2 Defining the Open SOA Platform

This chapter covers

- Evaluating open source products
- Selecting the products

In chapter 1 we explored some of the history behind SOA, and then we examined the key technology underpinnings of a SOA environment. Now we'll focus on identifying a suitable open source product for each of these technology areas. Collectively they'll comprise what we're calling the *Open SOA Platform*.

The open source community includes many early advocates of the recent wave of emerging SOA-related technology projects. Historically, open source has sometimes been considered a "late-follower," with commercial products first to hit the market, and then followed by "me-too" open source alternatives. One reason often cited by critics of open source is that open source projects are often not innovators but imitators (of course, some might argue Microsoft has done very well by following the imitation model). There may be some truth to that criticism, but many of the products we'll be examining are innovative and cutting edge. In some instances, the reason development has lagged vis-à-vis commercial offerings is simply because of resource challenges—open source projects are often supported and staffed by a small team of developers, many of whom have full-time responsibilities elsewhere.

Overall, it did take some time before a comprehensive collection of open source projects achieved sufficient breadth and maturity to offer a compelling alternative to the highly priced commercial alternatives. Now, you can choose among many options for crafting an entirely open source SOA environment. This chapter forms the basis for the remainder of the book—it identifies the open source products that we'll be exploring in greater detail in the chapters that follow. The selected products will form the basis for our Open SOA Platform, and we'll illustrate how these products can be integrated together in a coherent fashion so that, combined, they'll equal or surpass in value the offerings by the commercial SOA vendors. Figure 2.1 recaps the technologies involved in the Open SOA Platform and highlights (in double-width lines) those that we'll investigate moving forward (as you recall, JMS, application servers, and GUIs are covered thoroughly by other publications or are fairly commoditized in functionality).

Over the past five years I've had the opportunity to participate in "real-life" projects that have used many of the open source products discussed in this chapter. In choosing which ones will constitute our Open SOA Platform, I had to select a single product within each product category. This isn't intended to suggest those that aren't selected are any less worthy. As with any evaluation process, the final choice is based on some combination of objective and subjective facts (obviously, we all want to believe we only use objective facts, but human nature often dictates against such logic).

Before we dive into each of the technology categories and the open source possibilities within each of them, let's first establish some general, universal criteria that we can use when evaluating any of the products.

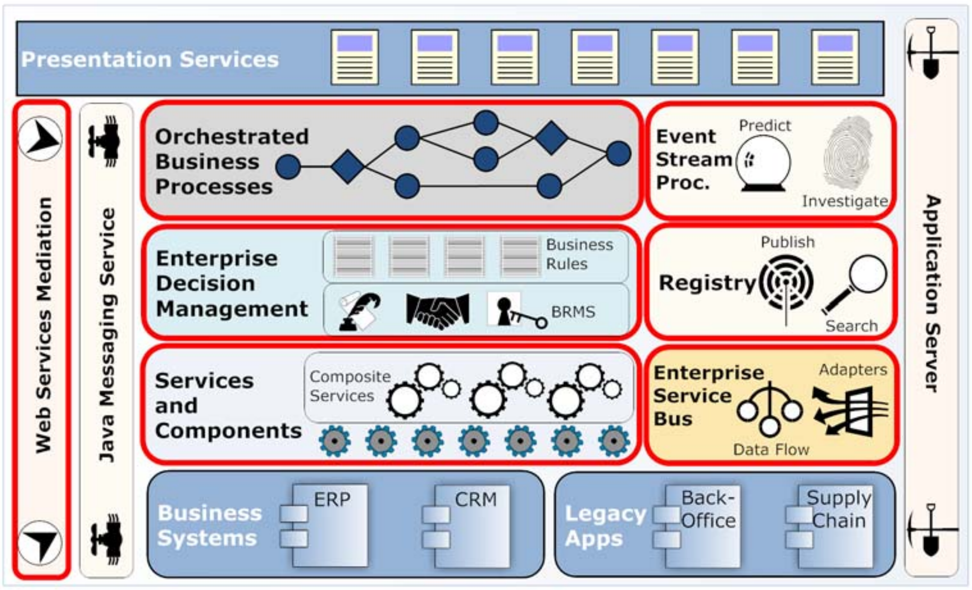

Figure 2.1 Open SOA Platform technologies. Those surrounded in double-width lines represent what's covered in this book.

2.1 Evaluating open source products

Some general criteria exist for examining all of the technology products that constitute the Open SOA Platform. They're described in table 2.1.

Table 2.1 Open source selection criteria, general guidelines

Criteria	Comments
Viability	Is the product widely used, and does it enjoy a strong user community? Is the solution well documented? Are sufficient development resources committed to the project?
Architecture	Is the architecture of the product complementary to the other products we are evaluating? Is it well documented and logical, and does it adhere to common best practices and patterns?
Monitoring and management	Does the product provide off-the-shelf monitoring and management tools? Since we are mostly evaluating Java products, does it utilize JMX, which is the standard for instrumentation and monitoring of Java applications?
Extensibility	Can the off-the-shelf solution be extended to add new functionality? Does a pluggable framework exist for adding new functionality?
"True" open source	This is a sensitive topic, but we want to consider only products that are licensed using one of the common open source licenses: GPL, LGPL, BSD, Apache, or Mozilla Public License. We want to avoid, if possible, "free" or "community" versions that retain restrictions in usage or modification.

Now that we've identified the general evaluation criteria that we can apply to evaluating the technologies that constitute the Open SOA Platform, let's look at each technology category and identify for each an open source product that we'll use. In this process, we'll identify competing open source solutions and address (a) the criteria used for evaluating the products within a category, and (b) the justification for why a given product was chosen. Let's start with BPM.

2.2 Choosing a BPM solution

As we discussed in chapter 1, BPM refers to software that can be used to model and execute workflow processes. BPM can be considered another form of application development, albeit more visual in nature. Also, design and early development of BPM processes can often be performed by subject matter experts instead of hardcore developers (that said, the latter is often still required, at least in later stages of the development cycle). Why is BPM considered part of SOA? It is because it directly benefits, and is enabled by, the exposing of reusable services that is central to SOA. With BPM, you can create business processes that span across multiple, previously stovepiped, applications. In this sense, BPM applications are often fundamentally different from traditional applications, and are less focused on performing a specific task and more oriented toward an entire business process. For example, a new hire process within a company may involve setting up the individual in a multitude of different systems, from benefits and payroll to a 401(k) system. A BPM process models that entire workflow and isn't isolated to populating just one of the applications with data.

What is a "stovepiped" application?

A *stovepiped* application, by its design, is completely isolated and self-contained. Legacy applications, which were often developed with little notion of integrating with external data or systems, are often considered stovepiped. SOA tools provide the ability to unlock the business rules and operations of these stovepiped applications into services that can be invoked externally. Existing investments can be leveraged without having to resort to replacing critical business systems.

It's worthwhile to distinguish between some of the terms used in BPM, as the terminology can sometimes be rather confusing:

- A *workflow* is generally understood as series of human and/or automated tasks that are performed to produce a desired outcome. A fancier name for workflow that is commonly used is *orchestration*.
- Closely related to workflow is a *process*. It's defined as "a set of activities and transactions that an organization conducts on a regular basis in order to achieve its objectives… It can exist within a single department, run throughout the entire enterprise, or extend across the whole value chain" [BPMBasics]. A process may involve one or more workflows.
- A *task* represents a specific work item that must be performed, most typically by a user. Tasks constitute the work within the workflow.
- A *node* is a generic command or step within a process. It can be a task, a wait state, or a decision. A business process consists of nodes.
- A *transition* (or, in XML Process Definition Language [XPDL] nomenclature, *edge*) defines how nodes are connected.

BPM systems, by their nature, involve modeling what can be complex processes. Mathematical algorithms are often used as the basis for implementation and can be fairly arcane to understand for those not steeped in its principles. The requirement to visually model workflows also represents a significant development challenge. These are perhaps the reasons why open source BPM solutions were, at first, slow to emerge. Recently that has changed, and you can now choose among several excellent open source BPM systems. We'll discuss how to make a wise choice in the next section.

2.2.1 BPM product evaluation criteria

As you recall, in section 2.1 we discussed general criteria for evaluating open source SOA software. There are obviously some additional BPM-specific criteria that we'll want to consider; they are listed in table 2.2.

What's the difference between BPM and BPEL?

BPEL (Business Process Execution Language) can be considered a subset of BPM. BPEL provides a semantically rich language for creating business processes that are composed of SOAP-based web services. It's a specification for how to materialize a business process that's composed of SOAP-based web services. BPEL, by itself, has no specific provisions for human activity–based tasks or queues (though the emerging BPEL4People—WS-BPEL Extension for People—will address some of these deficiencies), which are typically associated with workflow-based BPM processes. The BPEL standard also doesn't specifically address reporting, analysis, or monitoring, though some BPEL vendors have augmented their offerings to include such features. In other words, the term BPM is typically used when referring to complete product offerings whereas BPEL is typically used to refer to the web service orchestration standard.

