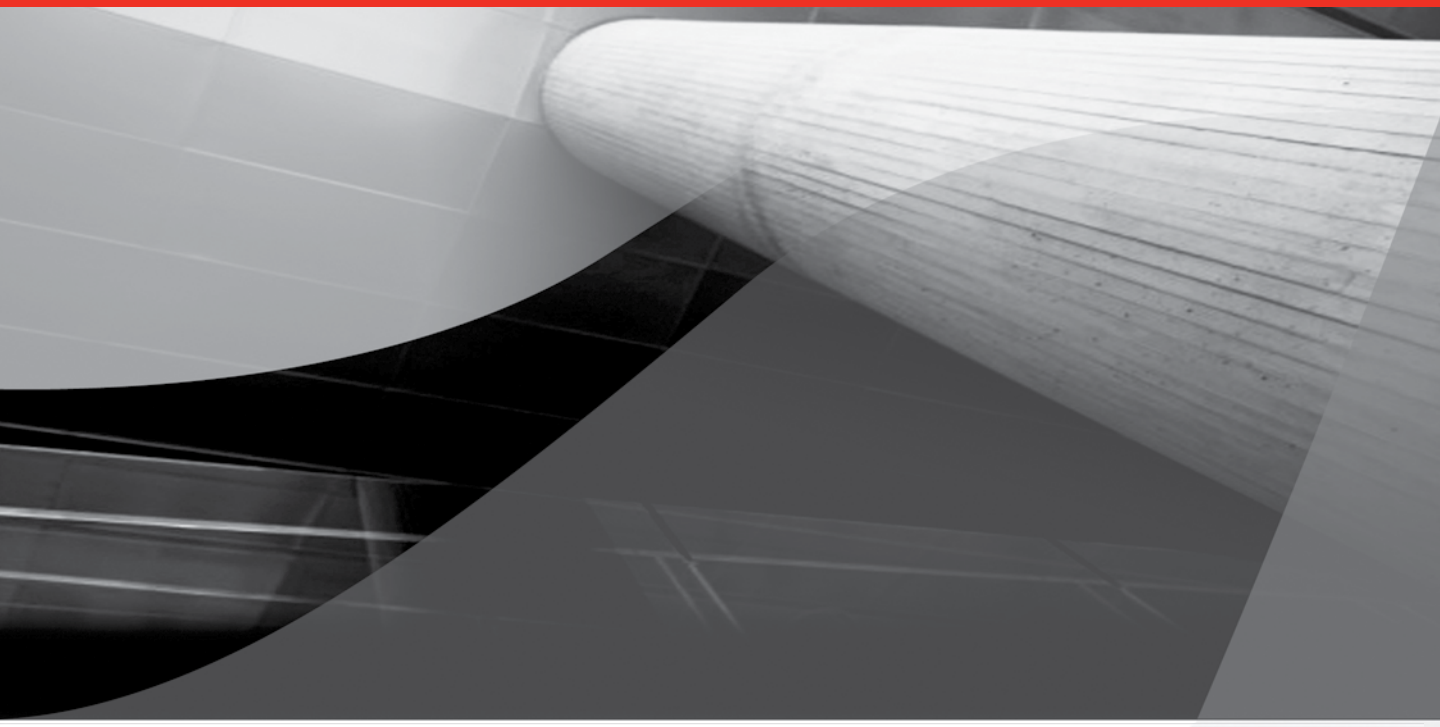
ORACLE



Oracle Fusion Middleware 11*g* Architecture and Management

ORACLE FUSION MIDDLEWARE 11*g*

Deploy, Secure, Optimize, and Manage Oracle Fusion Middleware Applications

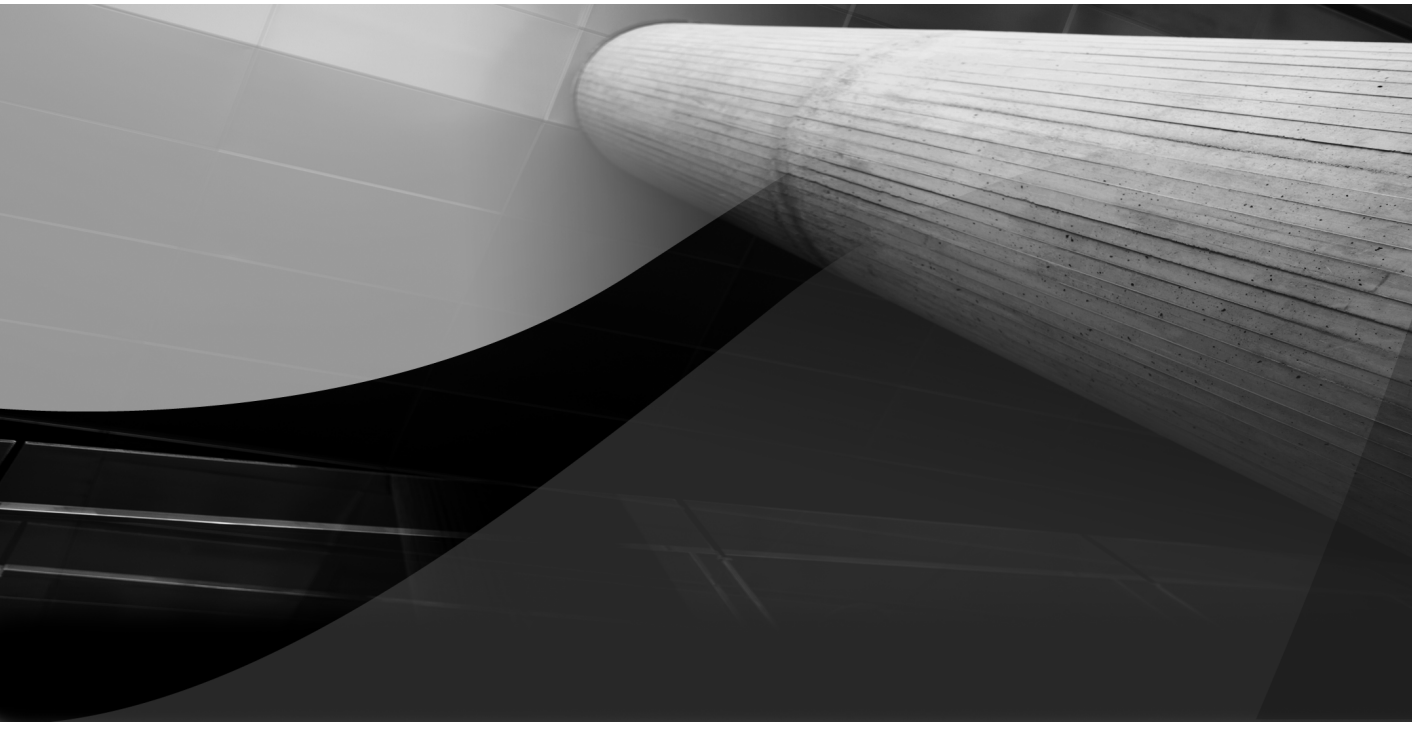
Reza Shafii
Senior Principal Product Manager, Oracle Fusion Middleware, Oracle

Stephen Lee
Senior Product Manager, Okta

Gangadhar Konduri
Senior Director, Product Management, Oracle Fusion Middleware, Oracle

Foreword by Adam Messinger, Vice President of Development, Oracle Fusion Middleware, Oracle

ORIGINAL • AUTHENTIC
Oracle Press
ONLY FROM McGRAW-HILL



CHAPTER 3

Fusion Middleware Common Infrastructure

racle has built a set of components that together provide common services used by Fusion Middleware products. Not only do these services facilitate a consistent architecture, they also provide common management capabilities across the Fusion Middleware products. In this chapter we will examine the core elements of these common components, which we will refer to as the Fusion Middleware common infrastructure. In doing so, we will look at the services provided by this infrastructure and their relationship with the WebLogic Server services we discussed in the previous chapter. As we will see, the Fusion Middleware common infrastructure not only provides a rich set of capabilities for the layered Fusion Middleware products built on top of it, but in some cases it can also be used by custom applications built on WebLogic Server to extend their capabilities and simplify their management.

Introduction to Fusion Middleware Common Infrastructure

In this section we review the core concepts required for an understanding of the Fusion Middleware common infrastructure and review its installation and configuration artifacts.

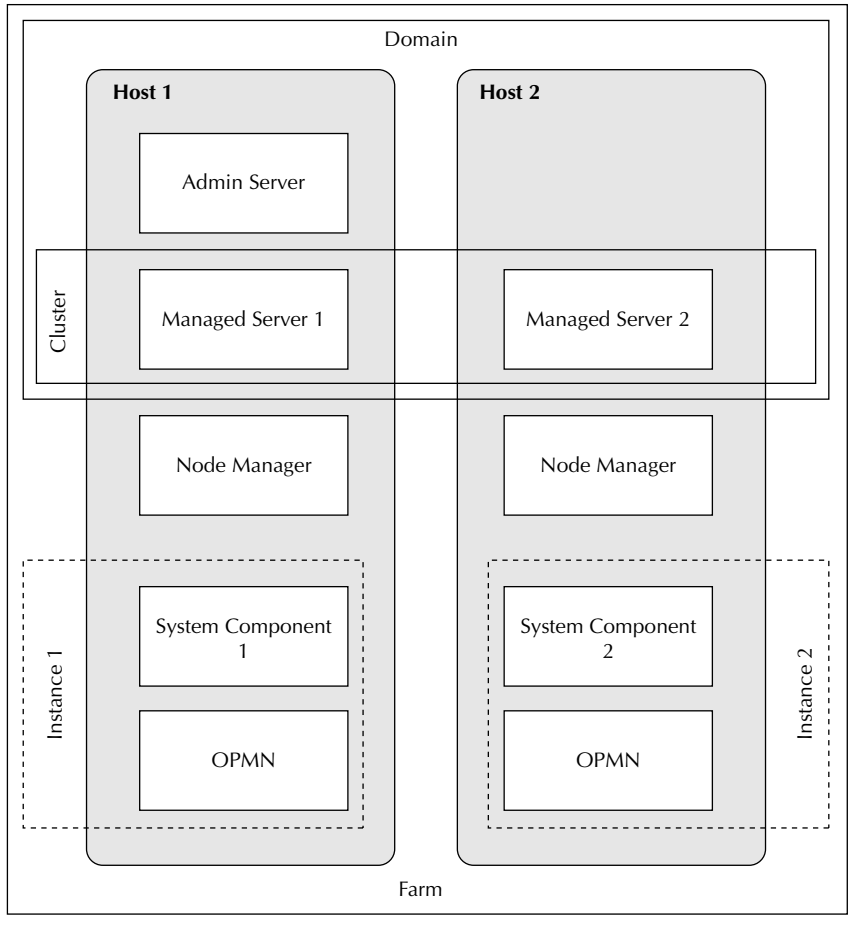
System Components, Instances, and Farms

As we saw in the previous chapter, the concept of a domain containing servers and clusters forms the basis of the WebLogic Server administrative management and administrative scoping model. The Oracle Fusion Middleware infrastructure extends this model with the concepts of a farm, instances, and system components. The addition of these elements is designed to allow for the management of products that have components not based on Java EE and therefore do not have a natural place within the WebLogic Server domain model. A number of products within the Fusion Middleware product set have such non–Java EE components. These products include:

- Oracle Portal, Reports, Forms, and Business Intelligence Discoverer
- Oracle Internet Directory (OID) and Oracle Virtual Directory (OVD) products of the Oracle Identity Management (IDM) Suite
- Oracle WebCache and the Oracle HTTP Server (OHS) web server

In the context of this book, the non–Java EE products we will be discussing are limited to the IDM products and the OHS web server. We will discuss the IDM products in more detail in the next chapter and briefly discuss the Oracle HTTP Server in this chapter's use case.

