

Oracle WebLogic Server: A Solid Foundation for Service-Oriented Architecture

An Oracle White Paper
Updated June 2008

Oracle WebLogic Server: A Solid Foundation for Service-Oriented Architecture

Introduction	3
Oracle and the Java Platform, Enterprise Edition 5	4
Better Business Logic for SOA	5
EJB 3.0 Eliminates Chores	6
Annotations Do the Work.....	6
Deployment Descriptors Are Simplified.....	7
Oracle WebLogic Server Implements EJB 3.0 with Pitchfork	7
Streamlined Persistence for SOA.....	8
Support for Both JPA and JDO	9
High-Performance Implementation.....	9
Oracle TopLink	10
Improved Web Services Processing	11
Improved Web Interfaces	12
Industrial-Grade SOA Deployment and Management.....	12
Conclusion.....	14

Oracle WebLogic Server: A Solid Foundation for Service-Oriented Architecture

INTRODUCTION

Service-oriented architecture (SOA) has sparked an IT revolution. Deploying coherent packages of software functionality as loosely coupled, coarse-grained services delivers dramatically improved application flexibility, allowing enterprises to continuously adapt constellations of services to keep IT capabilities aligned with business goals.

Oracle is the leader in helping enterprises realize the benefits of SOA with Java. With Oracle WebLogic Server, Oracle provides a solid foundation for SOA based on the Java Platform, Enterprise Edition 5 (Java EE 5). Oracle WebLogic Server is extremely easy to use right out of the box and delivers industrial-grade reliability, availability, scalability, and performance. Customers can rapidly upgrade existing services and manage them with powerful configuration, deployment, and management tools. They can also leverage integration with other Oracle Fusion Middleware products as well as their developers' experience with open source technologies such as the Spring Framework.

This paper discusses how Java EE 5 dramatically speeds the development of SOA applications and how enterprise developers can leverage its power by using Oracle WebLogic Server.

Existing customers of Oracle WebLogic Server already value Oracle's commitment to supporting mission-critical SOA initiatives. However, the latest version of Oracle WebLogic Server offers even more from this proven platform: customers can build services more quickly, compose them more easily, and manage them more effectively. For organizations that want industrial-grade capabilities for their developers, this solution is the clear choice as the most modern, robust, and powerful Java-based SOA platform.

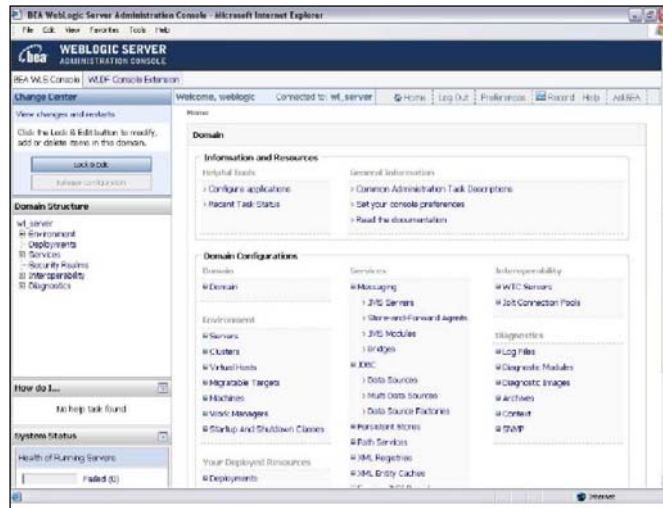


Figure 2: Oracle WebLogic Server

BETTER BUSINESS LOGIC FOR SOA

In SOA, different services can operate at different levels of abstraction. For enterprises, top-level services, which usually correspond closely to tasks in the business domain, might include “check credit score” in the loan-processing domain, “provision account” in the mobile telephony domain, and “submit invoice” in the accounts receivable domain. Obviously, implementing such services requires some kind of software model for the corresponding domain.

EJB 3.0 completely insulates developers from the plumbing. In large part, EJBs can be treated as regular Java objects, dramatically simplifying development. With this streamlined approach, developers can finally use EJB to implement a wide variety of domain models that support top-level enterprise services.

The original goal of EJB was to provide the infrastructure for building rich domain models that could support sophisticated business processes. Entity beans manipulated data and implemented the behavior of domain entities such as loan application, customer, and purchase order. A portfolio of session beans coordinated the interactions of entities to execute each domain's processes—for example, “evaluate loan,” “enroll customer,” and “pay suppliers.”