Obviously, this only scratches the surface of the underlying functionality typical in any BPM solution. However, it does touch on some of the most important features and provides us with guidance on identifying what constitutes a BPM. That way, we can identify possible open source products, which is our next topic.

Table 2.2 BPM evaluation criteria

Criteria	Comments
Simplicity	BPM solutions, particularly those by commercial vendors, have a history of being very complicated to learn and even more difficult to deploy. Circumstantial evidence suggests many solutions become expensive "shelfware" and never live up to the promises anticipated. We want our solution to be simple to learn, deploy, and manage.
Lightweight/embeddable	In part related to simplicity, this criterion refers to the ability, if need be, to incorporate the BPM "engine" directly into an application. For example, you might be building a new loan processing application and want the ability to embed a workflow engine directly within it without having to manage it externally.
Process nodes	Are all standard process nodes available out of the box? This would include decision/conditional routing, human-interface task support, forks/splits, and joins/merges. Can callout nodes or capabilities exist to invoke Java and web services?
Transactional requirements	Do auditing, logging, and rollback/compensation features exist? Are long-running transactions supported? Are roles and users supported?

2.2.2 Open source BPM products

As we pointed out earlier, BPM solutions tend to be fairly complex in nature. This is both because of the visual modeling requirements and the complex workflow algorithms that drive the BPM engine. Fortunately, within the past few years, we've seen exciting developments in the open source community surrounding BPM, and there are now several excellent products to choose from. Table 2.3 lists the most active BPM open source products available today.

Table 2.3 BPM open source product overview

Product	Comments
Intalio BPMS (Community Edition)	Feature-rich BPM that uses business process modeling notion (BPMN) to generate BPEL-based orchestrations. Unfortunately, only parts of Intalio's solution are open source, with some confusing licensing restrictions. Also, since BPMN is converted to BPEL (with that code being proprietary), extending the product seems problematic, and reliance on BPEL means support for only SOAP-based web services.
ActiveBPEL Engine	An efficient and highly regarded BPEL engine. Models can be designed using the free, but not open source, Designer. Important functionality such as persisting process instances to a database, or versioning of processes, is only supported out of the box in the commercial Enterprise version. My experience using the product suggests that the open source release isn't suitable for production usage.
Apache ODE	Apache ODE (Orchestration Director Engine) is a runtime BPEL engine. Its API is such that you can extend it in new and interesting ways, and thus aren't tied to the SOAP-only invocation of BPEL. The licensing model is very attractive, and the engine is lightweight and can be exposed, via Java Business Integration (JBI), to ServiceMix, an excellent open source ESB, which we cover later. Apache ODE doesn't come with a designer per se, but you can use the beta of the Eclipse BPEL editor.
Enhydra Shark and Java Workflow Editor (JaWe)	Shark is a workflow engine that adheres to the XPDL workflow standard that's supported by the Workforce Management Coalition (WfMC). JaWe is an XPDL editor, but has some limitations compared with its commercial cousin, Together Workflow Editor. Documentation specific to Shark was difficult to locate, and the emphasis, like with Intalio and ActiveBPEL, is to push you toward commercial products.
JBoss jBPM	A mature, efficient, and lightweight process/workflow engine with a usable Eclipse-based modeler. Uses its own terse XML graph notation language known as jPDL (jBPM Process Definition Language), and includes support for all core modeling nodes, such as decision and fork. Can be easily extended and isn't tied to a particular deployment framework. Unlike several others, there is no commercial "upgrade," and no functionality is specifically excluded.
ObjectWeb Bonita	Powerful, XPDL-compliant workflow engine. Well documented and mature. Includes excellent human-task UI integration (i.e., form generator). Doesn't come with an open source editor, and requires the JOnAS (Java Open Application Server) application server.
WSO2 Business Process Server	The WSO2 Business Process Server is based upon Apache ODE, and adds a web-based administrative interface along with simulation capabilities.

While the overview in table 2.3 doesn't delve deeply into the feature sets of each available solution, the criteria we established does point to Apache ODE, JBoss jBPM, or Bonita as the most appealing of the solutions. We'll address the reasons for this next.

2.2.3 Selecting a BPM solution

For several of the products listed in table 2.3, licensing issues were a major consideration in their exclusion from further consideration. In the case of Intalio, only some portions of their product are truly open source. With several others, the open source releases are more of a teaser to encourage upgrading to a commercial product (Shark/JaWe, ActiveBPEL). While Apache ODE can be fairly easily extended, it doesn't come with any built-in support for human-interface tasks, which (though not a part of the core BPEL standard) are an essential part of a BPM. Also, given that it's a BPEL execution engine, it's limited to working with SOAP-based web services, and can't, for example, directly invoke a Java class or populate a JMS message (granted, you could extend it to support this, but then it's no longer truly supporting the BPEL standard). For these reasons, we didn't select ODE, or WSO2's Business Process Server, which is based on ODE, as the BPM product.

ObjectWeb's Bonita offers an attractive open source solution. It has a proven heritage dating back to its 1.0 release, and with the release of Version 2, added support for XPDL. Unfortunately, Bonita doesn't come with an XPDL editor. Instead, Bonita suggests using one of the available open source or commercial editors. This raises a concern, as the open source XPDL editors don't appear to be sufficiently robust (at least compared with their commercial alternatives). An additional concern is the newer version's reliance on the JOnAS application server. This will limit the ability to embed the engine within other applications. Because of these reasons, we didn't consider Bonita moving forward.

This leaves JBoss jBPM. It's a simple-to-use, but very powerful, workflow engine. As mentioned, jPDL is the XML vocabulary used to express business processes, and they can be created visually using the jPDL Designer, an Eclipse-based plug-in. Further, centralized administration of jBPM processes can be managed through the jBPM Console, which is a web-based management tool. jBPM has the financial backing of JBoss and enjoys a fairly vibrant user community, based on forum and mail list activity. It also is being extended to support those who want to use BPEL scripts for workflow execution (at its core, it's a graph-based process engine). For these reasons, we selected it as the BPM solution for our SOA technology stack. Let's take a more in-depth look at jBPM.

2.2.4 Introducing JBoss jBPM

The jBPM project's first official release was in early 2004, followed by the 2.0 release later in the year. At approximately the same time, the jBPM team merged with JBoss, officially making jBPM a part of the JBoss family of products. Since the merger, the product has been managed and led by largely the same team, which has resulted in a solid, robust, and time-tested product. At the time of this writing, the 3.3 release of jBPM was the latest production version, with 4.0 in early alpha (we didn't use the 4.0 release for this book as it remains very fluid).

JBoss describes jBPM as "a flexible, extensible framework for process languages," or alternatively as a "platform for graph-based languages." The jBPM Process Definition Language (jPDL) was the first, or "native," process language developed on this framework. jBPM comes with the jPDL Eclipse plug-in Designer for easily creating business processes, along with a web application page-flow framework for creating human-based tasks. It supports persistence of process instances by storing them within nearly any open source or commercial database (using the well-respected Hibernate object-relational database mapping framework). Chapters 5, 6, and 7 will delve into great detail on jBPM.

2.3 Choosing an enterprise decision management solution

Enterprise decision management (EDM) is an approach to automating and improving the decisions a business makes on a day-to-day basis. It plays an important role in our Open SOA Platform, as it provides the centralized management for all of the business rules and logic associated with each of the applications.

Fundamentally, an EDM is about extracting the decisions and rules that are today embedded into applications or people and systematically exposing them as rule assets that can be centrally managed and authored. Some have gone so far as to proclaim a "Business Rule Revolution" is under way, insofar as it "represents an emerging undeniable need for the right people to know what a business's rules are, to be able to change those rules on demand according to changing objectives, to be accountable for those rules, and to predict, as closely as possible, the impact of rule changes on the business, its customers, its partners, its competition, and its regulators" [VonHalleGoldberg].

What's the difference between BRMS and EDM ?

EDM, besides sounding a bit sexier and less boring than *Business Rule Management System* (BRMS), is also considered to be a superset of BRMS. By that, it also includes leveraging analytical models that can be derived from data warehouse or business intelligence capabilities to conceivably create self-tuning rulesets. The reason we chose EDM for this book was that EDM is becoming the more recognized acronym for rule-based systems. Consider it similar to how *workflow* slowly became subsumed by the more *glitzy sounding business process management* (after all, workflow does sound pretty dry).