The following diagram illustrates the relationship of the administrative elements of a farm, instances, and system components within a WebLogic Server domain. We will discuss each of these new elements in more detail in the remainder of this section.



System component refers to the configuration of a non–Java-EE-based Fusion Middleware product component (the underlying technology behind

the component could be anything, including C/C++ or Java Standard Edition). A system component process is therefore the actual runtime operating system process (or a set of processes) that uses a particular configuration. The life cycle of system component processes is managed by a Fusion Middleware common infrastructure component called Oracle Process Manager and Notification (OPMN). The set of system components managed by the same OPMN entity and share single parent directory for their configuration is referred to as a Fusion Middleware instance. Table 3-1 provides a description of the most important paths directly under the directory structure that encapsulates the content of a Fusion Middleware instance's configuration, the root of which we will from now on refer to as INSTANCE_HOME.

Directory/Filename	Description
INSTANCE_HOME/bin	Contains the opmnctl command for managing the system components of this instance.
INSTANCE_HOME/ diagnostics/logs	Contains the log files for each configured system component under directories with names matching each system component's name (for example, "ohs1"). This directory also contains an "OPMN" directory, which contains the logs for the instance's OPMN process.
INSTANCE_HOME/config	Contains the configuration files for each configured system components under directories with names matching each system component's name. As an example, for an OHS system component, this directory would contain the OHS httpd.conf.

TABLE 3-1. *Instance Home Directory Structure*

A Fusion Middleware instance can be registered with a single WebLogic Server domain, which allows for the system components within that instance to be managed by the Enterprise Manager Fusion Middleware Control web application. We will discuss the Enterprise Manager capabilities in detail within a dedicated section later in this chapter. Finally, we need to introduce the concept of a farm. The set of Fusion Middleware instances registered with the same domain for management by Enterprise Manager is referred to as a farm. In other words, a Fusion Middleware farm consists of a single domain and its set of registered Fusion Middleware instances, each of which is a group of system components managed by a single OPMN entity.

Oracle Process Manager and Notification

Oracle Process Manager and Notification (OPMN) consists of a set of artifacts that together allow for the management of the Fusion Middleware system components. As mentioned in the previous section, a single OPMN entity—that is, the set of artifacts that form OPMN—is associated with each instance of system components within a farm. An OPMN entity is composed as follows:

- An opmn.xml file within the INSTANCE_HOME/config/OPMN/opmn directory. This file contains the configuration of OPMN itself as well as the list of all system components that belong to the single Fusion Middleware instance managed by it.
- An OPMN process that constitutes the OPMN runtime. This process is known as "process manager" and is responsible for the life cycle of system components in terms of starting, stopping, registering with a domain, and detecting failures. An instance of OPMN is configured as a set of UNIX daemons or Windows processes as part of the system component's instance configuration. On UNIX, OPMN is actually made up of two daemon processes: a parent that only watches the child and restarts it if necessary, and a child that does all the work. On Windows, it is typically run as a Windows service, where the service process serves the purpose of the parent process on UNIX. However, it is possible to run OPMN in the same two-process setup on Windows if a service is not desired. A command-line tool called opmnctl serves as the main interface for user

interaction with OPMN. One copy of this tool resides within the ORACLE_HOME/ opmn/bin/opmnctl directory and is used for the creation and deletion of instances. Additionally, each instance has a copy of opmnctl under its INSTANCE_HOME/bin directory for the management of that particular instance, such as the starting and stopping of system components and their registration or deregistration with a particular domain.

OPMN serves three main purposes within the Fusion Middleware architecture. First, OPMN serves as an agent that allows for the starting, stopping, and automatic restarting of system components. In this regard, OPMN very much provides the same purpose as the WebLogic Server node manager process, except that it does so for non–Java EE components. Second, OPMN allows for the collection of status, metrics, and log information from system components and provides a central access point to this information. Finally, OPMN serves as the bridging point for the integration of a Fusion Middleware instance with a domain within a farm by exposing all of its information and capabilities through an MBean within the farm's admin server. We will be analyzing in more detail how OPMN provides these capabilities within the "Enterprise Manager Fusion Middleware Control" section of this chapter.

Installation and Configuration Artifacts

The Fusion Middleware common infrastructure components do not have a dedicated installation program and are therefore not installable in isolation. These components are instead installed as part of the installation of the Fusion Middleware products that rely on them. The installation tool for such products is the Oracle Universal Installer (OUI). When the OUI for a particular Fusion Middleware Product is executed, it installs the product-specific binaries within a dedicated directory referred to as the product's Oracle Home (ORACLE_HOME). In addition to the installation of a Fusion Middleware product's binaries within its ORACLE_HOME, OUI also creates a directory known as a common Oracle home (COMMON_OH) as a child of its target MW_HOME directory if one does not already exist. The COMMON_OH is populated with the binaries of the Fusion Middleware infrastructure. Table 3-2 provides a description of the most important paths directly under the COMMON_OH directory.

Directory/Filename	Description
COMMON_OH/modules	Contains the Java libraries and binaries that form the functionality of the Fusion Middleware infrastructure components.
COMMON_OH/bin	Contains the common tools for the administration of all Fusion Middleware products installed within MW_HOME.
COMMON_OH/common/templates	Contains the WebLogic Server domain configuration templates that contain the domain configuration for the Fusion Middleware infrastructure components.
COMMON_OH/doc	Contains a snapshot copy of the Fusion Middleware documentation library.
COMMON_OH/OPatch	Contains the Oracle Patch (OPatch) tool, which is used for the application of one-off patch updates to any of MW_HOME's installed Fusion Middleware products.
COMMON_OH/rda	Contains the Remote Diagnostic Agent (RDA) engine, which is a tool used to capture detailed information from a specific Fusion Middleware environment to aid Oracle support in assisting with service requests.

TABLE 3-2. *Common Oracle Home Directory Structure*

Beyond its use for the installation of the product's ORACLE_HOME and COMMON_OH within a specific MW_HOME, OUI is also used for the configuration of a farm environment for Fusion Middleware products with OPMN-managed (non–Java EE) components. For this purpose, OUI can be used for the following specific tasks:

- Creation of a domain with the component's Java EE–specific elements deployed

- Creation of an instance and system configuration for the component's OPMN-managed elements
- Registration of an instance with a domain to form the farm that can be centrally managed through Enterprise Manager

The first task listed can only be performed by OUI for products such as the Oracle Portal that have both Java EE and OPMN-managed components. For products with only OPMN-managed components, such as the Oracle HTTP Server (OHS), the creation of a farm, including its domain, as part of the configuration of a specific environment is not required, and the instance can be optionally registered with an existing domain. Also, note that the creation of an instance configuration and its registration with a domain can also be performed through opmnctl instead of OUI. For the creation of configuration environments for products that are purely Java EE based (for example, Oracle SOA Suite and Oracle WebCenter), no creation of instance configuration and registration with a domain is necessary and therefore the creation of the domain is performed through the use of the WebLogic Server domain template configuration tools, which we covered in the "Domain Creation and Templates" section of Chapter 2. We will discuss the domain creation process for these products in more detail in later chapters dedicated to these topics. Table 3-3 summarizes the installation and configuration process we just discussed for each of the Oracle Fusion Middleware products that are part of the scope of this book.

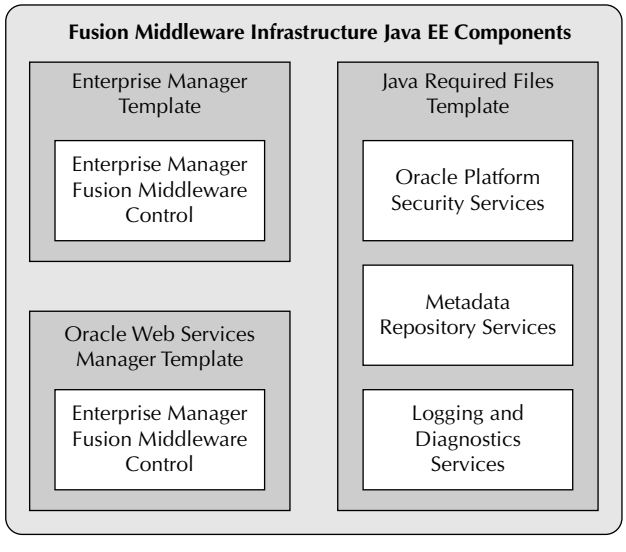
As described in Table 3-3, each Fusion Middleware product that requires a WebLogic Server domain for the hosting of its Java EE component's deployment archives has a top-level WebLogic Server domain configuration template. This template ensures that the product's required deployment archives are deployed to all domains that have been created through it. Furthermore, the template ensures that the archive's target servers are configured with the required WebLogic Server resources as needed by the product's deployment archives.

Each top-level product template has a dependency on three other templates that deploy the components of the Fusion Middleware common infrastructure within the domain being created. These three templates are Java Required Files (JRF), Enterprise Manager Fusion Middleware Control (from now on simply referred to as Enterprise Manager), and Oracle Web Services Manager (OWSM). We will discuss Enterprise Manager and OWSM

Product	Installation	Configuration
Oracle WebLogic Server	Performed through the WebLogic Server installation program as covered in the "Installation and Domain Artifacts" section of Chapter 2	Performed through the WebLogic Server configuration framework and via the use of the base WebLogic Server domain configuration template, as covered in the "Domain Creation and Templates" section of Chapter 2.
Oracle HTTP Server	Performed through OUI	Instance configuration and domain registration are performed through OUI, opmnctl, or Enterprise Manager.
Oracle Internet Directory (OID)	Performed through OUI	Instance configuration and domain registration are performed through OUI, opmnctl, or Enterprise Manager. Optional domain configuration for the management of OID is also performed through OUI.
Oracle SOA Suite, Oracle WebCenter, Oracle Application Development Framework (ADF), and some identity management products	Performed through OUI	Performed through the WebLogic Server configuration framework and via the use of the product's top-level domain configuration template.