In many cases, earlier versions of EJB proved too complex to support this approach. The API exposed too many of the mechanisms necessary to ensure the robust execution of domain processes. Building effective domain models typically requires many trips through a cycle of prototyping, testing, and refinement. Forcing developers to worry about the underlying “plumbing” during this cycle made EJBs too cumbersome for many projects.

In contrast, EJB 3.0 completely insulates developers from the plumbing. In large part, EJBs can be treated as regular Java objects, dramatically simplifying development. With this streamlined approach, developers can finally use EJB to implement a variety of domain models that support top-level enterprise services.

EJB 3.0 Eliminates Chores

The changes in EJB 3.0 primarily affect how developers interact with container facilities. With the exception of persistence, these changes do not significantly affect the facilities themselves.

In previous versions of EJB, developers had to perform onerous chores to interact with the container. The first big chore was implementing all necessary interfaces, which required creating home, local, and remote interfaces, as well as implementing the interface corresponding to the type of EJB. For the home and remote interfaces, a developer needed to handle all necessary exceptions. Entity Beans also required finder methods. The final step was implementing all lifecycle methods for the applicable EJB interface.

The second big chore was writing Java Naming and Directory Interface (JNDI) lookups to acquire resource references. Rich domain models naturally have many relationships among their elements. Each relationship, as well as any reference to infrastructure resources, required lookup code.

EJB 3.0 eliminates both of these chores. Developers write EJBs as plain old Java objects (POJOs). They interact with container facilities by adding simple, declarative annotations to the POJO code. The container takes care of the necessary chores, allowing developers to focus on building domain models.

Annotations Do the Work

As mentioned, annotations are the key innovation that improves the developer experience of EJB 3.0. Oracle helped pioneer annotations and has worked to broaden their use within Java EE. If a developer can unambiguously specify what is to be done, why not just take care of it automatically? For example, suppose a developer working in the loan processing domain wants to write a client for a stateless session bean that performs loan processing. Instead of all the interface implementation and JNDI lookup code necessary with previous versions of EJB, the developer would simply write

```
import loanprocessor.LoanProcessor

@Stateless public class LoanProcessorClient {

@Inject LoanProcessor

...

}
```

Java EE 5 includes corresponding annotations for other types of EJBs. In addition, instead of implementing lifecycle methods, developers can create new instances as with any other POJO. For entity beans, there are even annotations for the common case of specifying an automatically generated identifier as the primary key, and then performing lookups using this key. Beyond controlling EJBs, Java EE 5 also includes a variety of annotations for simplifying access to security, persistence, and Web services facilities.

The `@Stateless` notation takes the place of manually defining the remote interface while the `@Inject`-annotation takes the place of a manual JNDI lookup.

Java EE 5 includes corresponding annotations for other types of EJBs. In addition, instead of implementing lifecycle methods, developers can create new instances as with any other POJO. For entity beans, there are even annotations for the common case of specifying an automatically generated identifier as the primary key, and then performing lookups using this key. Beyond controlling EJBs, Java EE 5 also includes a variety of annotations for simplifying access to security, persistence, and Web services facilities.

Deployment Descriptors Are Simplified

As if the programming chores associated with previous versions of EJB weren't enough, developers also had to contend with complicated deployment descriptors. After doing the chores and writing the actual business logic, deploying and running an EJB required writing a deployment descriptor using XML. For session beans, the descriptor contained mostly redundant information such as the class, the names of associated interfaces, and the type of EJB. Then there were directives to infrastructure services such as transaction management and security. For entity beans with container-managed persistence, the deployment descriptor might contain dozens of entries specifying its abstract data schema and various queries against that schema.

Most production implementations, such as Oracle WebLogic Server, included tools for generating deployment descriptors and putting them in the correct location. But these became another item for the developer to worry about, instead of focusing on the business problem. In EJB 3.0, deployment descriptors are optional. A developer can write and execute any type of EJB without a descriptor, with the annotations and a set of defaults providing enough information to execute the EJB. And a developer who does want to specify a descriptor needs to specify only those entries that override the defaults. In EJB 3.0, implementing a sophisticated service that uses a group of cooperating EJBs requires far fewer files, each with far fewer entries.

