface (BPRI)	OMG	Administration and Monitoring Human Interaction System Interaction	Chapter 9
Workflow API (WAPI)	WfMC	Administration and Monitoring Human Interaction System Interaction	Chapter 7
Business Process Query Language (BPQL)	BPML	Administration and monitoring	Mentioned in Chapter 7
Web Services Choreography Interface (WSCCI)	W3C	Choreography	Chapter 8

TABLE 2-6. *BPM standards: where they fit (continued)*

Name	Organization	Type	Chapter in This Book
Web Services Choreography Description Language (WS-CDL)	W3C	Choreography	Chapter 8
Web Services Conversation Language (WSCL)	W3C	Choreography	Chapter 8
Workflow XML (WfXML)	WfMC	Choreography	Chapter 7
Business Process Schema Specification (BPSS)	OASIS	Choreography (and collaboration)	Chapter 9

At seventeen entries, a first glance at the list is intimidating, but the number of viable candidates for our architecture is small:

- Both of the major notation languages, BPMN and UML’s activity diagrams, are good, but BPMN is preferred because it is more expressive and defines a BPEL mapping.
- Of the five XML execution languages, BPEL wins by a landslide, because it is the most expressive, has a BPMN mapping, and enjoys the widest industry support.
- None of the three approaches covering administration and monitoring, or human and system interaction, is suitable for our architecture. BPQL and BPRI are in infancy and need time to develop; WAPI, by contrast, is past its prime and at best has a few ideas worth harvesting.
- Choreography is used in our architecture to generate skeletal local process models having the correct logic for communication with external participants. WS-CDL is the best of the five choreography standards, and is the recommended approach of the W3C’s web services choreography working group.
- The WfMC’s workflow reference model in general terms is similar to, and an inspiration for, our architecture, but its details are too closely aligned with the outdated WfMC WAPI, XPDL, and WfXML.
- The OMG’s BPD process metamodel is under development. When it is released, BPD will standardize the mapping between processes designed in BPMN, but run on BPEL. The BPMN design tool will export its processes as BPD models, and the BPEL runtime engine will import them in that form.

From this analysis, only three of the seventeen standards—BPMN, BPEL, and WS-CDL—currently fit into our architecture, but BPQL and BPD look promising for a future iteration.

Summary

The main points of this chapter include the following:

- BPM is replete with competing standards, but a sound architectural approach divides a BPM application into the right parts and selects the correct standard for each.

- The requirement of a BPM application is the ability to design, run, administer, and monitor processes that incorporate system and human interactions.
- Two types of architects benefit from this discussion. Product architects learn, at a high level, a good approach to developing a BPM product. Services architects learn the essential nature of a BPM product platform, enabling them to build good solutions on it and to educate their customers about this emerging technology.
- Design, the modeling of processes by business and technical analysts, requires a graphical notation language and a graphical design tool, like ITpearls's Process Modeler. BPMN is the best standard notation language.
- To run a process requires a runtime engine that can load designs and manage the execution of instances. But there is no way to "run" a notational drawing; it is just a design artifact. A runtime engine requires an executable language, just as a computer processor requires programs that use the processor's instruction set. To bridge this gap, a mapping from notational to executable language is needed. The design tool should offer an export option that uses the mapping to generate executable code. With respect to standards, BPEL is the best and most widely adopted executable language, and our chosen notational language, BPMN, includes as part of its specification a detailed mapping to BPEL.
- Administration and monitoring is crucial for the success of a production application. An administrative console, along with a sufficient data model and a standard system management interface language, helps administrators track and change the managed objects of the runtime engine, including process definitions, processes, activities, manual tasks (worklist items), and users and roles.
- Human interactions are manual work items that are performed by particular users or users with a specific role within the organization. The process initiates this work and waits for an event for the work's completion. The users view and execute their pending tasks in a graphical worklist console. Because tasks are long-lived, their state is kept in the administration database. A special Worklist web service manages human interactions on behalf of the process.
- System interactions are connections to internal or external systems. External systems are typically web-service-based partner processes. Internal systems are other applications on the corporate network with which the process needs to interface. The number of integration technologies is high, and it would be foolhardy to try to support each of them individually. The best approach is to offer the most common technologies natively (XML, web services, inline Java) and provide an adapter plug-in model for the rest (e.g., MQ, database, or mainframe).
- A process is a web service! Its event points are, in effect, service operations. For this reason, the BPM integration architecture must include a special web services listener that accepts SOAP requests and injects them into the process engine.
- The processes of this architecture are those of a particular corporation and are designed from a local perspective. The global perspective of choreography and collaboration (dis-

cussed in Chapters 8 and 9 and Chapters 8 and 9) helps build protocols with which our company must comply.

- Of the seventeen major BPM standards, only three—BPMN, BPEL, and WS-CDL—belong in our architecture. The emerging BPQL and BPDM look promising for a future iteration.

References

1. Oracle, “BPEL Tutorial 6: Working with the TaskManager Service,” <http://otn.oracle.com/bpel>.
2. ActiveBPEL, <http://www.activebpel.org>.
3. Pegasystems, “PegaRules Process Commander: Business Activity Monitor: Empowering BPM,” Pegasystems white paper, 2003, <http://www.pegasystems.com/downloads>.
4. W. M. P. van der Aalst, B. F. van Dongen, J. Herbst, L. Maruster, G. Schimm, A. J. M. M. Weijsters, “Workflow Mining: A Survey of Issues and Approaches,” *Data and Knowledge Engineering*, Volume 47, Issue 2 (Nov 2003),