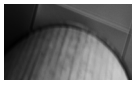
TABLE 3-3. *Oracle Fusion Middleware Product Installation and Configuration*

in dedicated sections later in this chapter. The JRF template contains the common Java libraries required by each Fusion Middleware product. These libraries provide common security, logging, and diagnostics, as well as metadata repository services to their consuming Fusion Middleware applications. They are made available to a WebLogic Server instance through the provisioning of Java EE applications, shared libraries, MBeans, WebLogic Server startup classes, or JVM system class path JAR files. We will discuss the security services of the JRF template, known as the Oracle Platform Security Services (OPSS), in more detail in Chapter 4 and later in Chapter 9. The logging and diagnostics services contained in the JRF template are discussed in detail in Chapter 11, but we also touch upon them within the "Enterprise Manager Fusion Middleware Control" section of this chapter. Finally, we explore the capabilities provided by the Metadata Repository Services (MDS) component of JRF later within a dedicated section of this chapter. The following diagram illustrates the relationship between these three different components that form the WebLogic Server domain-level elements of the Fusion Middleware common infrastructure and the three domain configuration templates we just discussed.



Before concluding this section, it is important to note that in most cases the Fusion Middleware common infrastructure as well as layered Fusion Middleware products provide their own WebLogic Server Scripting Tool

(WLST) commands. As reviewed in the previous chapter, WLST is a Jython-based scripting tool for the administration of WebLogic Server–based environments. Fusion Middleware products enhance the WLST set of commands with their own in order to enable a consistent scripting interface for the management of WebLogic Server domains with layered Fusion Middleware products. However, the execution of the WLST shell commands located within the WL_HOME/common/bin/wlst.sh|cmd directory or the direct execution of WLST through the Java command line does not provide access to the Fusion Middleware common infrastructure or layered product's command extensions. To be able to execute such commands, the copy of the WLST shell command within each product's ORACLE_HOME/common/bin directory must be used instead.

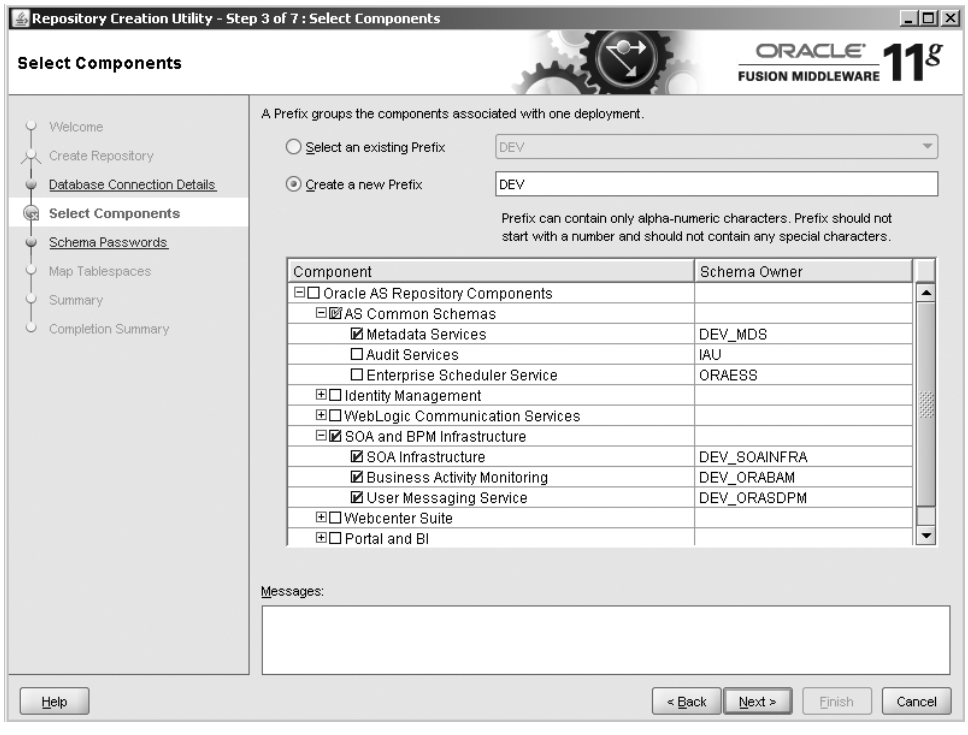


NOTE
To execute WLST commands provided by the Fusion Middleware common infrastructure or layered products, the WLST command located within each product's ORACLE_HOME/common/bin installation directory must be used.

Repository Creation Utility (RCU)

Some Fusion Middleware products have dependent database schemas that need to be created and seeded as part of their installation. The Repository Creation Utility (RCU)—a standalone tool downloaded separately from each Fusion Middleware Product—allows for the proper creation of such schemas. RCU allows users to specify a prefix to the product's schema names and thus allows for the existence of multiple schemas for the same product within the same RDBMS instance. RCU is also aware of the dependencies of a given product's schema on other schemas and ensures that all of them are created. Furthermore, RCU is designed to be used for use by DBAs and provides advanced schema management capabilities, such as the ability to specify a schema's table space configurations prior to its

creation. The following illustration shows the RCU screen used to specify the schemas to be created and seeded.



RCU maintains a schema (and creates it when it does not exist) called schema_version_registry that contains information about all Fusion Middleware schemas housed within a particular database instance. This table is used by the Fusion Middleware life-cycle tools and also provides a convenient way of querying and analyzing metadata about a given RDBMS instance's schema set. The information contained within the schema_ version_registry includes the following:

- Short name (for example, SOA)
- Long name (for example, Oracle SOA Suite)
- Owner (name of schema)
- Version (for example, 11.1.1.4.0)

- Upgrade (N or Y)
- Status (Upgrading, Valid, Invalid)

The schema_version_registry is created as a system-level schema. Users with DBA roles have access to the entire content of the schema_version_registry, and RCU-created users have access to their associated schemas.

To conclude our discussion of the RCU, we should note two other important aspects of this tool. First, the tool provides an extensible framework that allows end users, through a set of XML configuration and DDL files, to configure their own application-specific schemas to be created through it. Second, RCU is packaged separately from any Fusion Middleware product and does not require installation. To run the RCU, one simply has to execute the rcu.sh|bat file from the /bin directory of the RCU package.

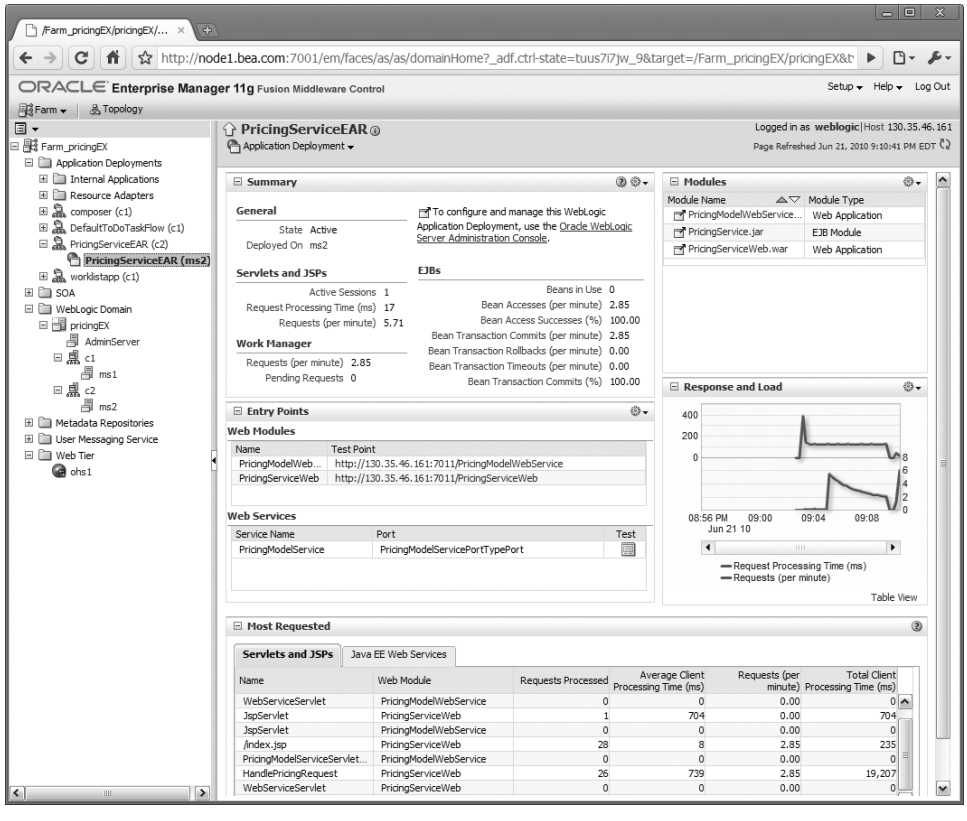
Enterprise Manager Fusion Middleware Control

Enterprise Manager Fusion Middleware Control (Enterprise Manager) is a web-based application that allows for the centralized management of all the components of a specific Fusion Middleware farm. The management capabilities of Enterprise Manager span the farm's domain as well as all of its registered instances of system components. Enterprise Manager provides its capabilities through integration with the WebLogic Server infrastructure, which we covered in Chapter 2, as well as the Fusion Middleware common infrastructure components. In this section we will be discussing in more detail Enterprise Manager's capabilities, architecture, and dependencies on the WebLogic Server and Fusion Middleware common infrastructure. Note that in this section we do not discuss the security, MDS, and OWSM functionality exposed through Enterprise Manager. Instead, we discuss security in Chapter 4 and MDS and OWSM functionality in dedicated sections later in this chapter.

Enterprise Manager Core Functionality and Architecture

Just like the WebLogic Server admin console, Enterprise Manager can be primarily thought of as a web-based JMX client that surfaces a key set of

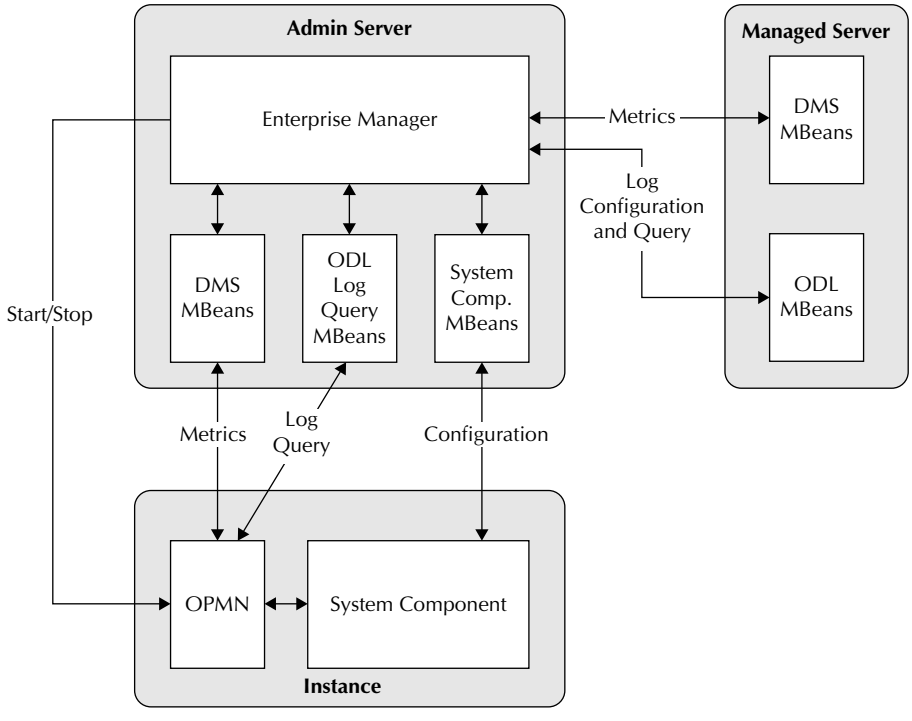
JMX MBeans for simplified management through its graphical user interface. The difference is that although the admin console strictly surfaces WebLogic Server–specific MBeans, Enterprise Manager surfaces key Fusion Middleware infrastructure (as well as a small subset of WebLogic Server) MBeans in order to provide a centralized management console to a Fusion Middleware farm. The following is a screenshot of an Enterprise Manager window when a Java EE–deployed application is selected.