The value of managing business rules in a centralized fashion, and making them maintainable by business users instead of developers, has long been recognized as a laudable goal. Unfortunately, tapping into those rules from external applications and processes was often a considerable challenge. Early business rule vendors had their own proprietary API, often in one or two supported languages. This made integrating the business rules difficult and ensured vendor lock-in. The advent of web services and SOA opened up a vast new opportunity for incorporating a BRMS. Since web services are designed to be language and platform neutral, centralized business rules can now be accessed by virtually any application. Further, composite applications, such as business processes designed using a BPM, can easily tap into a BRMS for decision-based content routing rules. Perhaps the hyperbole of a "Business Rules Revolution" isn't such an exaggeration after all. In this case, the foundations of SOA become an enabling force to this exciting, even enterprise-changing, technology.

Figure 2.2 depicts the main components of an EDM.

In figure 2.2, we see a repository of rules broadly categorized according to the types of rules they are, such as "Constraint Rules," which serves, for instance, to impose limits such as the maximum amount of credit to extend to a customer. These various types of rules constitute the rule repository, which obviously has a central role in a rules system. The *Rule Engine* component, sometimes referred to as the *inference* or *execution* engine, represents the algorithms and logic that define how the engine works. The *API/Web Service* layer defines how you interface with the system. Many EDMs include multiple language-specific libraries and APIs, and often a SOAP- or REST-based web service interface. The *Authoring IDE* is the tool for writing, editing, testing, and categorizing rules. An important aspect of the authoring environment is whether support for domain-specific languages (DSLs) is available. This refers to the ability to express rules in a language that's natural to the business user yet has rigorous semantics. Consider it analogous to building your own programming language using a business vocabulary (hence, it's sometimes referred to as "language-oriented programming"). The *External Apps* are those applications that are utilizing the rules engine.

What's the role of EDM in SOA? One of the principal tenets of SOA is designing systems that are flexible and agile. Rules engines are instrumental in advancing this concept, as they allow business rules to be changed independently of making application modifications. This effectively eliminates having to go through drawn-out development and testing cycles, thus improving agility. This obviously also contributes to loose coupling (another tenet of SOA), as the binding between an application and its business rules is no longer as tight. The next section delves more deeply into the criteria used for evaluating an EDM offering.

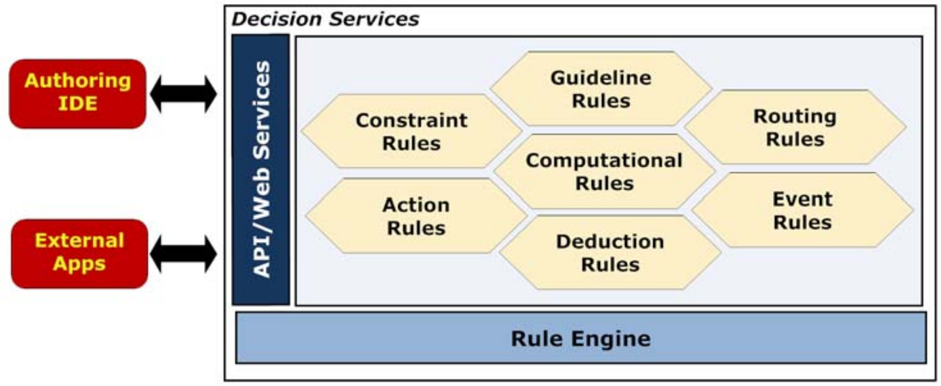

Figure 2.2 The components of an EDM, and its relationship to API services and rule engine

2.3.1 EDM product evaluation criteria

Section 2.1 identified some general criteria for evaluating the various SOA technologies, and an EDM obviously has some additional product-specific requirements. Table 2.4 identifies some key requirements we'll use when analyzing the suitability of the various open source rule systems.

Table 2.4 Open source selection criteria, general guidelines

Criteria	Comments
Centralized rule repository and classification	Central to the concept of EDM is a repository that enables rules to be classified, managed, and versioned. This should include the ability to add custom metadata and properties to each rule and ruleset. Security and access control are also important requirements.
Auditing and logging	In a time of increasing regulatory and compliance demands, the ability to audit the frequency and outcome of rule actions is essential. This can also provide analytical feedback to rule authors, allowing them to refine and improve their rules over time.
Integrated development environment (IDE)	A complete authoring environment for design, creating, testing, and publishing rules. Usually should include "wizards" or other forms of assistance for those new to the system.
Domain-specific language (DSL) support	We alluded to this briefly earlier: the ability to create a language based on business or domain nomenclature. An example of a rule expressed using a DSL is, "If Order is greater than $10,000, then sales manager approval is required." That, in turn, would be translated into a form that the rules engine could understand.
Robust API	Refers to the ability to integrate with the rules engine. This means not only providing programmatic access to the rule engine, but also whether it includes support for reading/writing data from popular SQL databases, where most fact-related data resides. In addition, the API should support multiple languages and/or have strong web services support.
Performance	Although performance was listed in section 2.1, it is worth reiterating because of the importance performance plays within an EDM. It's not uncommon to develop thousands of rules, and a highly efficient engine must be used since many rules must be fired in a real-time capacity.

Now that we have a foundation for assessing an EDM, we can turn to identifying the possible open source EDM candidates.

2.3.2 Open source EDM products

While commercial business rule solutions have been around for a decade or more, it's only been within the past five years or so that open source alternatives have become available. This is no doubt because of the increased visibility that has become associated with the "business rule approach," along with the success stories presented by the commercial vendors. Table 2.5 identifies the open source EDM products.

Table 2.5 EDM open source product overview

Product	Comments
Mandarax	Primarily just a rules engine with limited IDE support (Oryx, a UI editor for Mandarax, is maintained by a third party, and is a bit buggy and unrefined). Doesn't include a repository.
OpenLexicon	Fairly new product (2006) with limited documentation. Favorable license (modified Mozilla). Includes a polished management interface and repository. Can create rules through a web-based interface. DSL support is somewhat limited. Doesn't appear to be easily embeddable.
JBoss Rules (Drools)	Highly mature rules engine that has undergone several significant enhancements recently, which include the addition of BRMS repository functionality. DSL support is limited but useful. Highly efficient rules engine and decent Eclipse-based authoring environment. Lightweight and embeddable.
OpenRules	Restrictive license for commercial use (for example, you must purchase a non-GPL license if you're using OpenRules in a SaaS or ASP model). For this reason, it wasn't considered a viable selection for our Open SOA Platform. That said, it's a highly capable BRMS with a strong support community.
Jess	Jess, an early and highly respected rules engine, isn't open source or free, though it's commonly assumed to be (it's very affordable).
TermWare	Primarily targeted as an embedded solution. Doesn't include repository, management features, or IDE.

Based on the results in table 2.5, it appears as though the only two real choices are OpenLexicon and JBoss Rules (hereafter referred to as *Drools*, its historical name). Let's examine the reasons next.

2.3.3 Selecting an EDM

Mandarax, while maintained by a fairly small team of developers, does offer some innovative features. They include an elegant way of tapping into working memory from a variety of data sources, as well as a novel API approach to creating functions and predicate-style clauses using standard Java classes. Documentation is adequate. The biggest concern with Mandarax is that it's maintained by a small team and appears to have a limited user base. The concern is that, over time, without a strong user base the project could fall into quiescence and would no longer be actively maintained (a fate that afflicts the majority of open source projects). For this reason, we didn't consider Mandarax.

Both OpenRules and Jess were excluded from consideration due to their licensing restrictions. OpenRules, while proclaiming itself as open source, doesn't fit my criteria of open source: using it in certain commercial capacities requires purchasing a license. Although we are advocates of purchasing support for those open source applications that have a sponsoring company whose revenue model is based on that (such as JBoss), we think it's disingenuous to pitch a product as open source when a license must be purchased for commercial use. On the other hand, Jess clearly doesn't aim to mislead and doesn't position itself as open source (free versions for certain types of usage are available).

OpenLexicon shows great long-term promise, but the fact remains that it's still relatively new and lacks comprehensive documentation. Its nicely integrated BRMS features and well-designed user interface should definitely place it on anyone's open source short list. This leaves Drools, which has a long and proven track record and has been enhanced with more enterprise BRMS features, such as repository management.

