

# Oracle SDP – Enablers and Future Proofing your Investment

*An Oracle White Paper  
April 2006*

# Oracle SDP – Enablers and Future proofing your Investment

## TABLE OF CONTENT

Executive Summary .....	3
SDP Overview .....	6
Enablers and an Enabler Framework.....	6
Enablers.....	7
Enabler Framework.....	8
Oracle SDP and Enablers .....	8
SIP Servlet container .....	8
Call control Enabler .....	9
Presence Enabler .....	10
Media Server Control .....	11
Messaging Enabler.....	11
Enablers and application development.....	12
Conclusion.....	12

# Oracle SDP – Enablers and Future proofing your Investment

## **EXECUTIVE SUMMARY**

The Oracle Service Delivery Platform is a J2EE realization of the OMA Service Environment. As such, it provides a standards-based approach to support the full life cycle of next generation data services, starting from design and development all the way through deployment, management and service upgrades. By using carrier-grade J2EE as the basis of the Service Delivery Platform, Oracle is able to bring the full richness of IT technology and concepts to solve a difficult and challenging problem faced by almost all operators and service providers.

One of these concepts that is at the heart of the Oracle Service Delivery Platform is the notion of enablers. Simply put, an Enabler is a building block that encapsulates reusable functionality, and is packaged so that this functionality can be easily integrated into applications as a building block. Enablers are key to standard IT (and Java EE) developers taking an interest in the development of network-centric applications. For example, a call-control enabler can make it easy to build a click-2-call application without the developer of that application having to understand the details of the underlying network signaling technology.

In addition to making it easier for a developer to integrate with core network functionality, Enablers also solve another vexing problem for operators and service providers – how to make services portable in the presence of convergence, heterogeneous networks and heterogeneous OSS/BSS systems. This is where Enablers truly come to life – by providing a set of standardized APIs that abstract network-oriented or OSS/BSS-oriented functionality, the service developer can minimize the dependence on specific network protocols or OSS/BSS systems. This is particularly valuable in an era when voice revenue is being commoditized and operators and service providers need to distinguish themselves by bringing interesting data services to market quickly and ahead of their peers in the industry. Enablers can significantly reduce the time it takes to develop a new service.

Finally, by defining the appropriate set of enablers that abstract underlying functionality (e.g., call control), it is possible to build services that are truly future-proof. The Oracle SDP provides a layered architecture whereby it is possible to provide the same northbound API (or interface) for a particular functionality that

can stay the same even as underlying network elements may evolve over time. The implementation of the enabler evolves (through a network-element specific adapter layer) to accommodate this evolution, but a service that has been built on the northbound APIs exposed by the enabler does not need to change. This layering of functionality can reduce service development and migration time, particularly in a time of aggressive network evolution and convergence between various network types (e.g., wired and wireless for example). As a result, operators and service providers can follow either a revolutionary or evolutionary approach to new service development, future-proof their investments in new services that are developed on enablers and support a strategy for services that includes both service co-existence as well as service migration.

## INTRODUCTION

Telco and IT technologies are rapidly converging. Confronted with huge infrastructure costs, Telecommunication service providers need to develop new high value services faster to recoup their investments. At the same time, they have to deal with eroding revenue streams from traditional services (e.g. voice) and fend off new aggressive competitors such as virtual service providers and other internet-based service providers that provide competitive services at very low costs. For service providers, it is essential to be able to rationalize their environment to build, deploy and manage services as well as facilitate the development of new attractive services by third party partner service providers.

An example of a new and exciting service is triple play (voice + data + video) combined with integrated billing from a single operator or a service provider to a consumer. The demand for such converged services, combined with the convergence between wireless and fixed networks (e.g., broadband) further emphasizes the need to move away from silo-based architectures and instead rely on a horizontal, standards-based platform that enables convergence – enter a standards-based Service Delivery Platform (SDP).

The industry at large has realized the need for an SDP, particularly one that conforms to industry standards and is built on a standard architecture blueprint. The Open Mobile Alliance (OMA), a federation of industry leaders, is in the process of standardizing an architecture blueprint for SDPs under the OMA Service Environment (OSE) effort. Oracle is a leader in this standardization effort and the Oracle Service Delivery Platform is a realization of the OSE based on Java EE (formerly known as J2EE).

Enablers are a core concept in the Oracle Service Delivery Platform – in short, Enablers are common, reusable building blocks of functionality that provide standard north-bound APIs (either Java or Web services based) and integrate with either the underlying network or OSS/BSS systems through an adapter framework to implement the functionality exposed by the APIs. Enablers are central to achieving reuse of common functionality. In addition, a service that is developed using the APIs exposed by the enablers is future proof, since it is possible to extend the Enabler functionality for future networks without compromising the APIs that are exposed. This combination of future-proofing and reuse of common functionality gives enablers their power and central role in the Oracle Service Delivery Platform.