Enterprise Manager offers a wide range of functionality through a rich and context-sensitive web-based interface. Each Fusion Middleware product expands Enterprise Manager with its own product-specific functionality. Furthermore, Enterprise Manager allows for a view of a Fusion Middleware domain's deployments organized into internal Fusion Middleware product-specific archives and custom WebLogic Server application deployment archives. This view comes in handy because the Fusion Middleware

infrastructure components as well as the layered Fusion Middleware products each introduce a significant set of WebLogic Server deployment archives, which to the administrator of custom applications built on top of them are for the most part irrelevant.

There's a core set of functionality provided by Enterprise Manager that concerns all Fusion Middleware components. In the next few sections we will explore the different categories of capabilities that this core Enterprise Manager functionality falls into and discuss their associated architecture and integration with the Fusion Middleware common infrastructure. The following diagram illustrates the Enterprise Manager integration with OPMN and the domain components we will be discussing in more detail in each of the following sections.



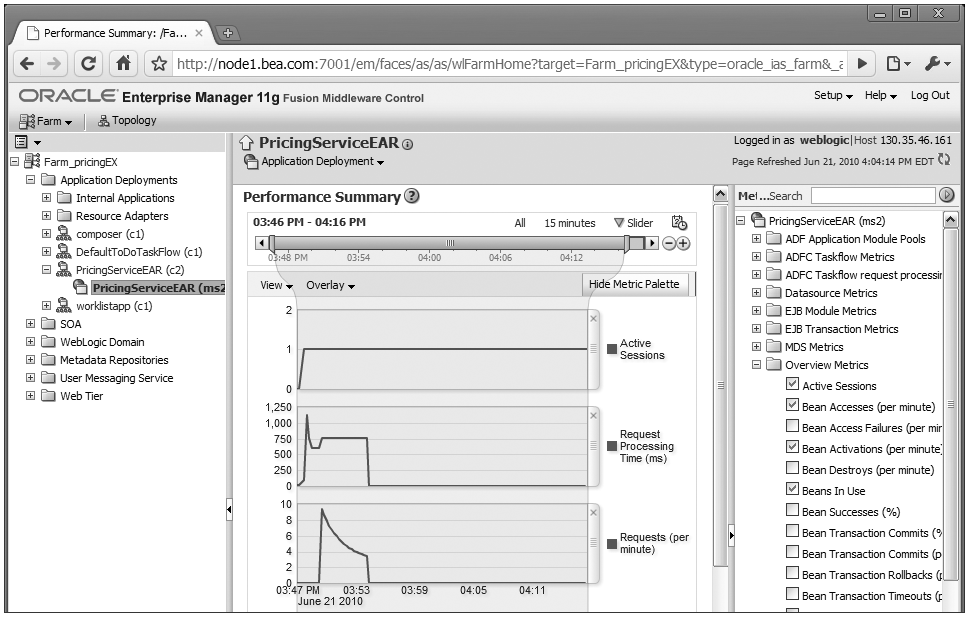
Component Startup and Shutdown

Enterprise Manager can serve as a central point of control for starting and stopping a Fusion Middleware farm's deployed Java EE applications, domain servers, and system components. To start/stop applications as well as WebLogic Server instances, Enterprise Manager uses the underlying WebLogic Server MBeans. Note that to start a WebLogic Server instance,

the node manager on which the server is running must be configured, as is the case for a plain WebLogic Server domain. For the stopping and starting of system components, Enterprise Manager makes direct calls to the system component instance's OPMN process, which in turn executes the operation on the target system component.

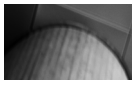
Metrics Monitoring

Enterprise Manager provides a consolidated view of performance data captured from all of the farm's components. Although this information is distributed in a context-sensitive manner throughout a number of Enterprise Manager pages, it can also be centrally accessed through the Enterprise Manager performance management page, which provides a wide range of analysis options, including graphing, filtering, and selecting specific metrics. The following screenshot shows an Enterprise Manager performance management page for a Java EE–deployed application.



The metrics surfaced through Enterprise Manager are obtained through a JRF layer component known as the Dynamic Monitoring Service (DMS). For managing WebLogic Server instances and Java EE applications, DMS

registers a runtime MBean within each managed server's Runtime MBean server. This MBean collects a set of metrics for each Fusion Middleware component running on its host server and exposes the information for collection from a special DMS collection MBean on the admin server. The DMS collection MBean periodically queries the DMS MBean on each manager server to collect a snapshot of its metrics. For the collection of performance metrics from system components within the farm, the DMS collection MBean queries each instance's OPMN process manager for performance data. OPMN, in turn, queries a specific DMS interface, which each Fusion Middleware system component makes available for the querying of its metrics. This DMS interface uses a custom RPC protocol, known as the Oracle Notification Server (ONS), for direct communication between OPMN and the system component process. Upon collection of a metrics snapshot from managed servers and system components, the DMS collection MBean on the administration server stores the information within an in-memory historical collection. Enterprise Manager, in turn, queries the DMS collection MBean to display the information available within its performance management windows.



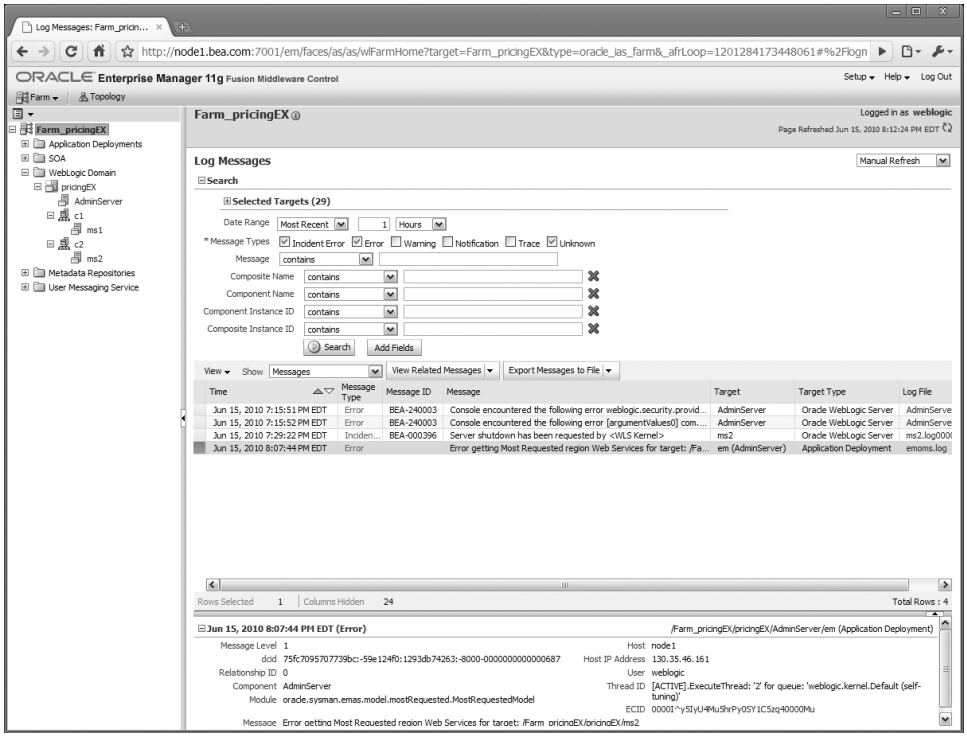
NOTE
The metrics data displayed in Enterprise Manager is not persisted and is stored within the administration server's heap only. As such, a restart of the administration server clears all historical metrics data for a given farm.

DMS also provides WLST-based interfaces for access and management of a farm's metrics.

Log Management

Enterprise Manager provides central logging management for all farm components. The log management capabilities for a domain, its servers, and applications consists of both log querying as well as log configuration capabilities, whereas for system components only log-querying capabilities are available. The log configuration capabilities of Enterprise Manager consist of the ability to set logging parameters, such as the maximum log level for a specific Fusion Middleware layered component, and the ability to

customize log file rolling parameters. The Enterprise Manager log-viewing capabilities consist of the ability to query log messages—at either the farm, domain, server, system component, or layered component specific artifact (for example, SOA composite) level—and to filter them based on date range, message type, message content, and/or originating component. The following is a screenshot of an Enterprise Manager Log Query page at the farm level.



The log management capabilities of Enterprise Manager are exposed through a JRF layer component called Oracle Diagnostic Logging (ODL). ODL is a Java logging API-based framework used by Fusion Middleware components to log their messages in a standard format. Fusion Middleware layered components that run on WebLogic-managed servers log their messages to the domain's DOMAIN_HOME/servers/<server name>/logs/<server-name>-diagnostics.log file (where <server-name> is a placeholder for the actual managed server name concerned). System components log

their messages to their instance's INSTANCE_HOME/diagnostics/logs/<system-component-type>/<system-component-name>/<system-component-name>.log file. In a similar fashion to the DMS component, ODL registers a runtime MBean within the server's Runtime MBean server for each managed server within a domain. This MBean provides access to the server's ODL log messages and their logging configurations. On the admin server, ODL provides an MBean that exposes the domain's aggregated ODL configuration and log messages by communicating with each managed server's ODL MBean. The admin server ODL MBean can also query a farm's OPMN process manager for the ODL logs of their associated instance; however, it does not provide the capability to configure the ODL logging configuration of system components. When OPMN receives a log query request, it looks through the log directory of the target system component within the instance directory structure and returns the requested results to the ODL MBean. Enterprise Manager uses the admin server ODL MBean to provide its centralized log management capabilities.

ODL exposes some of its management capabilities through WLST. We will be exploring ODL and the Fusion Middleware logging capabilities in more detail in Chapter 11.