2.3.4 Introducing JBoss Rules (Drools)

The Drools project began in 2001, and the first production-ready release was the 2.x version that appeared in 2003. By 2005, Drools had become a popular open source rules engine, so much so that in October of that year, it joined the JBoss family of products. With the deeper pockets afforded by JBoss (and then Red Hat, which, in turn, acquired JBoss in 2006), the 3.0 release of Drools offered significant performance enhancements and introduced an authoring/IDE Eclipse environment. In addition, a new rule language, DRL, simplified rule creation. Even more substantial improvements accompanied the 4.0 release. The rule language was enhanced; a Hibernate framework was introduced for populating working memory; performance was further improved; and, perhaps most significantly, BRMS functionality was added. The 5.0 release, which will be available by the time of this publication, adds further enhancements, related to process flow and includes complex event processing features (we are using the 5.0 release for the examples presented in this book). Drools can now claim to be a true feature-rich alternative to commercial BRMS offerings.

The Drools team at JBoss now includes over 12 full-time staffers, along with a fairly large contingent of non-JBoss contributors. The project has excellent documentation, which can be somewhat of a rarity in the open source world. The mailing list is also quite active.

If there's a knock against Drools, it's that a prebuilt web services interface isn't available. We address this deficiency in chapter 11, where you'll learn how to easily expose Drools rules as SOAP-based web services.

2.4 Choosing an ESB

As discussed in chapter 1, an enterprise service bus (ESB) is considered middleware that lies between business applications and routes and transforms messages along the way. Since the ESB acts as a messaging bus, it eliminates the need for point-to-point connectivity between systems. Instead, when one system needs to communicate with another, it simply deposits a message to the bus, and the ESB is then responsible for determining how to route the message to its destination endpoint. Any necessary transformations are performed along the way. Figure 2.3 illustrates the central role an ESB can play.

An important role an ESB plays is bridging between different protocols. For instance, an interface to an ERP system may require SOAP, but an internal CRM may only support XML over JMS. An ESB can translate between these protocols and lift JMS messages originating from the CRM and into a SOAP web service call understood by the ERP (and vice versa). Typically, ESB "adapters" perform the function of communicating with a disparate group of protocols, such as SOAP, CORBA, JMS, MQ Series, MSMQ, FTP, POP3, and HTTP, among others. We'll examine the ESB evaluation criteria next.

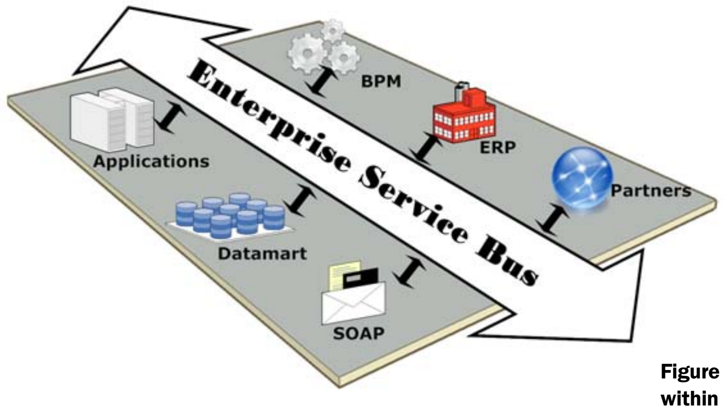

Figure 2.3 Central role of an ESB within the enterprise

2.4.1 ESB product evaluation criteria

In selecting which open source ESB to use for the SOA technology platform, let's consider several ESB-specific requirements, as shown in table 2.6.

Table 2.6 Open source selection criteria, general guidelines

Criteria	Comments
Protocol adapters	An ESB should support, at a minimum, adapters for the following protocols: POP3/SMTP, HTTP, FTP, SOAP, JMS and File.
Data-flow processing/ choreography	An ESB must often perform a series of tasks as part of a data gathering, routing, and transformation process. This requires the ability to chain together multiple steps into a processing pipeline that may require content-based routing, splitting, aggregating, and exception logic. For real-time processing, an ESB event-flow choreography may eliminate the need for BPM-type orchestrations (which are more suitable for long-running transactions).
Clustering and failover	Given the central role an ESB plays within a SOA environment, it must feature clustering and failover capabilities. In addition, the ability must exist to distribute, among a number of different servers, the various ESB services. For example, XSLT transformations can be very CPU intensive, so it may be desirable to isolate such processing on a separate server or servers.

Table 2.6 Open source selection criteria, general guidelines *(continued)*

Criteria	Comments
Transformations	Most ESBs, if not all, are XML-centric. That is, the messages that flow through the bus must typically be in XML format (binary data can be addressed through Base-64 encoding). As such, the ability to transform from one XML format to another is essential. While every ESB supports XSLT transformations, not all support XQuery, which adds significant query and transformational capabilities.
Service extensibility	A well-defined API should exist that easily permits creation of new services or adapters.

Although disagreement exists as to who invented the ESB (both Sonic Software, now a division of Progress, and TIBCO claim that honor), the first real mature commercial products began to appear around 2002. Emerging in 2004 was the first real open source ESB, Mule. It was closely followed by ServiceMix, which in turn was succeeded by several others. Now, there are at least half a dozen compelling open source ESBs. Indeed, it's difficult to make a clear-cut decision based on competing features, as several possess nearly identical capabilities (and this is no small feat, given how comprehensive these products are). Instead, the decision simply may come down to personal preference. In other words, you can't go wrong by picking nearly any of the top-tier ESBs.

What is the different between choreography and orchestration?

In a *choreographed* process flow, each node within the process determines which path to proceed moving forward. For example, each node could reside within its own Java virtual machine. It receives a message through some in-port queue, performs its processing, and then determines which out-port queue to deposit the message. The node is, in a sense, oblivious to its role within the larger process. With an *orchestration*, however, the process flow is managed centrally and typically within a single Java virtual machine. In the case of BPEL, each time a process is initiated, an "instance" of the process is created, and managed by the BPEL engine. If it is long-running, the instance may be persisted to a database (a process known as *dehydration*). Within a choreographed service, there's no concept of a "process instance," and the messages instead reside, somewhere, within the process nodes.

There's one distinction that can be made between some of the competing products—those that support the *Java Business Integration* (JBI) specification and those that don't. What is JBI? It's a Java Community Process (JSR 208) specification for building a runtime integration architecture and was formally approved in summer 2005. It expands on WSDL 2.0's message patterns to create a container that can house services and the consumers of those services. Without getting too immersed now into the technical nomenclature of JBI, suffice to say that it represents a standard for creating ESB components and its runtime messaging environment. Although it originally began with much fanfare, several early proponents such as IBM and BEA (now Oracle) soured on the JBI, and the follow-up version of the standard, intended to address many of its perceived inadequacies, has languished.

How important is JBI? That's a matter of great debate. Obviously the proponents of ServiceMix and OpenESB would argue that it's an important differentiator, as you are then not tied into a potentially proprietary solution. However, non-JBI implementations, such as Mule, could rightly point out that their product is based on open standards, just not JBI (though they do now offer JBI integration capabilities). It arguably also makes their products easier to use and configure, as JBI has some fairly abstruse configuration and deployment requirements. JBI does appear to be gaining some momentum, especially as the 2.0 specification (JSR 312) works its way through the approval process (it's purported to address some of the biggest deficiencies in the 1.0 spec).

With the JBI considerations in mind, let's take a look at the open source ESB products.

2.4.2 Open source ESB products

While the product category known as ESB is a fairly recent development, several open source products were quick to emerge. In part this was because a community of experienced developers already existed with great familiarity with messaging solutions such as JMS. There's a now a solid selection of products from which to choose, with several very mature. The open source ESBs are identified in table 2.7.

As table 2.7 indicates, there are several excellent choices. Let's take a closer look.

Table 2.7 Open source ESB product overview

Product	Comments
ServiceMix	Early (2005) JBI-compliant ESB. Has dozens of components/adapters and supports nearly every protocol. Allows creation of fairly complex data flows using enterprise integration pattern components. Active project with frequent releases.
MuleSource Mule	Broad connectivity options and is strong in transformation, routing, and security. Like ServiceMix, supports common enterprise integration patterns for real-time choreography. Vast array of components/adapters. Well documented, mature, and proven. Broad range of app servers supported.
Apache Synapse (WSO2 ESB)	Positioned as a lightweight ESB that, while supporting essential ESB functionality, is simple to use by way of XML configuration. In addition, it's designed with high performance and availability features that make it especially suitable for web mediation.
JBoss ESB	A fairly new entrant that still appears to be maturing. Not a greatly active user community, and using web services is tedious. Does provide nice integration with other JBoss middleware products.
OpenESB	Like JBoss ESB, a fairly new project that's still maturing. Version 2 promises to offer significant enhancements. Good IDE support through NetBeans plug-in. GlassFish App Server v2 has built-in support for OpenESB, but support for other app servers is lacking. Documentation is fairly sparse.