Oracle WebLogic Server Implements EJB 3.0 with Pitchfork

As noted earlier, EJB 3.0 does not significantly change the facilities provided by EJB containers; instead, it just makes these facilities much easier to use. Oracle WebLogic Server implements EJB 3.0 as an extension to its proven container—one that provides the intelligence necessary to run more-streamlined EJBs by translating annotations into specific instructions and resolving declaratively specified dependencies.

To create this extension, Oracle worked with the developers of the open source Spring Framework to create Pitchfork, a special version of Spring—a popular framework that helped pioneer simplification of Java application development through the use of dependency injection. Adding Pitchfork to Oracle WebLogic

Adding Pitchfork to Oracle WebLogic Server, the EJB container creates a best-of-breed solution: a proven industrial-grade container and productivity-enhancing framework.

Server, the EJB container creates a best-of-breed solution: a proven industrial-grade container and productivity-enhancing framework.

Two side effects of this approach are backward compatibility and forward thinking. Because Pitchfork acts as a broker between EJB 3.0 code and traditional container facilities, Oracle WebLogic Server is fully backward compatible with EJB 2.1. EJBs simply bypass Pitchfork and use container facilities directly. And customers can run EJB 3.0 and EJB 2.1 side by side. In addition to backward compatibility, Pitchfork facilitates forward thinking. Many developers have adopted advanced programming approaches such as aspect-oriented programming using Spring. Pitchfork allows those same approaches with EJB 3.0.

STREAMLINED PERSISTENCE FOR SOA

By shielding developers from complexity, EJB 3.0 allows them to focus on building representative domain models. Persistence is the next challenge. Executing business processes requires manipulating the data of record. The loan application, customer, and purchase order entities mentioned previously as examples all have Java representations that execute in the EJB container. However, performing real work requires loading and saving the corresponding representations stored in back-end databases, which ensure that there is only one “true” copy of each fine-grained unit of data and that different services manipulating the same units of data do not interfere with each other.

In theory, a multitier approach such as SOA makes the application layer independent of the data layer. In practice, a large portion of application layer code gets devoted to data management tasks. The fundamental problem rests in how the application layer adds value. Most services provide reasonably unique value within a particular domain—they execute an assigned set of tasks within broader processes. This specificity requires them to manipulate business entity data in a particular way. Unsurprisingly, they want to arrange the required data in the most convenient form.

Therefore, each service must map fine-grained units of back-end data to its coarser-grained entity models. Manually writing and debugging the mapping code is extremely time-consuming and error-prone. Automated mapping approaches are not a panacea because they trade off flexibility and complexity. Not enough flexibility means that developers must manually write a lot of code to meet the requirements. Due to excess complexity, developers feel as though using the automated tool is equivalent to writing a lot of code. Based on a large body of Oracle’s field experience, Oracle WebLogic Server incorporates Oracle Kodo, which seamlessly integrates both JPA and Java Data Objects (JDO). With Oracle WebLogic Server, developers can choose the optimal mechanism for particular domain models.

Based on a large body of Oracle’s field experience, Oracle WebLogic Server incorporates Oracle Kodo, which seamlessly integrates both JPA and Java Data Objects (JDO). With Oracle WebLogic Server, developers can choose the optimal mechanism for particular domain models.

Support for Both JPA and JDO

As discussed previously, EJB 3.0 uses annotations to reduce the coding of interface implementations and entries in deployment descriptors. The advantage of the latter is most clear for entity beans with container-managed persistence. Abstract coding plus verbose deployment descriptors made for cumbersome usage in EJB 2.1. Moreover, although EJB 2.1 made some effort to make its persistence facilities independent of database technology, it tended to make the typical case of using relational databases far more complicated than necessary.

The ease of use and functional enhancements that come with JPA give developers the ability to quickly implement persistence for most common SOA data access requirements.

JPA addresses all these shortcomings. Developers simply provide code annotations instructing the container how to access the appropriate relational database data. An `@Id` annotation before a set of accessor methods defines the primary key. There are even annotations for specifying composite keys. An `@OneToMany` annotation before a collection definition defines a one-to-many relationship. For many-to-many relationships, an `@JoinTable` annotation provides the means to set up a join table. More-sophisticated annotations such as `@Transaction` and `@NamedQueries` include a large set of attributes for precisely controlling specialized interactions with relational databases.