The rest of this document is organized as follows: we first provide a brief summary of the components in the Oracle Service Delivery Platform – additional details can be found in the companion white paper *Carrier grade J2EE as the foundation of the Oracle SDP*. We then discuss the concept of enablers and a general purpose enabler framework. We discuss next some of the specific enablers that are in the Oracle SDP and conclude with the role of enablers in application development.

## SDP OVERVIEW

At its simplest, the Oracle Service Delivery Platform is a standards-based environment for the design, development, deployment, and management of services that lie at the heart of IT and network convergence. SDP is a natural extension of the Oracle Fusion Middleware suite into the service delivery arena. In particular, the SDP has the following features:

- **A carrier-grade execution environment** for services that access next-generation as well as traditional networks. The explicit design of the SDP that enables services to work on today's networks as well as tomorrows ensures future proofing. This is particularly relevant from an ROI perspective, since next generation services must be deployed on today's networks for the next 12 to 18 months as new networks (such as IMS) come into being.
- **Enablers and Enabler framework** that provide development time and run-time encapsulation of key components and building blocks that are critical to service development.
- Components that bring **SOA** (Service Oriented Architecture) principles to a service delivery context by enabling orchestration, policy enforcement and composition.
- A **service gateway** that allows partners controlled access to critical network resources with SLA enforcements.
- A **unified user profile** that provides a unified view of user (subscriber) information to the service layer and execution environment in the presence of silos of information and multiple networks
- **Out of the box services** built on the standards-based platform that can be used to demonstrate immediate ROI.

The SDP opens up the network to a vast community of services and developers without sacrificing the management, security and control necessary to deliver the quality of service the telecom industry demands. Additional detail on each of the components can be found in the companion white paper *Carrier-grade J2EE as the foundation of the Oracle SDP*.

## ENABLERS AND AN ENABLER FRAMEWORK

The notion of having a layered architecture where the complexity of underlying systems can be abstracted from application developers who build on those underlying systems is well known in the IT industry. Such an approach brings many benefits to an application developer, including application portability across various systems that expose similar functionality, future-proofing and ease of service migration. The Oracle Service Delivery Platform brings this IT mindset into the

service provider domain with its concepts of out of the enablers as well as a common enabler framework.

## Enablers

Enablers are building blocks that expose a simple northbound API (with a Java or a web service binding for example), and isolate the user from the complexities of the underlying network capabilities. Enablers are a key pillar to the overall strategy of future-proofing service development and expanding the reach of developers that are able and willing to develop network-centric applications. In the Oracle SDP, enablers will typically provide the following features and benefits:

- **Abstraction** of underlying network technology choices and settings. For example, regardless of the vendor, the specific protocol extension used by a specific vendor, an enabler can expose the same capability “northbound” to an application developer. This can guarantee the stabilization of application development APIs with respect to underlying network technology choices. As a result, migration at the network level and integration with existing and future network elements can be carried out incrementally without wholesale replacement of silos as is the case today.
- **API-based** support for service integration. By supporting a broad range of APIs (Java or C-based vs Web services based), enablers can permit a number of integration scenarios including in-house development as well as integration with 3rd parties. This improves operator and service provider flexibility when they make a decision on whether the implementation of a particular feature should be done by their own developers or whether such an effort can be outsourced to 3rd parties. Also, API-based application integration offers significant benefits when compared to the traditional Telecom approach of protocol-based component integration.
- Simpler **OSS/BSS integration**. By providing enablers (such as charging) that encapsulate common OSS/BSS integration requirements, operators can consolidate and share OSS/BSS systems across multiple services. Consolidation across multiple networks and network technologies is critical to support convergence between wireless, wired and broadband networks, for example. Having a unified view of charging, identity management, subscriber profile, CRM and PRM systems makes service migration and network technologies and vendors significantly more cost effective when compared to a silo-based approach.

The Oracle Service Delivery Platform vision includes several out of the box enablers that have the characteristics defined above and which expose common, reusable functionality. Later on in this document, we discuss the following out of the box enablers in more detail:

- SIP Servlet container
- Call control

- Presence
- Media server control
- Messaging

### **Enabler Framework**

In addition to providing the out of the box enablers listed above, Oracle believes that it should be possible to build a thriving ecosystem of partners who can build additional enablers of their own to solve specific challenges. For example, a system integrator could develop a library of enablers to speed up the effort of developing custom applications. An ISV may wish to build enablers as a mechanism to componentize their application as well as expose reusable application components to 3<sup>rd</sup> party developers and system integrators. A general purpose enabler framework can accomplish this.