Table 2.7 Open source ESB product overview *(continued)*

Product	Comments
Jitterbit	Positioned more as an "end-user ESB" that's simple to use without being a developer. However, lacks broad protocol support. The concept of JitterPaks is novel and makes exchange of prebuilt integrations feasible. Backend written in C++, which limits appeal to Java shops. Strong LDAP integration capabilities.
Bostech ChainBuilder ESB	Adds polished user interface and management features to JBI containers such as ServiceMix or OpenESB. Eliminates a good portion of the tedium in configuring and packaging JBI assemblies. Documentation is adequate, though the project doesn't appear to have a lot of downloads, which raises concern about viability.
OpenAdapter	Mature, elegant, and lightweight ESB. Although it's been around for a long time, documentation is poor. Project activity is low, although a dedicated group of developers keeps the release cycle frequent. Maybe best suited for embedded-type applications.

2.4.3 Selecting an ESB

Both OpenESB and JBoss ESB are fairly new entrants into the space. While it's true that JBoss ESB has been around prior to JBoss purchase of the solution, it only recently introduced SOAP-based web services support. Sun's OpenESB appears to be gaining some momentum, but overall it lacks in documentation and mindshare (there's also confusion about its role in Sun vis-à-vis the SeeBeyond ESB that was acquired with Sun's purchase of SeeBeyond). At this point, we consider both OpenESB and JBoss ESB too immature, at least compared with some of the others, to consider as viable options.

Jitterbit, while very interesting, isn't positioned as a full-fledged ESB in the vein of the others. That said, it has a clever, user-friendly interface that's intended for technical business users and not necessarily developers. It supports the most common transport protocols and has excellent database connectivity with easy-to-use extraction wizards. On the negative side, documentation remains relatively weak, and there are some licensing restrictions introduced through its own Jitterbit Public License (which is unfortunate). Given the end-user orientation of the product, it isn't well suited for the complex ESB routing and transformational abilities that our Open SOA Platform demands. As such, it was excluded from consideration.

OpenAdapter is one of the easier ESB products to learn and use. It's very mature, and is lightweight and fast. It also has a devoted development team that provides frequent releases. Notwithstanding these positive attributes, it doesn't appear to have significant momentum or user adoption. Disappointingly, its documentation is poor, with only a few of their adapters adequately documented. Because of these reasons, we determined that OpenAdapter wasn't a good fit for the platform.

Both ServiceMix and Mule represent excellent choices. They both offer a broad range of functionality and support a wide range of transport protocols. A strong case can be made for either product. However, we believe that for most environments, Apache Synapse is the better choice. Why? The main reason is one of simplicity. Most of the ESBs we've talked previously about include relatively complicated configurations. This is particularly true of ServiceMix, which, by its JBI heritage, has a complex deployment structure. The same, albeit to a lesser degree, applies to Mule.

One of the earliest, and still most popular uses of an ESB, is to service-enable existing legacy applications. Common usage scenarios include exposing legacy services with a SOAP or HTTP wrapper. As you'll learn, however, this can be better accomplished using the Service Component Architecture (SCA). That being the case, the role of an ESB becomes less pronounced and instead is used primarily as a protocol bridge. Indeed, JMS solutions such as ActiveMQ, which is the default messaging product for many open source ESBs, now incorporate enterprise integration patterns, via Apache Camel, that can perform many tasks traditionally left to the ESB. This includes functionality such as routing, transformations, message splitting/aggregation, and content filtering. It may well be that the central role that ESBs have typically played within a SOA environment will reduce in next-generation architectures.

In light of these developments, we believe that Apache Synapse, because of its dual capacity as both a lightweight ESB and service mediation (discussed in section 2.8) is a prudent choice for most enterprises. For those requiring more sophisticated ESB capabilities, such as advanced routing features or more esoteric protocols adapters, consider using Mule or ServiceMix.

2.4.4 Introducing Synapse as a lightweight ESB

Synapse originated in 2005 from the X-Broker source code denoted by Infravio, which subsequently was purchased by WebMethods, which was then sold to Software AG. While the motivations for the donation are unclear, it likely was because Infravio was a vendor within the SOA registry space, and the X-Broker code wasn't considered a key offering. What is interesting is that, more recently, Synapse has become closely affiliated with WSO2, which has re-branded Synapse as WSO2's ESB. Most of the project members for Apache Synapse belong to WSO2. WSO2's ESB, which is also open source, tracks closely with the official Apache Synapse releases, and offers some nifty graphical front-end management and monitoring enhancements to Synapse. However, we won't demonstrate the use of WSO2's version, since learning the essentials of Synapse is the most important consideration (and matches our desire to keep things as lightweight as possible).

The initial Apache incubator proposal submitted by Synapse definitely positions it as an ESB-type product, with highlights citing multiprotocol connectivity, transformation features, and high performance, and management. Special emphasis is placed on proving support for the WS-* standards stack, which includes WS-Addressing, WS-ReliableMessaging, WS-Security, and WS-Policy. This is noteworthy, as Synapse will be used for such purposes within our Open SOA Platform. The latest release as of this writing is 1.2, which added numerous enhanced capabilities as well as improvements for scalability and robustness. That release builds upon the 1.1

WSO2's ESB 2.0 and Carbon

As we pointed out, WSO2 has largely provided the financial and development resources behind Apache Synapse. As this book neared production, WSO2 released a significantly upgraded version of their ESB product, upon which the future version of Synapse will likely be based. In this new 2.0 release, the WSO2 ESB was rewritten using their new Carbon framework, which is a modular, OSGi-based solution. Unfortunately, we didn't have an opportunity to evaluate this product yet, but please visit our SOA blog at http://jdavis.open-soa.info/wordpress for ongoing and updated information.

release, which added task scheduling, XQuery support, file system support through Apache VFS, and database mediation.

A simplified view of the Synapse architecture is shown in figure 2.4.

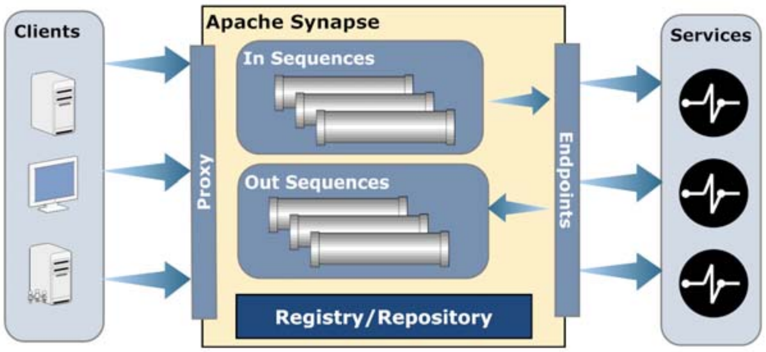

Figure 2.4 Simplified Apache Synapse architecture

As shown in figure 2.4, a request arrives from a client, and the proxy configuration determines which processing sequence to apply to the inbound message. Sequences are then applied to perform transformations, credential mapping, caching, security processing, and the like. Sequences can be applied to both the inbound and outbound messages, thus providing great flexibility. A remote or local registry can be used to facilitate reuse of commonly used sequences. Chapter 9 will go into much greater detail with code samples on the use of Apache Synapse.

2.5 *Choosing an ESP solution*

Event stream processing (ESP) is an emerging field that has begun to gather a lot of interest. It's considered a part of a broader trend known as *Event-Driven Architecture* (EDA). EDA is a style of application architecture that's centered on asynchronous, "push-based" communications. It's entirely complementary to SOA and uses asynchronous messages, in lieu of RPC-style function calls, to perform distributed computing. An *event* is simply an act of something happening, be it a new order, shipping notice, or employee termination. The system that records the event (or *sensor*) generates an *event object*, which is sent by way of a *notification*. The consumer of the notification (or *responder*) may be another system that, in turn, uses the event to initiate some action as a response. This is where the concept of ESP comes into play (which alternatively is sometimes called *complex event processing*, or CEP). Since ESP is a fairly nascent technology, let's take a closer look at it.

2.5.1 What is event stream processing?

ESP involves building or using tools to design, manage, and monitor the events that flow through an EDA-oriented environment. *Event patterns* are used to filter event data in order to detect opportunities or anomalies. An ESP solution must be able to support a high volume of events, perhaps millions daily, in order for it to be a viable offering. A business rule engine can be used in tandem with the event patterns to determine who receives what alerts. The relationship between these entities is shown in figure 2.5.