JPA does more than just simplify persistence features from EJB 2.1. It also includes much-needed enhancements. Most important, developers no longer have to specify lifecycle methods for entity beans. The container automatically provides an `EntityManager` object for controlling the lifecycle of instances. JPA also enables Java classes to specify strategies for handling inheritance when mapping to the database. Compared to the EJB query language from 2.1, the JPA query language includes several new features, such as bulk operations, outer joins, and subqueries. Together, the ease of use and functional enhancements give developers the ability to quickly implement persistence for most common SOA data access requirements.

Despite JPA's improvements, many developers will still want to use JDO. Because JPA focuses on relational database persistence, it adopts a relational style. When working with advanced object-oriented models, JDO's more object-oriented syntax could be attractive. In many cases, the choice will simply be a matter of taste. With Oracle WebLogic Server, developers can use container-managed persistence with JPA or bean management persistence with JDO. While JPA and JDO share much underlying functionality, each does have capabilities the other lacks. Oracle WebLogic Server provides extensions to both APIs, making them functionally equivalent. Developers do not have to sacrifice capabilities when choosing one or the other.

Oracle's commitment to building high-value services with Java EE 5 goes beyond giving developers a choice of persistence API. It includes delivering high performance no matter which alternative a developer prefers.

High-Performance Implementation

Oracle's commitment to building high-value services with Java EE 5 goes beyond giving developers a choice of persistence API. It includes delivering high performance no matter which alternative a developer prefers. As mentioned, the same persistence engine executes both JPA and JDO functions.

Oracle Kodo includes a long list of industrial-grade features. Perhaps the biggest challenge in providing persistence for top-level business services is supporting large, long-running transactions. Completing a significant step in a business process can result in extensive updates to many different pieces of data. Oracle Kodo supports transactions of unlimited size. Guaranteeing the coordination of complex business processes can require transactions that remain in progress for minutes, hours, or even days. Oracle Kodo efficiently manages connections to datasources during such long-running transactions.

ORACLE TOPLINK

Also included with Oracle WebLogic Server is an alternative high-performance persistence technology—Oracle TopLink. The solution is a commercial-grade release and superset of TopLink Essentials. Oracle, as a cospecification lead for the new EJB 3.0/JPA, helped design and provide architecture guidance for the new JPA specification. In addition, Oracle contributed the TopLink Essentials code for the JPA reference implementation. TopLink Essentials is now open source code.

Also included with Oracle WebLogic Server is an alternative high-performance persistence technology—Oracle TopLink. The solution is a commercial-grade release and superset of TopLink Essentials.

When included with Oracle WebLogic Server, Oracle TopLink includes advanced object-relational mapping (ORM) capabilities beyond those in TopLink Essentials. These include coordinated caching to support clustered application deployments and additional nonintrusive, optimistic locking policies. Oracle TopLink provides platform-independent, stored procedure and function support; it enables historical mapping and point-in-time querying. Java Management Extensions (JMX) managed beans (MBeans) allow for management and monitoring of Oracle TopLink sessions and their caches. When working in an Oracle Database environment, Oracle TopLink provides the following features:

- A virtual private database
- XML-type mapping and SQLX query generation
- Hints
- Hierarchical querying
- ORMs, arrays, structures, object references, and nested tables
- Custom line of business, time stamp, and double-byte datatypes

Finally, Oracle TopLink supports mapping to executive information systems using Java Connector Architecture resource adapters. It offers object-XML mapping based on Java Architecture for XML Binding (JAXB) 1.0 and provides early

support for JAXB 2.0 functionality. Because Oracle is committed to a hot-pluggable architecture, developers can choose to use the persistence technology that is right for their project.

IMPROVED WEB SERVICES PROCESSING

EJB 3.0's use of regular Java objects makes it quick and easy for developers to build rich domain models and connect them to back-end databases. Of course, these models must then be made available as services within the larger business processes supported by the enterprise SOA. Seamless cooperation among all services requires a solid foundation for Web services protocol processing.

There are two primary protocol processing issues. First, interoperability requires implementation of an entire protocol stack and verified compatibility with stacks from different vendors. Oracle WebLogic Server includes support for the latest security- and performance-oriented Web services (WS) protocols. In the security area, updates support WS-Security 1.1 and WS-SecurityPolicy 1.1 and 1.2. There is new support for WS-Trust and WS-SecureConversation, which enable services to establish a shared security context and maintain long-term trusted relationships.