Other Capabilities

Beyond the startup/shutdown, performance monitoring, and logging capabilities we discussed in the previous sections, Enterprise Manager provides three other notable cross-cutting capabilities. The first of these is a web service testing tool that can be invoked through the application page of any deployed Java EE application with web services. This tool simplifies the testing of web services interfaces and provides some advanced testing capabilities, such as the ability to specify the credential information that would need to be carried out by test invocations for web services with security policies. The second capability is a system MBeans browser that can be invoked through the context-sensitive pop-up menu from any of the Enterprise Manager's tree pane entities. This tool provides the capability to graphically browse, view, and (in the case of configuration MBeans) edit a farm's MBeans directly. The third and final capability to highlight comes into play when new managed servers need to be added to an Enterprise Manager–managed domain. As we discussed within the "Installation and Configuration Artifacts" section of this chapter, JRF components are enabled on managed servers at domain configuration time through the use of the JRF

domain configuration template. However, for managed servers created after the initial creation of a domain, the JRF components required are not automatically provisioned. Enterprise Manager automatically detects such servers and within their server information page provides an Apply JRF button, which can be used to deploy the appropriate JRF components on the server and therefore enable full Enterprise Manager management capabilities. Note that this operation can also be performed through the WLST "applyJRF" command.

Metadata Repository Services

Metadata Repository Services (MDS) is a component of the Fusion Middleware common infrastructure that provides the ability for layered Fusion Middleware products to decouple their metadata from their core logic and content data by storing it within a shared repository. The services and management model provided by MDS facilitate a simple administration and customization of the metadata of the layered Fusion Middleware components. An understanding of MDS is important for the management of Fusion Middleware environments, and in this section we will review its main capabilities and architecture.

Metadata Management

Fusion Middleware applications often have a dependence on metadata in order to allow for a flexible approach for the customization and extension of their custom application logic to fit specific usage scenarios. As we will see in detail within the next section, a good example of such metadata includes the definitions of predefined web services policies that are available for the runtime used within a Fusion Middleware domain. The purpose of MDS is to provide a service that allows for centralized management of such metadata. In the subsequent chapters of this book we will discuss in detail the specific usages of MDS-managed metadata by individual Fusion Middleware products, but some examples include Oracle Application Development Framework (ADF) application customization information and Oracle SOA Suite composite's configurations.

MDS metadata content is stored in repositories that can be either file based or database schema based. Schema-based MDS repositories are created using the RCU, whereas a file-based repository can be any file

system directory. To be usable by an application deployed to a Fusion Middleware domain, MDS repositories must first be registered with the domain and must have one or more partitions. A partition is a logical subdivision of the content of a particular repository, and in most cases applications that share an MDS repository each have their own dedicated partition. Both domain registration and partition management can be done either through Enterprise Manager or through WLST commands.

As part of its metadata management capabilities, MDS provides versioning, purging, and import/export capabilities to its consuming applications. Versioning allows changes to any metadata object by its consumer applications to lead to a new version of that object, thus leading to auditable and recoverable changes. For recovery, MDS's label management commands—available through WLST—must be used to create a label for an application's current MDS content. Rollback of the metadata content to a previous state across all of the application's metadata objects is then possible through label promotion. MDS's WLST purging command allows for cleanup of old versions of metadata content for maintenance purposes. It should also be mentioned that MDS provides customization capabilities that allow for changes made to an application's initially seeded metadata to be treated as separately managed customizations. This MDS capability is heavily used by the Oracle Application Development Framework and Oracle WebCenter, and we will be discussing it in more detail in the context of these products' needs within upcoming chapters dedicated to these topics. Finally, the MDS import and export capabilities can be used to export the content of an MDS repository to a file and later import it into another file-based or RDBMS-based MDS repository.

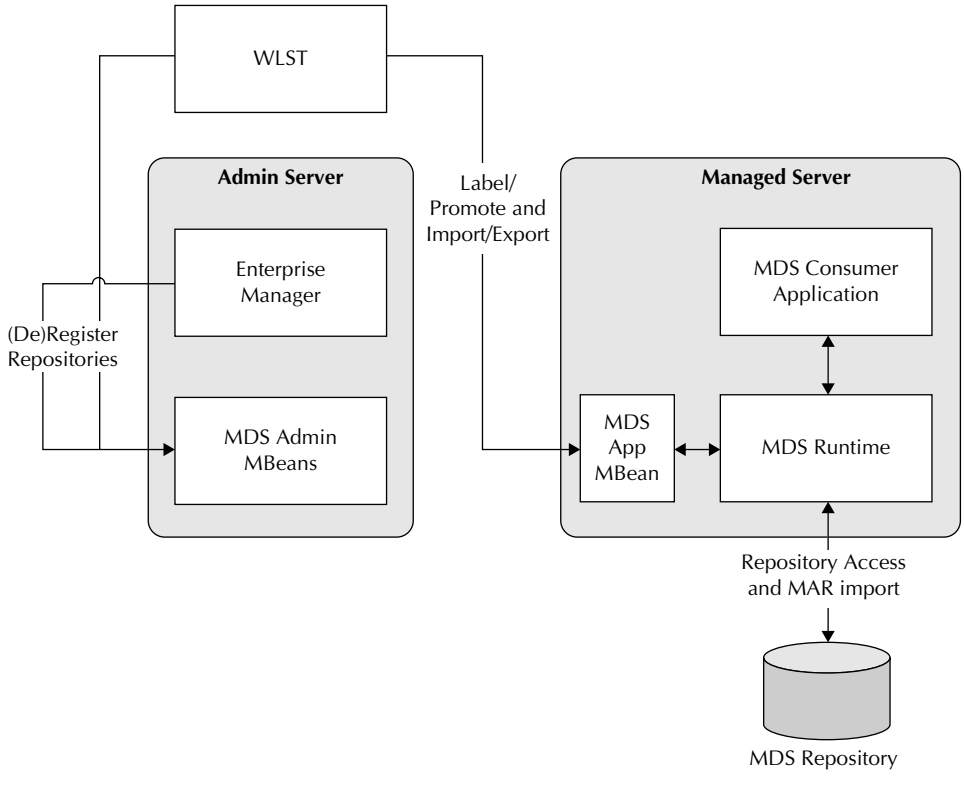
Deployment Model

Applications associated with Fusion Middleware layered products that use MDS (such as SOA Suite and ADF) have two MDS-specific deployment-time considerations that are important to note. First, they need to be explicitly associated with a specific MDS repository and partition for the storage of their metadata information. This association is contained within an MDS-specific application deployment descriptor and can be specified either at development time through JDeveloper, prior to deployment through the use of the MDS WLST "getMDSArchiveConfig" command, or through the Enterprise Manager Deployment Wizard, which provides the option of

performing this association on the deployment archive prior to deployment to its WebLogic Server targets. Second, application archives that have MDS dependencies can contain seeded metadata (for example, Oracle SOA Suite composite definitions) that needs to be deployed to the layered component's MDS repository upon the deployment of the archive to a WebLogic Server domain. This seeded metadata must be packaged in a special archive file, known as the Metadata Archive (MAR) file, within the EAR deployment archive of the application. Upon deployment of a Fusion Middleware application with a MAR file, the MDS runtime imports its metadata content into the application's associated MDS repository partition.

Runtime Architecture

The MDS runtime components consist of the MDS runtime engine and a set of MBeans that are applied to a domain as part of the JRF template. The following diagram illustrates how these MDS components interact.



The MDS runtime engine provides the implementation for the MDS services used by consuming applications to interact with their MDS repository and is also the entity that triggers the deployment of an EAR file's MAR content into MDS upon deployment. The runtime engine also exposes a set of MBeans used by Enterprise Manager and MDS WLST commands to provide the MDS label management and import/export capabilities. Finally, the runtime engine provides a built-in cache for each application's accessed metadata objects. The parameters for the tuning of this cache—as well as other MDS engine runtime parameters—are exposed for each application as MBeans and can be modified for each WebLogic Server instance through the Enterprise Manager System MBeans browser. MDS also exposes a set of MBeans on the domain's admin server that are used by Enterprise Manager and MDS WLST scripts for the registration and deregistration of existing MDS repositories.

Oracle Web Services Manager

Oracle Web Services Manager (OWSM) is a component of the Fusion Middleware common infrastructure that provides the ability to create, manage, and enforce security and management policies that can be attached to web services deployed within a WebLogic Server domain. In this section we will review the core capabilities and architecture of OWSM.

Policy Management

Web services deployed to a Fusion Middleware environment often need to be enhanced with specific configurations that tune their behavior to match environment-specific security, quality of service, and management requirements. Such enhancements are usually better managed separately from the implementation of the web services through externally defined policies applied to web services at invocation time. This separation allows for the addition or modification of polices and their specific configuration to happen independently from the modification of the web services implementation logic and thus to be easily adjusted to suit the application's needs. OWSM enables such separation of policies from web services implementation by providing the capability to define web services policies either at development time or runtime and by attaching them to specific web services endpoints within a Fusion Middleware domain. For web

services deployed to the WebLogic Server container we discussed in the "Web Services Container" section of Chapter 2, OWSM policies can only be defined and applied to JAX-WS-based web services. OWSM, however, does support a third type of web services known as Fusion Middleware infrastructure web services. This is used by the Fusion Middleware layered products—namely Oracle SOA Suite, WebCenter, and Application Development Framework—and will be explored in more detail in upcoming chapters.

Development-time web services are only available for Fusion Middleware infrastructure web services and can be defined through Oracle JDeveloper, Oracle Fusion Middleware's primary integrated development environment (IDE), and embedded within the application's deployment archives. Given this book's focus on the runtime and environment-specific architecture of the Oracle Fusion Middleware components, we will not be discussing the development-time management of OWSM web services policies, but instead will focus on OWSM's runtime capabilities.

Within a domain, OWSM stores its policies and their attachment information—in terms of the endpoints to which a policy is attached—within a dedicated MDS partition that is configured when the OWSM domain configuration template is used. MDS allows OWSM to support versioning of policies as they are modified through its built-in versioning capabilities. Also by the virtue of MDS, the migration of OWSM policies can be performed through MDS-level import and export commands, as we discussed in the "Metadata Repository Services" section.

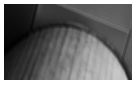
At runtime, the domain's OWSM policies are managed through two different Enterprise Manager pages. The first page (accessed through the domain node's Web Services | Policies option) allows for the definition of new policies and or modifications of existing policies. The second page (accessed through a specific application's Web Services | Web Services Endpoint | Policies option) allows for the attachment of specific policies to the application's web services, known as service endpoint policies, or its web services clients, known as client endpoint policies. Service endpoint policies are attached to the web services implementation interface and lead to a set of tasks performed on incoming web services requests prior to the request's processing by the web service. Client endpoint policies, on the other hand, lead to a set of tasks performed on outgoing requests messages and are as such attached to web services consumer endpoints. Note that the attachment of client policies at runtime are only supported for Fusion

Middleware infrastructure web services. This means that they apply only to web services consumers that are either SOA composites or ADF applications.