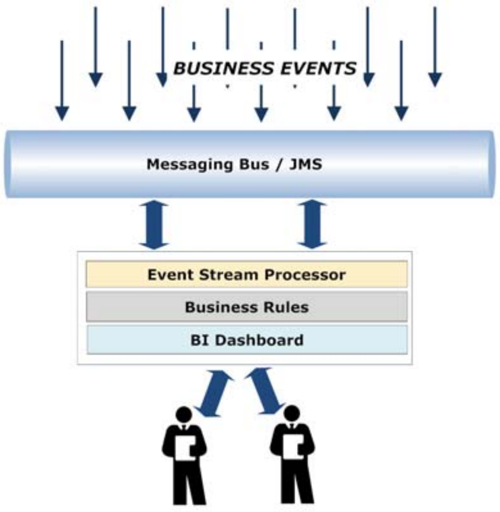

Figure 2.5 Event stream processing used for receiving business event notifications

In figure 2.5, messages that arrive into the JMS bus are interrogated by the ESP (sometimes referred to as *wire-tapping*). The business rules in the illustration may be contained directly within the ESP or externally managed, and drive the logic that occurs when certain patterns are detected. The results can then be fed into a BI dashboard.

BI, BAM, and ESP: are they all the same thing?

Business intelligence (BI) refers broadly to the technologies and applications used to analyze and present business information to targeted business consumers. *Business activity monitoring* (BAM), though similar to BI, tends to focus on real-time analysis of information. BI, on the other hand, often works in conjunction with data warehousing technologies to present analytics on historically gathered data. ESP shares the same real-time monitoring emphasis as BAM, but the source of data is derived directly from event streams. Historically, BAM solutions might cull real-time data from transaction records or BPM systems, but now are being enhanced to support ESP. So, BAM can be considered a super-set of ESP.

Perhaps because ESP is a fairly new concept, there's a dearth of open source solutions currently available that specifically address ESP. Esper is the only widespread open source ESP currently available. Some others are currently in development, including Pion. Several open source BI tools, which can be used in conjunction with an ESP to create executive dashboards, have become popular. Pentaho is perhaps the most recognized open source BI vendor, but others have successfully used tools such as JasperReports and Eclipse Foundation's Business Intelligence and Reporting Tools (BIRT) to create effective BI solutions. Though not open source, SeeWhy Software offers a "Community Edition" BI product that contains significant ESP capabilities. It can be used in production but is limited to a single release on any single-processor server.

Given that Esper is the only open source Java ESP currently available, let's examine it in greater detail.

2.5.2 Introducing Esper

The Esper project (whose name was derived from ESP-er, someone born with telepathy or paranormal mental abilities) was first released in August 2006. However, the project founder, Thomas Bernhardt, had developed earlier prototypes of ESP type solutions while working at a large financial institution. Since its initial release, a steady stream of updates has been provided (the most recent release, as of this writing, was 3.0). Beyond typical bug fixes, the main focus of enhancements relate to the Event Query Language (EQL), which is an SQL-like language for developing query expressions against inbound events. With EQL, you register prebuilt queries into the ESP engine, and as data is received, it's evaluated against those queries. Because events often must be viewed within the context of time (that is, no order in 15 minutes at night may be normal, but during the day, may indicate a website outage, for example), EQL provides "temporal window" syntax that allows time-period queries to be defined.

The documentation for Esper is quite good, especially since it's a fairly new project. This is likely because the founders of Esper have created a sponsoring company called EsperTech, which aims to build on the open source code base to introduce high availabilities and management features to Esper. This model is, admittedly, less than ideal for open source advocates, as it may mean some advanced features likely won't find their way into the open source release (this model contrasts with JBoss, who make their revenue entirely from support and do not limit the features found in their open source products).

Let's now turn our attention to the registry, which is used to store reference information about the artifacts that comprise a SOA.

2.6 Choosing a registry

The registry's role in our Open SOA Platform is to store the various software artifacts that are used in achieving a SOA environment. Historically, the *Lightweight Directory Access Protocol* (LDAP), which is a specification for directory services, was commonly used for registry purposes. It has become nearly ubiquitous in the enterprise because of Microsoft's Active Directory (AD) product, which is LDAP based. Most people mistakenly assume, in fact, that AD/LDAP is just intended for user and group management. Clearly, this is an excellent use of LDAP, but it's capable of considerably more. LDAP is ideally suited for any type of hierarchical directory where high-performance queries are required (with less emphasis on transactional updates and inserts).

Figure 2.6 depicts how LDAP could be used for managing a variety of artifacts, from individuals to BPM processes.

Although LDAP can be configured to support the management of software artifacts, it isn't necessarily ideally suited for this function. In particular, storing of the actual artifacts themselves, with the ability to query its contents, isn't easily accomplished without extensive customizations.

A more suitable fit than LDAP might be *Universal Description, Discovery, and Integration* (UDDI), which is a web services standard for publication and discovery of web services and their providers. While some vendors have released UDDI-based products (such as HP's Systinet), it has never achieved significant adoption. This is perhaps due to several reasons: complexity of the standard and its jargoned and arcane nomenclature (tModels, for example); its initial emphasis on public-based registries; and the initial lack of any strong UDDI open source offering. At best, UDDI is limping along, and the now available open source UDDI projects show little activity or enthusiasm.

One trend that has begun to emerge is that proprietary registry offerings have started to appear in SOA governance products. They are usually integrated with policy management features that dictate what services can be called by which clients. This is a sensible marriage, as governance obviously is closely tied to asset and artifact lifecycle management. Until recently, there have been no real open source SOA governance projects. Thankfully, that's now changing. WSO2 has released their WSO2 Registry product, and MuleSource released Galaxy, a SOA Governance product that is predicated on a registry. Since both are an initial 1.0 release, they're obviously a bit green around the edges, but these are exciting developments. Let's now take a look at some of the criteria we'll use for evaluating registry products.

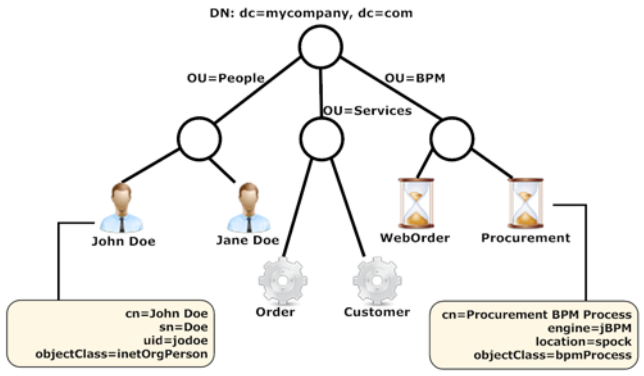

Figure 2.6 An example of an LDAP repository storing users, services, and BPM process metadata

2.6.1 Registry evaluation criteria

As you recall, in table 2.1 we identified some broad open source criteria that can be applied across all products we are evaluating. In addition, table 2.8 introduces some requirements specific to registries.

Table 2.8 Registry evaluation criteria

Criteria	Comments
Artifact and metadata repository	The ability to classify and store artifacts by type; for example, a WSDL or SCA configuration file. Should also allow for custom, searchable properties to be defined by artifact type.
Indexed searching	The ability to search metadata specific to the artifact type; for example, search operations within a WSDL, or components within a SCA composition.
Administration	Must include a graphical (preferably web) interface for managing and administering the repository. This would include the ability to add new artifacts, artifact types, search, and reporting.
Logging and activity monitoring	Should provide the ability to monitor activity within the system. This would include such things as new or modified artifacts and metadata modifications.
Role-based permissions	The ability to define users and user groups by roles.
API	The ability to interact with the repository through a programmatic API. Ideally, would be SOAP- or REST-based.

The next section identifies the possible open source products that can be used for the Open SOA Platform.

2.6.2 Open source registry products

The open source products that potentially can serve as the registry (see table 2.9) are broken into two main types: LDAP based and proprietary. For reasons we've already cited, the LDAP products have some disadvantages insofar as they're designed more as directory servers than artifact repositories. Nonetheless, it's worthwhile to consider them, since LDAP does provide extensibility features. The two open source UDDI implementations, Apache jUDDI and Novell's Nsure UDDI Server, weren't considered, for the reasons cited earlier regarding UDDI.

Table 2.9 Open source ESB product overview

Product	Type	Comments
OpenLDAP	LDAP	Proven, reliable, and has been around the longest. Now works with most popular backend databases. High performance and supports very large databases. Documentation is poor, which is surprising given its long heritage (though some LDAP books do cover OpenLDAP). Fairly complex to administer, and Windows platform support is sporadic (most run it on Linux or Unix).