In the performance area, there is new support for the XML-binary Optimized Packaging (XOP) and Message Transmission Optimization Mechanism (MTOM) specifications. XOP lets services transmit binary data as is, without base64 encoding, and put it in the same MIME package as the rest of an XML document. This approach reduces both required bandwidth and encoding overhead. MTOM describes how to use XOP to optimize the transmission of Simple Object Access Protocol messages. Oracle has participated in several interoperability events to ensure that its implementations of these protocols, as well as others in the Oracle WebLogic Server stack, work well with those of other vendors.

The second protocol processing issue is more subtle. The Java API for XML-Based Remote Procedure Call (JAX-RPC) processing API from past versions of Java EE supports only the RPC style, which, unfortunately, is the least flexible of all Web services interaction styles. Java EE 5 introduces a new processing API—Java API for XML Web Services (JAX-WS)—which supports the more-flexible document-oriented style. Oracle WebLogic Server also provides basic support for a third style—Representational State Transfer—which can simplify certain interactions. The processing infrastructure for multiple Web services styles allows developers to tailor interactions within an SOA to meet different enterprise requirements.

Although not strictly a protocol processing issue, client-only Web services stacks are a practical issue for customers who want to build lightweight applications that act only as consumers of services, without the burden of server-oriented functionality. Oracle WebLogic Server provides a special Java library that implements only the client portions of the protocols defined in the Web Services Interoperability Organization Basic Profile.

Oracle WebLogic Server includes support for the latest security- and performance-oriented Web services protocols.

IMPROVED WEB INTERFACES

Nearly all business processes supported by an SOA require some form of user interaction, and most processes involve a great deal of it. Top-level SOA services tend to provide a natural representation of business tasks. Leveraging this representation to provide more-sophisticated user interfaces is a tremendous opportunity. Java EE 5 provides an integrated package of enhancements to its Web interface APIs. These enable more-sophisticated user interactions, make interface programming easier, and expand the breadth and depth of how people interact with services.

The core of the Java EE 5 Web interface APIs is JavaServer Faces (JSF). The primary goal of JSF is to make developing Web interfaces more like developing GUIs. The model underlying the framework enables developers to connect interface capabilities to logic components, essentially creating abstract widgets. Interactions between user and widgets use an event-driven model. The final development step is binding the abstract interaction model to a specific technology such as the Web.

This interaction style makes it very easy to handle interface state within the framework. An immediate benefit is that developers do not have to implement a lot of state-related Java and Expression Language (EL) code in their JavaServer Pages (JSPs). JSF can also gracefully handle value conversion and validation. And because JSF uses the same model as a GUI, it works very well with interface development tools. As traditional client/server developers learned a long time ago, building an interface with a good tool is far more productive than coding it by hand.

Now that JSF is part of Java EE, it works seamlessly with JSP. They share a unified EL, and JSP is the default rendering mechanism for JSF. More important, there is a tremendous opportunity to extend the user interface capabilities of the standard platform. JSF, combined with JSP tag libraries, makes it easy to create reusable user interface components. Moreover, most of the framework functions are plug and play. Developers can replace them piece by piece with enhanced versions, or can even swap out the entire framework with an alternative such as Spring.

User interfaces are the key to using the power of SOA. Making them easier to build and creating opportunities for rapid future enhancement lays the groundwork for an even better return on investment on SOA.

INDUSTRIAL-GRADE SOA DEPLOYMENT AND MANAGEMENT

By making it easier to write business services and their interfaces, Java EE 5 promotes a much richer SOA ecology. Actually building up this ecology within enterprises requires keeping individual service instances and the broader SOA environment healthy. An API specification alone cannot meet this requirement. Enterprises need an industrial-grade implementation—one with the deployment, management, and robustness required of any mission-critical IT component.

User interfaces are the key to using the power of SOA. Making them easier to build and creating opportunities for rapid future enhancement lays the groundwork for an even better return on investment on SOA.

Oracle WebLogic Server has proven its ability to operate in an enterprise environment. The latest version extends this success to cover Java EE 5, so customers can manage all the new features from the familiar console, while making it as easy as possible to upgrade existing services. Upgrading means simply redeploying an existing service to the new platform—no porting necessary.

Oracle WebLogic Server has proven its ability to operate in an enterprise environment. The latest version extends this success to cover Java EE 5, so customers can manage all the new features from the familiar console, while making it as easy as possible to upgrade existing services.