Although custom policies can be created through the Enterprise Manager policy creation and modification page, OWSM comes with a set of prepackaged policies designed to be sufficient for addressing the most common web services policy requirements. The set of prepackaged OWSM policies fall into the following different categories.

- **Security** Security-based policies form the majority of the OWSM policies. Such policies are used to indicate a request's required security characteristics in terms of the type of authentication information needed, SSL requirements, and/or authorization rules for the web service's invocation.
- **Message Transmission Optimization Mechanism (MTOM)** MTOM policies are used to validate compliance to the MTOM specification for SOAP requests with binary content.
- **Web Services Addressing and Reliable Messaging** Policies falling in these categories allow for the checking of the SOAP request headers for compliance with their respective web services specification.
- **Management** Currently OWSM only provides a single prepackaged management policy that is used to specify that a request's content must be logged at the client or service endpoint to its server's log file.

Note that only security policies are supported for WebLogic Server JAX-WS-based web services and the remaining category of web services policies is only supported for Fusion Middleware infrastructure web services. Security policies that require authentication lead to the web service's request thread within the WebLogic Server to have an authenticated subject upon successful authentication through the domain's configured authentication providers. From an authorization policy point of view, web services calls cannot be protected through the WebLogic Server's default XACML authorization provider policies. To define policies that can be enforced using a subject that has been authenticated through an OWSM security policy, the attachment of a separate authorization policy that specifies the web service's authorization rules is required. The OWSM authorization policies use the Oracle Platform Security Services (OPSS) authorization policy management services, which we will be discussing in detail in the next chapter.



NOTE
WebLogic Server authorization provider policies are not enforced on web services security endpoints that are protected using OWSM security policies. For this purpose, OPSS authorization combined with OWSM authorization policies must be used instead.

In addition to the Enterprise Manager interfaces available for managing policies and their attachment to web services endpoints, OWSM also offers a set of WLST management commands for the attachment and detachment of policies to web services endpoints. For a detailed description of these commands, please refer to "Oracle Fusion Middleware Security and Administrator's Guide for Web Services."

The Anatomy of a Policy

Each OWSM policy consists of an ordered set of assertions that are themselves instances of assertion templates. Assertion templates describe a set of tasks that need to be performed on a web service's request content. On top of the set of prepackaged policies, OWSM also comes with a prepackaged set of assertion templates that can be used for the creation of custom policies. Aside from its sequenced set of assertions, other important attributes of a policy are as follows:

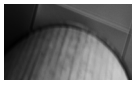
- **Category** Can be one of MTOM (Message Transmission Optimization Mechanism), Security, Addressing, Management, or Reliable Messaging. As we will see in the next section, a policy's category affects its order of execution at runtime.
- **Local Optimization** Can have a value of "on," "off," or "check identity." A value of "on" signifies that when a web service invocation's client and server endpoints are located on the same managed server—and thus running within the same local operating system process—then the policy does not need to be executed, whereas a value of "off" means that the policy must always be executed. A value of "on" is used for optimization purposes in order to save the performance costs of the policy's execution for nonremote and therefore potentially more trusted, invocations. A value of "check identity" signifies that the policy is not executed upon local invocation only when the request thread is associated with an authenticated subject.

With this understanding of OWSM policies in mind, we can now turn our attention to the OWSM runtime architecture and the way in which policies are enforced.

Runtime Architecture

To provide the policy management functionality we covered within the last section, Enterprise Manager interacts with an application known as the policy manager. WLST OWSM commands also lead to interactions with the policy manager applications, although they are first directed to OWSM-specific MBeans that are deployed to the domain's administration server. The policy manager application is deployed to a single server within any Fusion Middleware domain as part of its creation through the OWSM domain configuration template. As such, the OWSM policy manager is a singleton service, and the failure of its managed server signifies that the attachment, addition, or modification of policies will not be possible until the server is brought back up. However, a failure of the policy manager application does not prevent policy enforcement on web services endpoints within the domain.

The policy manager is responsible for maintaining the OWSM MDS policy repository, which contains all policy and assertion template definitions as well the mapping of attached policies to their target endpoints. The mapping of policies to web services endpoints uses information about the managed servers on which the endpoint's web services or its clients are deployed. For this reason, the content of the OWSM MDS policy store is tightly coupled with the domain's managed server and application deployment topology.



NOTE
When using the MDS import/export functionality on the OWSM partition in order to migrate it from one environment to another, one needs to ensure that the two environments have the exact same topology in terms of the number and names of managed servers that have applications with OWSM attached policies.

Each web service deployed to an OWSM-enabled domain is automatically extended with its own OWSM agent. An OWSM agent is a web services handler that is associated with its associated web services client or service

endpoint and is responsible for the enforcement of all OWSM policies attached to its endpoint. OWSM agents discover their attachments by periodically polling the domain's policy manager application, through an entity known as the Policy Enforcement Point (PEP), for information about their policy attachment changes. A single PEP instance exists per managed server, and it is responsible for the polling of the domain's policy manager for the discovery of attached policies and their changes to the managed server's web services endpoints. The PEP also caches these policies in memory for use by the managed server's agents. Whenever the PEP detects policy changes for a web services endpoint, it stops the deployment archive associated with that web service, modifies its web service's WSDL to reflect the attached policy set's needs using WS-Policy statements, and restarts the archive so that it can start serving requests again. Figure 3-1 illustrates the OWSM architecture we have discussed so far in this chapter.

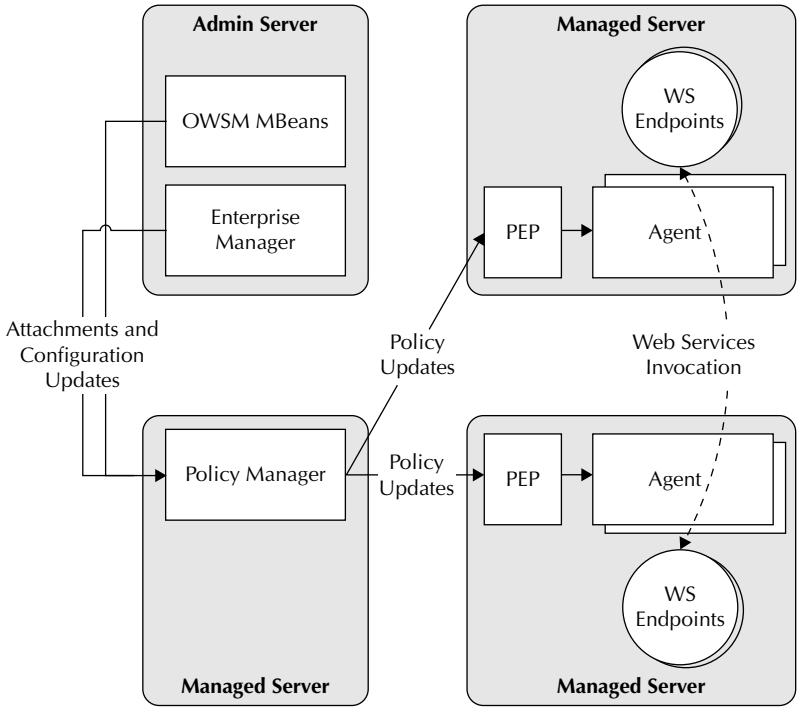


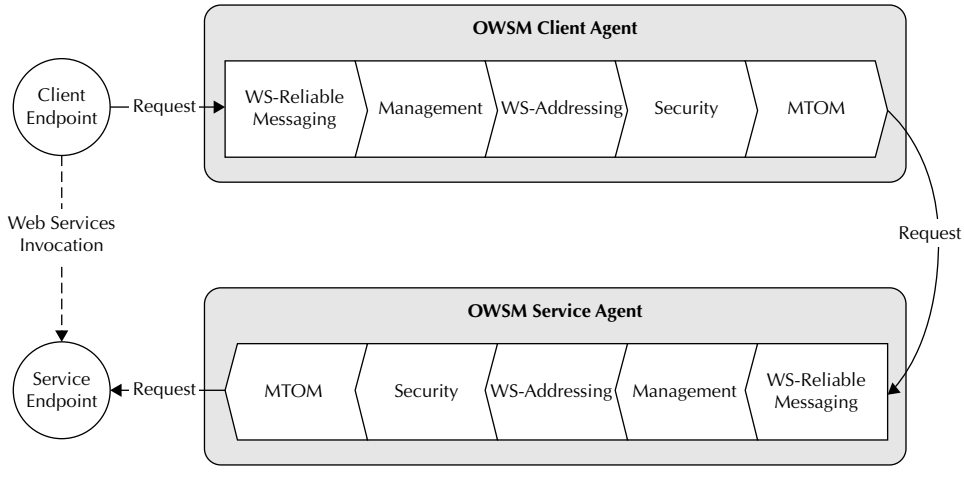
FIGURE 3-1. *Oracle Web Services Manager architecture*

Policy Execution Flow

As it should be apparent from our discussion in this section so far, each OWSM agent is essentially a policy execution engine that performs a web service's attached policies for all its invocation. The order in which a set of policies associated with a web service endpoint are executed depends on the policy's category and the endpoint type. For services endpoints, the order of precedence used by OWSM agents is as follows:

1. MTOM
2. Security
3. WS-Addressing
4. Management
5. WS-Reliable Messaging

On the other hand, the order of precedence for the execution of client endpoint policies is exactly the reverse, with Reliable Messaging policies being executed first and MTOM policies last. For security policies, where it is possible to have both authentication and authorization policies associated with the same endpoint, the authentication policies are always executed prior to the authorization policies because the authentication of the request's subject is required for the processing of authentication policies. The following diagram illustrates the OWSM policy execution flow between a client and service endpoint.



The execution of a policy by an OWSM agent translates into the execution of its individual assertions. The order in which policy assertions are executed is the order in which they are specified.

Runtime Configuration

A number of parameters that control the runtime behavior of the OWSM policy manager and the domain's PEPs and agents are configurable through the Enterprise Manager platform policy configuration page (accessed through the domain node's Web Services | Platform Policy Configuration option). A detailed discussion of the configuration options provided by this page are beyond the scope of this book. We will, however, highlight two important configuration properties related to the caching behavior of the PEP instances running on a domain's servers. The first property is called cache.tolerance and indicates the polling interval for the PEP polling the policy manager in order to refresh the server's policies. The default value for this parameter is 60000 (1 minute). The second property is called cache.size, and it indicates the maximum number of policies that can be stored within the policy cache. Both of these parameters are modified through the policy configuration page's Policy Cache tab.

Chapter Use Case

In this chapter's use case, we pick up where we left off with the Chapter 2 use case. We begin by extending the pricing application with the Fusion Middleware common infrastructure in order to front it with an Oracle HTTP Server that is managed as part of a farm. We then continue by protecting the web service of the pricing application using Oracle Web Services Manager WS-Security policies.