Table 2.9 Open source ESB product overview *(continued)*

Product	Type	Comments
Fedora Directory Server (Red Hat)	LDAP	LDAPHeritage dates back to Netscape DS, and so it is mature. Excellent graphical administration console. Synchronizes with Active Directory. Good documentation. Intended to run on Red Hat or related Linux flavors (such as CentOS). No Windows capability.
ApacheDS	LDAP	100% Java-based solution. Excellent performance and support for many advanced features, such as triggers and stored procedures. Nice Eclipse-based plug-in (Studio) for browsing and editing repository. Lightweight and easy to administer.
OpenDS (Sun)	LDAP	100% Java-based solution that looks quite promising. Sun is positioning it as a possible replacement for their existing Sun ONE DS. At the time of this writing, version 1.2.0 has been released.
MuleSource Galaxy	Proprietary	Position as a SOA Governance product, it's based on a repository designed for managing SOA-based artifacts. This includes Mule configurations, WSDL, XML files, and Spring configurations.
WSO2 Registry	Proprietary	Designed to store, catalog, index, and manage enterprise metadata related to SOA artifacts. Includes versioning features and is lightweight enough to be embeddable.

As you can see, selecting the right product for the metadata repository service is difficult, as many high-quality open source products now exist (a good problem to have!).

2.6.3 Selecting a registry

We eliminated Sun's OpenDS from consideration, as it was still in beta during the early stages of writing this book. It is worth noting, however, that it has received excellent marks by those who have used it extensively. Some early benchmarks indicate that it's much faster than other Java-only based solutions (such as ApacheDS). It's being positioned as a complete, enterprise-ready solution, with advanced features such as "multi-master" replication and load balancing. The three principles touted in its development are ease-of-use, performance, and extensibility. The documentation is surprisingly strong for a fairly young open source project. Even though OpenDS's earlier beta status eliminated it from consideration, it's worth keeping a close eye on moving forward.

The Fedora Directory Server appears positioned primarily for Red Hat flavors of Linux—no Windows version exists. This fact limits its appeal and excludes it from our consideration. Even though it doesn't run natively on Windows, it's worth pointing out that it does have one of the best Active Directory synchronization features available.

This venerable OpenLDAP makes for an excellent choice. However, it too lacks strong Windows platform support (there are some Windows releases, but they're significantly behind the Linux versions). It can also be a challenge to administer and is fairly complex for those not well versed in Linux systems administration. ApacheDS, unlike OpenLDAP, is lightweight and simple to set up. It's also the only LDAP-certified open source product (Open Group certification). New releases appear to be bridging the performance gap between DS with OpenLDAP, and its Java codebase is appealing (assuming you're a Java developer).

While ApacheDS shows great promise as a directory server, it's still LDAP, which makes it rather challenging for supporting the storage and search of artifacts. The hierarchical nature of LDAP is also not ideally suited for our needs. Let's look at the two remaining proprietary products, Galaxy and WSO2 Registry, both of which were released in early 2008.

WSO2's Registry product appears to be a great fit for our registry needs. Positioned solely as a registry product, it's designed as a catalog for services and service descriptions. Artifacts can be structured data, such as XML-based files, or binary documents, such as Word or Excel. Metadata classification is supported, as are user-assigned tags, which can be useful for searching (think Flickr for the services). Versioning capabilities are supported, and the user experience is intuitive due to its Web 2.0 design (which is beautifully designed). User roles are also supported and configurable. Dependency and lifecycle management support is built in as well. One of the most attractive aspects of the product is the simple-to-use API. You can programmatically fetch objects from a remote repository in a few lines of code, and extending the registry to support custom object types by adding specific behaviors specific to them can be easily done.

The Galaxy product supports the same general feature set as WSO2's Registry, such as resource categorization, monitoring, and lifecycle and dependency management. In addition, the 1.5 release included some advanced features such as replication (available only in their pay version called Enterprise), scripting support, and an event API. That said, WSO2's Registry is easy to use, and trumps Galaxy with better Atom/RSS support and automatic versioning/rollback features. A good case could be made for selecting either product, but I remain a little leery of MuleSource's dual-licensing model, whereby some of Galaxy's most attractive features are only available for those who purchase the Enterprise license. WSO2, however, is 100 percent open source end to end, so no features are purposely excluded from their base product. For these reasons, we selected WS02's Registry product.

2.6.4 Introducing WSO2 Registry

WSO2's Registry product is officially positioned as a marriage of SOA registry with Web 2.0 collaboration features. The Web 2.0 features pertain to its ability for users to tag, comment on, and even rate registry entries/metadata. Figure 2.7 shows the essentials parts of Registry.

Beyond the core requirements of searching and managing artifacts and their metadata, the product supports the definition of artifact types. Using this feature, Registry can automatically introspect and index certain types of artifacts. Those supported out of the box include things such as WSO2's ESB (Synapse with added management capabilities) XML configuration files, WSDLs, and XML Schemas. You can easily define,

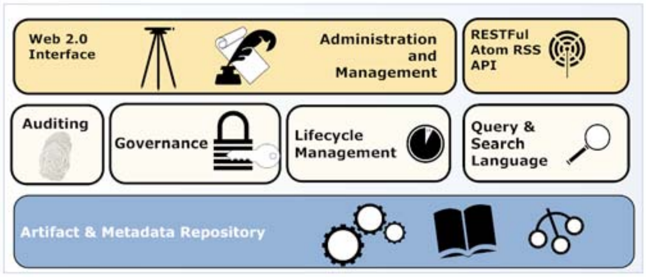

Figure 2.7 WSO2's Registry "marketecture" of features

through its extensible handler mechanism, your own custom behaviors related to filtered object types.

The lifecycle features of Registry enable larger enterprises to manage artifacts by their state within the development lifecycle. For example, you could search on artifacts that are in the QA state. Promotion of the objects throughout the defined lifecycle is also supported. The dependency management features pertain primarily to document types that support inclusions. For example, an XSD schema import within a WSDL can be automatically detected and then associated with the WSDL. Since schema documents play such a central role in a SOA environment for defining services, this is an important feature. The monitoring features provide excellent logging of all activity performed within the system, and nearly everything is exposed through a RESTful AtomPub API.

Bonus chapter

Coverage of WSO2's Registry product can be found in a bonus chapter found at Manning's website: http://www.manning.com/davis/. In part, we chose this approach since the Registry product is currently undergoing a major rewrite as part of WSO2's new Carbon platform, and we want to use that release as the focus for the chapter.

Let's now turn our attention to arguably the most critical artifacts of all: the services that constitute a SOA environment.

2.7 Choosing a service components and composites framework

Services are the catalyst behind a successful SOA environment. Exciting developments have occurred in this area over the past few years. The first salvo occurred with the release of Eclipse 3.0. The product was rewritten to include the OSGi framework for its runtime engine. OSGi uses a Java-based component model to dynamically manage the lifecycle of applications. With it, you can install, uninstall, start, and stop models within a runtime application. This technology represents the basis for Eclipse's plug-in architecture.

The OSGi framework, whose specification is managed by the OSGi Alliance, was initially formed with an emphasis on embedded devices and appliances. However, it rapidly has become adopted within regular and even enterprise, applications. There are currently three excellent implementations: Apache Felix, Knopflerfish, and Equinox (the basis for the Eclipse OSGi implementation). Many of the Apache-based projects are beginning to incorporate the OSGi framework.

Arriving a bit later was the *Service Component Architecture* (SCA) and its companion technology, *Service Data Objects* (SDO). The 1.0 specification was delivered in fall 2005 and included such notable sponsors as IBM, Oracle/BEA, and IONA. SCA positions itself as an architecture for building applications and systems using a SOA. It defines a declarative XML-based mechanism for creating components and for exposing those components as services that can be invoked through any number of different protocols. Components can be wired together in a fashion similar to Springs "inversion-of-control" feature, and are written in a way that is communication protocol neutral (that is, the components have no awareness as to which protocol will be used to invoke them, such as SOAP, JMS, or EJB). Given that a lot of folks are probably not yet familiar with SCA, let's examine some of its core concepts in more detail.

2.7.1 Examining the Service Component Architecture

In SCA parlance, a *composite* is a collection, or assembly, of components or services. A service can be thought of simply as a component that's exposed through an external protocol (for example, SOAP). A component, like a composite itself, can contain properties and references to other components or services. You can see the relationship between these items in figure 2.8 (which is a simplified view of SCA).

As figure 2.8 shows, a *binding* is how you define through what communications protocol to expose a given component as a service.

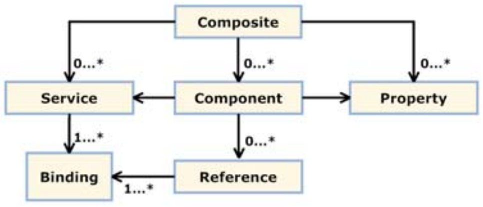

Figure 2.8 A simplified SCA class diagram

SDO is a companion specification that defines a standard for exchanging data graphs or sets. What's unique about the standard is that it supports the notion of *change logs*. This allows for offline modifications to be captured and recorded. The graphs themselves can be serialized into XML, and class-generation tools exist to create SDO objects from an XML Schema (alternatively, they can be created dynamically, and the metadata describing the structure can be gleaned dynamically as well).