Enterprises also face the challenge of upgrading clients to new versions of a service. Several features ease this natural consequence of a mature SOA. Multiple versions of the same services can run on the same server or cluster. Administrators can segment version access by client population—for example, making newer versions available only to clients coming from local or internal network segments. Finally, administrators can declaratively specify the client migration policy.

Oracle WebLogic Server addresses several other customer issues. For enterprise networks that do not support multicast, the latest version offers unicast clustering. When running a cluster, migrating the Java Message Service and Java Transaction API services from one machine to another used to require several manual steps. The latest version features automatic service migration. It also enables recording and scripting of console operations.

As Oracle WebLogic Server becomes a standard part of the IT infrastructure, many enterprises want to monitor it as part of their Simple Network Management Protocol (SNMP) consoles. The latest version supports SNMP 3, including an SNMP view of JMX Runtime MBeans. Moreover, the SNMP agent for the administration server now provides a view for the entire Oracle WebLogic Server domain. These improvements demonstrate Oracle’s commitment to making SOA a benefit for the entire IT organization, not just architects and developers.

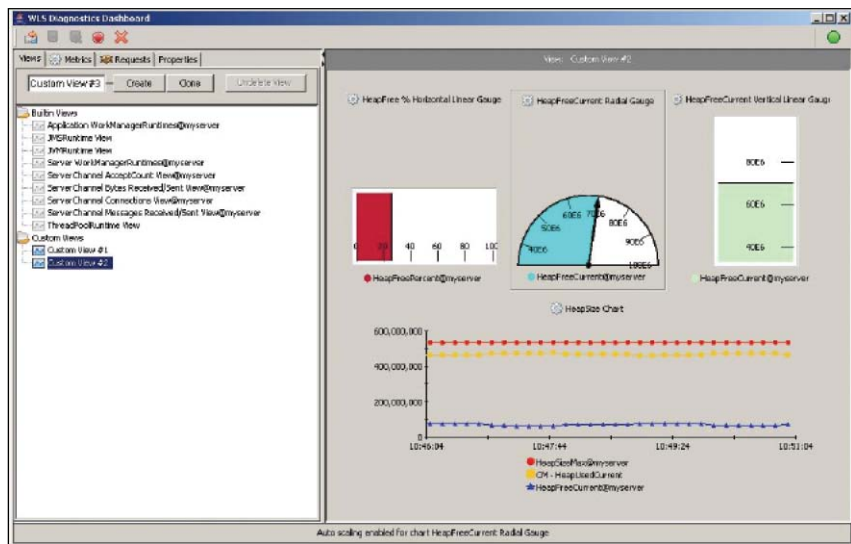


Figure 3: Oracle WebLogic Server diagnostic framework

CONCLUSION

Java EE 5 dramatically speeds development of SOA applications. EJB 3.0 simplifies business logic by enabling developers to focus more on the domain model and less on the middleware plumbing. JPA simplifies the most common persistence management tasks associated with mapping domain objects to back-end relational databases. The Web interface APIs enable richer and more-flexible user interaction with SOAs, while JAX-WS enables richer and more-flexible cooperation among services in an SOA.

Oracle WebLogic Server is the Java EE platform with the most popularity, productivity, and performance. Now it is one of the first to put the power of a production-ready Java EE 5 implementation in the hands of enterprise developers.

Oracle WebLogic Server is one of the first production-ready implementations of Java EE 5. It not only adheres to the letter of the standard, but embraces its spirit—simplicity without sacrifice. Developers get the advantage of a simplified API without giving up any of the proven infrastructure at the foundation of Oracle WebLogic Server. Oracle TopLink is also included, delivering a high-performance instance of the Java EE persistence reference implementation. Helping customers make their businesses run better is the ultimate goal, and Oracle WebLogic Server delivers Java EE 5 capabilities in the way that best supports it.

Oracle WebLogic Server is the Java EE platform with the most popularity, productivity, and performance. Now it is one of the first to put the power of a production-ready Java EE 5 implementation in the hands of enterprise developers. Existing customers can take immediate advantage of the improved APIs, while new customers get the confidence of alignment with a proven leader.



Oracle WebLogic Server: A Solid Foundation for Service-Oriented Architecture
Updated June 2008

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2008, Oracle and/or its affiliates. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change without notice.
This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.
Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners. 0408