Use Case Description

The evaluation of the pricing application by the specialty products business development team has gone well, and as a result the group would like to roll out the application for use by the company's entire sales force. Given that you had deployed this application only for evaluation purposes up until

now, you have worked out a plan for its deployment for wider use within the enterprise and have been assigned a project by your management to execute on this plan. The two main milestones of your project are as follows:

- Integrating the pricing application with the company's identity management infrastructure through the use of the company's enterprise LDAP server as the identity store and the company's Oracle Access Manager (OAM) as the Single Sign-On (SSO) provider.
- Integrating the pricing application web service interface with the company's Enterprise Resource Planning (ERP) system through the use of Oracle SOA Suite.

To lay the groundwork for these milestones, you should begin by enhancing the existing pricing application's domain—which is currently a vanilla WebLogic Server–based domain because it was not built using any Fusion Middleware product domain configuration template—with the Oracle SOA Suite template. The enhancement of the domain will allow you to front the application with a centrally managed Oracle HTTP Server (OHS) instance, which is needed for integration with OAM. The domain enhancement will also allow you to protect the application's web services endpoint with an appropriate OWSM WS-Security policy so that it can only be accessed by the company's ERP system. In this chapter's use case, we will go through the details of accomplishing these first steps in achieving the two project milestones identified for rolling out the pricing application for use by the company's sales force.

In the following sections we outline the steps required to extend the pricing application's domain with Oracle SOA Suite, front it with an OHS instance managed by the domain, and secure its web service through OWSM. After the completion of these steps, the evaluation topology we created in Chapter 2 (and as shown in Figure 2-5) will be modified to resemble the one shown in Figure 3-2.

In the next two chapter's use cases, we will outline the remaining steps required to complete the two milestones and ensure that the pricing application is ready for use by its enterprise audience.

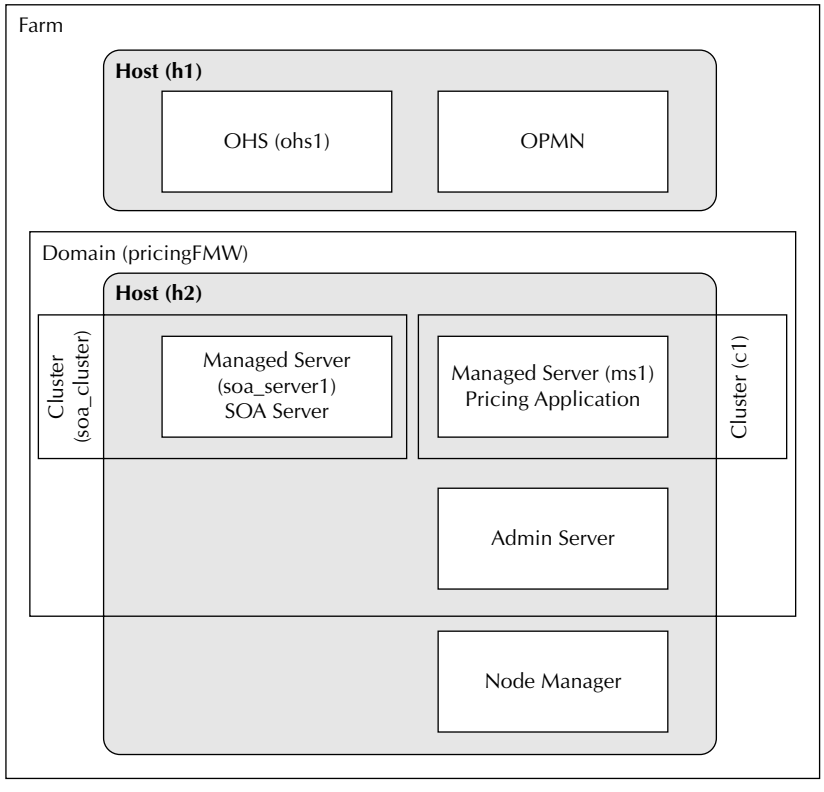


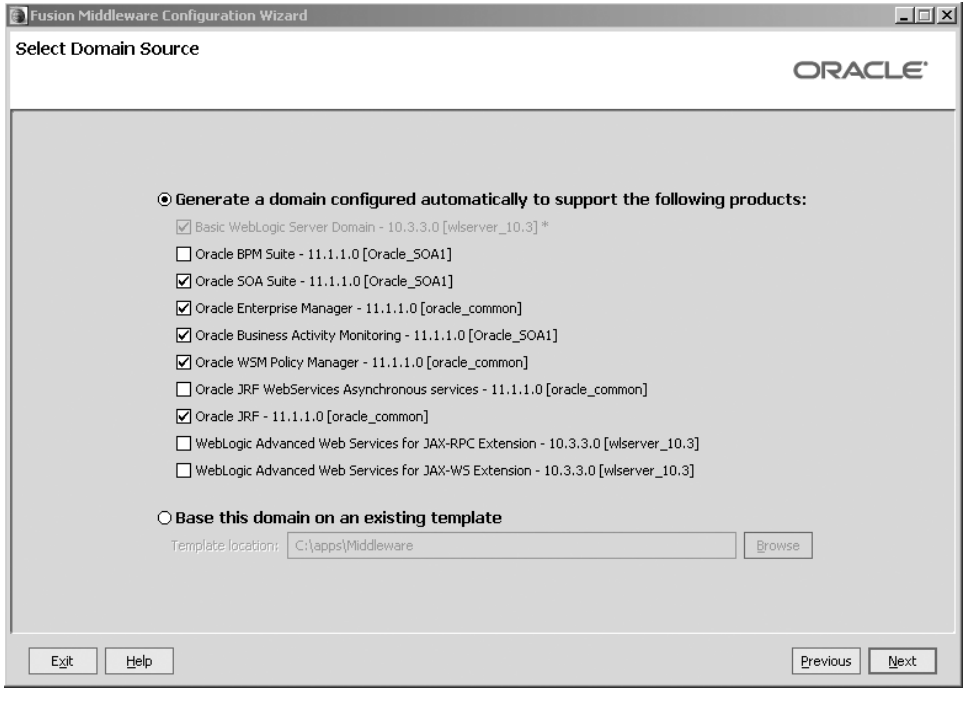
FIGURE 3-2. *Pricing application's intermediate topology for company-wide rollout*

Creating the Extended Application Domain

The extended pricing application domain will need to be an Oracle SOA Suite–based domain because it will need to host SOA Suite deployment artifacts that will be integrating the company's ERP system with the pricing application's web service. Having both SOA Suite and the pricing application as part of the same domain will simplify the administration of the pricing application because they will be manageable in the context of a single farm and therefore a single Enterprise Manager, admin console, and WLST scripts. We will work on deploying the SOA Suite artifacts in Chapter 5,

but in this chapter we begin the required configuration work by building a SOA Suite domain that also has a dedicated managed server for hosting the pricing application.

To create this domain you should begin by installing Oracle SOA Suite 11*g*R1 within the same MW_HOME directory where your existing pricing application's WebLogic Server installation resides. This installation also lays down the domain configuration templates for SOA Suite and the Fusion Middleware common infrastructure templates on which they depend. Prior to using these templates to begin the domain creation process, you will need to download and use the RCU to create the schemas required by SOA Suite and the Fusion Middleware common infrastructure. Once the schemas have been created, you can use the WebLogic Server Configuration Wizard to create a SOA Suite domain (called pricingFMW) by selecting the Oracle SOA Suite (this leads to the automatic selection of the OWSM and JRF templates as dependencies), Oracle Business Activity Monitoring, and Enterprise Manager templates, as illustrated next.



Note that the configuration wizard will prompt you for information regarding the schemas you created using the RCU in order to create the associated JDBC data sources within the domain. By default, the SOA Suite templates we have selected lead to the creation of two managed servers—named soa_server1 and bam_server1—to which are deployed the required SOA Suite deployment archives we will review in detail in Chapter 5. As per the best practice we discussed in the previous chapter's use case, you should also ensure that each of these managed servers is assigned to its own cluster so that they can more easily be scaled out in the future if the need arises. Once you have completed the execution of the configuration wizard, you will have a domain suitable for the deployment of SOA Suite applications, but is not yet configured for the deployment of the pricing application itself. Because you already have a domain with all the required resources configured for the pricing application—that is, the domain we created as part of the use case for Chapter 2—ideally you would be able to just extend the SOA Suite domain you just created with your existing pricing application so as to skip the need for the manual modification of the SOA domain with the managed server and resources needed. The WebLogic Server configuration framework allows you to do this by creating a domain configuration extension template from your existing pricing application domain.

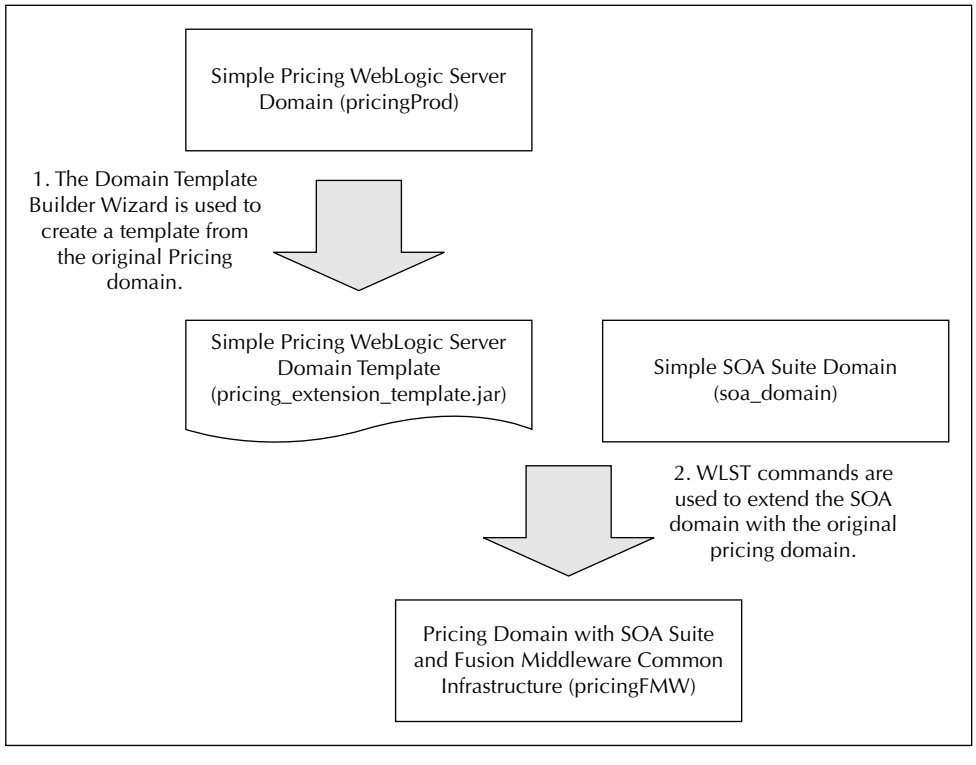
To do this, you need to begin by using the WebLogic Server Domain Template Builder Wizard—which allows you to create domain configuration templates from an existing domain—to create a domain extension template from your existing pricing domain (let's call this template pricingextensiontemplate.jar). The template builder wizard comes in handy because it allows you to easily specify the extra users, groups, and roles required by the domain. For the purpose of the pricing application, this allows you to ensure that the application's required "sales" role is specified within the template extension you are creating. With the domain extension template for your original pricing domain in hand, you are now ready to extend the SOA domain you created earlier. This will lead to the addition of the pricing application, its managed server, cluster, and other required WebLogic Server resources such as the applications required JDBC data sources and JMS modules. The WLST script for the creation of the domain will look as follows:

```
readDomain('pricingFMW');
addTemplate('./pricing_extension_template.jar');
updateDomain();
```

Although the domain created by this script can be used for the deployment of your pricing application, there is one more task you will need to perform to ensure that you can take full advantage of the Fusion Middleware infrastructure stack—and that is to ensure that the pricing application's target cluster (c1) is extended with JRF by using the "applyJRF" WLST command, as follows:

```
applyJRF('c1','/domains/pricingFMW');
```

This command ensures that the JRF components are appropriately deployed and targeted to the "c1" cluster and its single managed server so that they can be appropriately managed through the farm's Enterprise Manager. With the preceding steps you have developed a simple and repeatable way of creating a pricing domain that not only has all of the Fusion Middleware common infrastructure components but also the Oracle SOA Suite components enabled. The following diagram illustrates the high-level steps we discussed in this section in order to create this domain.