What's the relationship between OSGi and SCA? In a sense, they're competing technologies, as they both define a component framework for creating services. However, OSGi primarily was designed for running within a single JVM and doesn't inherently support exposing of services through a wide range of protocols. SCA, on the other hand, was developed with the goal of supporting a multilanguage distributed environment. There's an initiative to bridge the two so that you can, for instance, easily deploy

OSGi services within SCA. In that respect, the technologies can nicely complement each other, and indeed, the next major release (2.0) of Tuscany is being internally rewritten to run within an OSGi container. For purposes of our SOA Platform, we won't specifically address OSGi, but we strongly encourage further research on the subject if you aren't already familiar with it. Apache Tuscany, an SCO and SDO open source implementation, will be addressed in great detail starting in chapter 3.

Upon first examination of SCA, many Java developers are led to believe that it's a substitute for Spring (as many of you are aware, Spring is a popular Java application framework). In part this confusion arises because SCA, like Spring, enables references (or other components) to be injected at runtime. Spring, like SCA, also supports the notion of properties, which can be declaratively defaulted in the XML configuration. Spring-WS even supports exposing Spring-based beans as web services, so that's another similarity. That said, important distinctions exist, such as SCA's aforementioned multiprotocol and multilanguage support. In addition, SCA more intuitively supports asynchronous and conversational programming models. Like OSGi, Spring integration is also available for SCA.

Because of the reasons cited, and SCA's integrated support for SDOs, it's the service and component framework technology of choice for the Open SOA Platform. Let's now further explore Apache Tuscany, the open source implementation for SCA and SDO.

2.7.2 Introducing Apache Tuscany

Apache Tuscany is a fairly new project, with its first beta releases in 2006 followed by the 1.0 release in fall 2007. The development team appears well staffed and is likely funded by the likes of IBM. The project recently was anointed as a top-level Apache Project from its prior incubator status, which corresponded with the 1.3 release in August 2008. Version 1.4 was released in January 2009, and is the basis for the code samples used in this book. The SCO and SDO standards have been transferred to the aegis of the OASIS organization. This is a significant development, as it lends great credibility to the project and removes the cloud that the combined specification was just a product of a few select vendors. OASIS has also set up a special website called Open Service Oriented Architecture (www.osoa.org) dedicated to advancing the standards.

The Tuscany and OASIS websites collectively contain extensive documentation. The specification documents for SCA and its related technologies are well written and comprehensive. There are also a burgeoning number of SCA-related articles and some upcoming books dedicated to the standard. The demo and code samples that come with the Tuscany distribution are also very solid and a wonderful source of information.

Commercial support for SCA and SDO has become realized by product releases by IBM (WebSphere), Oracle/BEA (WebLogic, AquaLogic, Workshop) and Oracle (SOA Suite 11g). Clearly, the momentum for SCA and SDO continues unabated.

The last remaining technology that helps form the basis for the Open SOA Platform is web service mediation.

2.8 Choosing a web services mediation solution

Web service mediation rounds up our Open SOA Platform. Web service mediation plays several key roles within a SOA environment. They include the following:

- *Runtime governance*—A service mediator can use security profiles to determine which clients can access what data. For example, you can modify an outbound data packet to restrict what data is presented. You can also monitor compliance with service-level agreements. Monitoring and logging can be used for compliance and auditing.
- *Version rationalization*—Often multiple versions of a company's API exist. A mediator can transform earlier versions into a format/specification consistent with the most recent version. This eliminates having to manage multiple versions of backend code.
- *Traffic management*—In certain circumstances, it may be desirable to discriminate traffic based on a client profile. For example, a SaaS provider may choose to charge extra for more than *x* number of requests per minute. For those clients not paying extra, inbound requests will be governed.
- *Protocol mediation*—This refers to the ability to easily translate messages from one protocol to another: for example, converting a REST-based XML-over-HTTP request into a SOAP format required for internal consumption. Or another scenario is to add or remove WS-Security headers from an inbound SOAP request.

Figure 2.9 illustrates the role a web service mediator plays in receiving inbound requests from a client.

Historically, some of these features were available through hardware devices, such as F5 Networks' BIG-IP family of products, Cisco's various content switches, or Intel's XML Content Router. As you might imagine, these generally require a fairly deep pocketbook. Until recently, pure-play open source mediation products didn't exist.

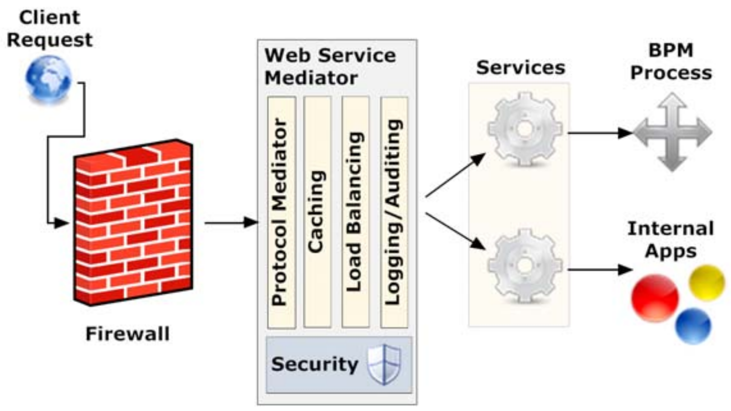

Figure 2.9 Web service mediation used as a proxy for inbound requests

Open source's hidden documentation

One of the most undervalued forms of documentation available in open source projects is the JUnit test cases that are usually available with the source. There are often a multitude of tests available for nearly every facet of behavior. What's most instructive is how the tests, or assertions, are defined, as they shed great light on the anticipated behavior of the application. Sometimes the test cases also provide insights into methods not documented within the regular documentation.

Granted, some of the features can be accomplished through an ESB, such as version rationalization. Some proxy and caching servers, such as Squid, also provided some of the requisite functionality.

The Apache Synapse project, which launched in 2005, became the first open source web service mediation designed solution. Because it does share some overlap in terms of functionality with an ESB, it can also do double duty as a lightweight ESB (you may recall from section 2.4 that it was, in fact, selected as the ESB for our Open SOA Platform). The Synapse feature set, which includes proxy, caching, load-balancing/fail-over capabilities and superb WS-Security support, clearly positions it as best suited for web service mediation. Let's examine the Synapse project in more detail.

According to the press release announcing Apache Synapse, it's "an open source implementation of a Web service mediation framework and components for use in developing and deploying SOA infrastructures" [Synapse]. Joining WSO2 in announcing Synapse was Blue Titan, IONA, and Sonic Software—all well-respected players in the SOA community. The first production release was in June 2007, and was followed by a 1.1 release in November of that year. Synapse is part of the Web Services Project at Apache (and is also a top-level Apache project), and the 1.2 release is the basis for our coverage of the product in this book.

The documentation, at first blush, seems rather inadequate. However, much of the best documentation resides within the write-up for the 50 or so samples that come with the distribution. Collectively, they provide a great deal of worthwhile information (you can find additional information on WSO2's website listed as their ESB product). The project mailing list is also fairly active.

WSO2's release of Synapse also includes a nice administrative interface to Synapse, and you'll learn more about it in chapter 9's in-depth dive into Synapse.

2.9 Summary

This chapter conducted a whirlwind examination of the key product categories of the Open SOA Platform. For each, we identified a product, usually among several excellent choices, as our selection. The categories and products selected are shown in table 2.10.

Table 2.10 Product categories and selections

Product category	Product selection	Home
Business process management	JBoss jBPM	http://labs.jboss.com/jbossjbpm/
Enterprise decision management	JBoss Rules (Drools)	http://labs.jboss.com/drools/
Enterprise service bus	Apache Synapse	http://synapse.apache.org/
Event stream processing	Esper	http://esper.codehaus.org/
Metadata repository	WSO2 Registry	http://wso2.org/projects/registry
Service components and composites	Apache Tuscany	http://tuscany.apache.org/
Web service mediation	Apache Synapse	http://ws.apache.org/synapse/

These products are well regarded and supported, and form the basis for the remainder of the book. The biggest challenge is how to integrate these products in a meaningful way so as to create a compelling open source SOA.

A note on the examples and source code

Throughout many of the chapters, example code is presented to assist the reader in understanding the concepts. To move the discussion along, we skirt past how to set up and run the examples. However, the downloadable source code contains a README.txt file for each chapter that walks through setting up your environment and running through each of the examples. If you encounter any issues, please use the Manning Author forum associated with this book at http://www.manning-sandbox.com/forum.jspa?forumID=416 to report any problems, and we'll attempt to resolve them as quickly as possible.