In the next section we discuss how to extend this domain with a fronting OHS instance that can be centrally managed through the domain's Enterprise Manager.

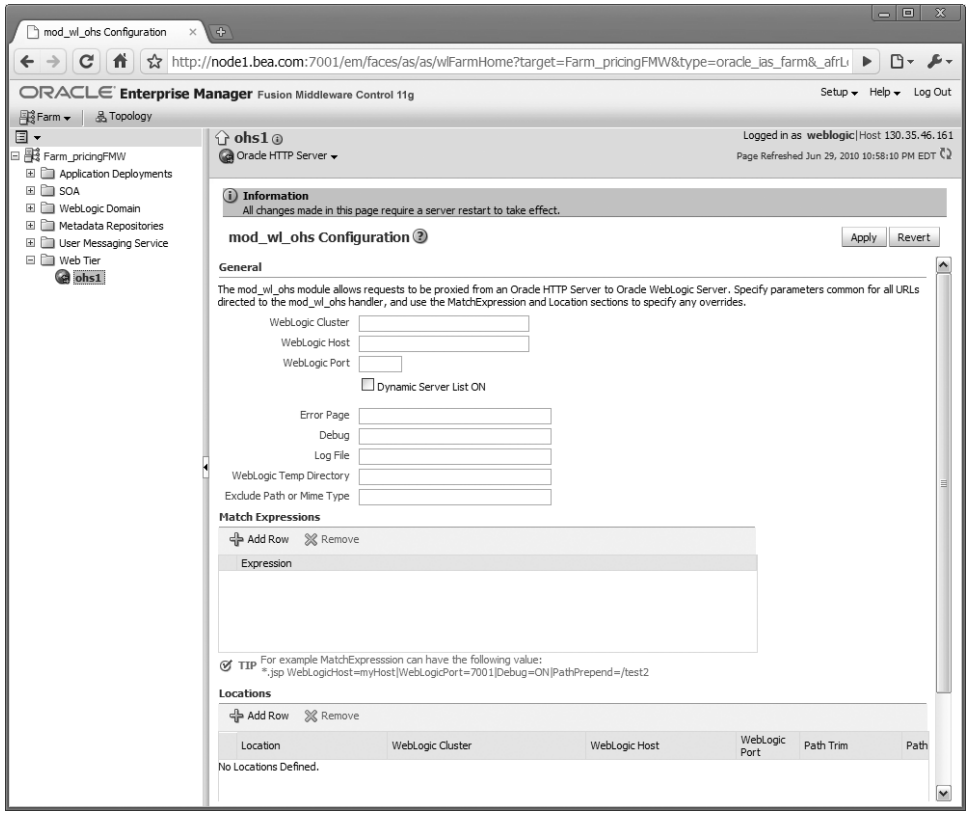
Fronting the Application with Oracle HTTP Server

In this section we complete the creation of the topology illustrated in Figure 3-2 by configuring an OHS system component that is managed as part of the same farm as the pricing application's domain and that proxies requests to the pricing application's web application. The registration of the OHS system component with our pricing application's farm will allow us to simplify the administration of the application's environment through central monitoring and configuration of the OHS instance with the farm's Enterprise Manager.

To configure an Enterprise Manager–managed OHS instance, you should begin by running the Oracle Fusion Middleware Web Tier OUI installer and selecting the Oracle HTTP Server option as well as the Associate Selected Components with WebLogic Domain option. Because you are installing OHS on its own, you need to install the OHS ORACLE_HOME within a new MW_HOME directory. To register the OHS instance that you are configuring with the "pricingFMW" WebLogic Server domain you created in the last section, you should specify the domain's administration server information as part of the Specify WebLogic Domain window and proceed to create an instance called "pricingwebtier" with an OHS system component called "ohs1." After completion of the ohs1 system component, the OHS Apache processes as well an OPMN process manager process should be running on your host, and you should be able to use the opmnctl command within the INSTANCE_HOME/bin directory of the ohs1 system component to start and stop its processes. Also, if you navigate to the Enterprise Manager page of the farm, you should notice that the ohs1 instance has been added to the bottom of the management tree.

To make sure those OHS proxies send requests to the pricing web application appropriately, you need to configure a module known as mod_wl_ohs. This is a WebLogic Server–specific module that is prepackaged and preconfigured with OHS for convenience. The mod_wl_ohs module can be configured to proxy requests to backend WebLogic Server deployed web applications. To configure mod_wl_ohs, you can right-click on the ohs1

node within Enterprise Manager and select the Administration | mod_wl_ohs Configuration option. This should bring you to the OHS configuration page, shown here.



To ensure that the mod_wl_ohs is configured to proxy incoming requests targeted at the pricing web application to the domain's managed server it has deployed, you need to add a row to the "Locations" section of the Enterprise Manager's mod_wl_ohs configuration page and fill in the following values:

- **Location** This field identifies the context root portion of the request URLs that need to be proxied.

- **WebLogic Host** This field identifies the host where the WebLogic managed server containing the target application is running.
- **WebLogic Port** This field identifies the listen port of the managed server specified in the WebLogic Host field to which requests need to be proxied.

Completion of these fields through Enterprise Manager will lead to the modification of the OHS instance's INSTANCE_HOME/config/OHS/ohs1/mod_wl_ohs.conf file with the following Apache configuration directives, using sample values:

```
<Location /PricingServiceWeb>
  SetHandler weblogic-handler
  WebLogicHost node1.oracle.com
  WebLogicPort 7002
</Location>
```

For a basic proxy setup through OHS, all you need are these fields. After filling in this information and restarting the OHS server through the Enterprise Manager's "control" options, you should be able to access the pricing web application by pointing your browser to the host on which you have configured OHS and using port 7777, which is the default port the OHS listens to. Using the example from the directives listed here, to access the pricing application's URL you would specify node10.oracle.com:7777/PricingServiceWeb (assuming that OHS was configured on node10), even though the actual application might be deployed on a managed server on host node1.oracle.com and listening to port 7002.

Securing the Application Web Service with Oracle Web Services Manager

The web services interface of the pricing application exposes a single operation (modifyModel), which is used to modify the pricing model for a given product. Model modifications are driven by business parameters that have their sources within the company's ERP system. Given the knowledge that the sole client of this web service will be a SOA Suite application that will be interacting with the ERP system in order to detect changes to these business parameters (we will work through the deployment of this SOA application in Chapter 5's use case), you can protect the web service

endpoint so that only the authenticated SOA composite can invoke it. This will prevent unknown users from pointing a web services client to the web service's WSDL to perform "modifyModel" operations.

To secure the application's web service endpoint, you can use the OWSM prepackaged wss_username_token_service_policy and binding_permission_authorization_policy. The first policy ensures that incoming requests have an appropriate WS-Security user name and password header and that the value of these headers is authenticated through the WebLogic Server authentication providers in order to obtain an authenticated subject. The second policy ensures that the invocations to the web service will lead to an authorization check to the domain's Oracle Platform Security Services (OPSS) policy store, where you will need to configure the authorization permission for this web service's access. We will discuss OPSS and the way to create the required permissions for this web service in the next chapter's use case. An important note to make is that even though the policies you have configured do not require SSL encryption of the message content—including the user name and password—you should be comfortable with this choice for the following reasons:

- The open text values of the modifyModel operation contain only numerical parameters that do not reveal any sensitive information that would otherwise have to be protected.
- The credential information being propagated will always be dedicated to the SOA composite and not associated with a specific individual.
- The service invocation will be occurring between company servers behind firewalls; therefore, the web service's request traffic will not be accessible to any external entities.
- The absence of SSL for the web service endpoint will simplify the architecture of the pricing application and will lead to a more efficient execution of the pricing service.

You can attach these policies through Enterprise Manager or by using the attachWebServicePolicies WLST online command. Once it's attached, you can check the WSDL of the pricing service web service (through its http://<host-name>/PricingModelWebService /PricingModelService?WSDL URL)

and should notice that its definition has been prefixed with the WS-Policy WssUsernameToken10 definition shown in the following listing. This policy is then associated with the service through the WSDL's binding definition.

```
<wsp:Policy xmlns:wsp=" … ">
   <sp:SupportingTokens
xmlns:sp="http://schemas.xmlsoap.org/ws/2005/07/securitypolicy">
     <wsp:Policy>
         <sp:UsernameToken sp:IncludeToken="http://
schemas.xmlsoap.org/ws/2005/07/securitypolicy
/IncludeToken/AlwaysToRecipient">
            <wsp:Policy>
               <sp:WssUsernameToken10/>
            </wsp:Policy>
         </sp:UsernameToken>
      </wsp:Policy>
   </sp:SupportingTokens>
</wsp:Policy>
```

Although ideally you would proceed to test the web service through the Enterprise Manager web services test page, we cannot go through that step at this point because no OPSS authorization permissions have been associated with this web service despite its binding_permission_authorization_policy permission. In the next two chapter's use cases we will complete the work required to achieve the two milestones defined at the beginning of this section—namely, to roll out a version of the pricing application that is integrated with the enterprise's Oracle Fusion Middleware IDM infrastructure as well as the corporate ERP system through Oracle SOA Suite.

Conclusion

In this chapter we discussed the most important aspects of the Fusion Middleware common infrastructure through an overview of how it is installed, configured, and managed within a Fusion Middleware environment. We also discussed the Metadata Repository Services and Oracle Web Services Manager services, which are key components of this infrastructure. In the next chapters we begin our exploration of the specific layered Fusion Middleware products that are built on top of the common infrastructure we discussed in this chapter.