Given the basis of carrier-grade J2EE as the basis of the Oracle SDP, Oracle's approach to an enabler framework is perhaps predictable. In the Oracle Enabler framework, custom enablers expose north bound APIs through Java or Web Service bindings and integrate southbound to network elements or west bound to OSS/BSS systems through Java Connector Architecture (JCA) v1.5. In addition, an enabler framework provides full life cycle support for enablers, including deployment, activation, upgrade, management, deactivation and un-deployment. As a result, 3<sup>rd</sup> party enabler developers can rest assured that the enablers they build on the Oracle Service Delivery Platform have the same range of benefits that accrue to standard, out of the box enablers such as presence or call control that Oracle provides. The specifics of the enabler framework functionality summarized above will be detailed elsewhere.

### **ORACLE SDP AND ENABLERS**

The Oracle Service Delivery Platform will include a number of out of the box enablers, built on the enabler framework discussed earlier. In this section, we provide an overview of some of these. It's worth noting that given the open standards-based nature of the Oracle Service Delivery Platform, Oracle fully expects ISV and SI partners to develop additional enablers that will expose common, reusable functionality.

#### **SIP Servlet container**

The SIP Servlet container (along with out-of-the-box functionality that includes a proxy, a registrar and a SIP location server) implements the SIP Servlet programming model defined by the Java community process (JSRs 116 and 289). SIP Servlet is a programming model that is comparable to the "Request/Response" model commonly known as HTTP Servlets. Indeed, HTTP Servlets were an inspiration for the SIP Servlet programming model, given their widespread usage and their simplicity. Accordingly, the SIP Servlet programming model provides a "container" that can be used to build SIP-specific services whereby the

application developer does not need to understand the complete signaling details of the SIP protocol and can instead program at a higher level of abstraction. SIP Servlets is also a widely accepted programming model in the industry, with commercial implementations from a number of leading IT vendors, including Oracle.

In addition to providing an implementation of the SIP Servlet container programming model, the Oracle SIP Servlet Container enabler provides a number of out-of-the-box commonly used SIP components. For example, the stateless Proxy server provided out of the box with the SIP Servlet container implements a high-performance stateless proxy that can be immediately deployed in an operator network as is. However, should someone wish to customize the behavior of the stateless proxy or build an intelligent stateful proxy, they can develop a custom servlet on top of the SIP Servlet Container that implements the stateful proxy functionality and be assured that their implementation of a stateful proxy is portable across multiple SIP Servlet containers. This is one of the key benefits of providing out-of-the-box enablers on a standards-based platform – in the common case, the enabler provides enough functionality for faster deployment and time to market, and when additional functionality is desired, that can be developed on the standards based platform in a portable manner.

In addition to the stateless proxy, the SIP Servlet Container provides a number of other out-of-the-box commonly used components, including a SIP registrar, a SIP location server, a STUN server and an ENUM server. The availability of these out-of-the-box components significantly reduces the end-to-end development time for SIP-enabled media-rich applications such as VoIP without diminishing the freedom that sophisticated developers have to build richer, more complex applications and services.

A final unique feature of the Oracle implementation of the SIP Servlet container is that it integrates with a variety of J2EE application servers, including JBoss and the Oracle Application Server. This multi-J2EE application server approach allows the SIP Servlet container enabler to fit into a number of different operator deployment scenarios, and maximizes the choices available to an operator or a service provider.

### **Call control Enabler**

The call control enabler provides north-bound Java and web services APIs for a number of different call control scenarios across multiple networks. The simplest usage scenario for a call control enabler is as a 3<sup>rd</sup> party controller. A simple example is the click-to-call (click-to-dial) scenario. For example, an E-commerce web site might have the click-to-dial icon on its web site to assist a user through the shopping process. For example, a user may have added an item to their shopping cart, but the pricing may not be displayed on the web site since the (r)etailer wishes to offer a discounted price only to the serious shopper. In this situation, the (r)etailer may simply place a click-to-call icon right next to the price indicator – when the user clicks on this icon, a VoIP session is initiated between a sales

representative at the (r)etailer and the prospective buyer. Since a shopper who takes the trouble to initiate a phone call to find out the actual price of the product is more likely to purchase the product, it is reasonable for the salesperson to provide this information to the shopper when they call using click-to-dial. The server instance that is running the click-to-dial functionality in this instance is acting as a 3<sup>rd</sup> party call controller (since the call involves the prospective buyer and the sales person), and is responsible for terminating the call when appropriate.

It is trivial to implement a click-to-dial application by using the call control enabler. The call control enabler supports first party and 3<sup>rd</sup> party call control modes, and in each mode, the ability to set up calls, tear them down or to modify the call state (e.g., add another party to a call). Since the call control enabler exposes both Java and web services based APIs, integration between an ecommerce portal and the call control enabler is trivial and can be accomplished using a number of industry standards such as WSRP (Web Services for Remote Portlets – JSR 168). This makes it particularly attractive to an enterprise/IT developer who simply wishes to integrate the click-to-dial functionality to an ecommerce site without wanting to understand the details of the signaling that goes on when a call actually is set up (either in a SIP-only environment or in an environment that involves multiple network technologies).

### **Presence Enabler**

Presence information is central to a whole generation of next generation applications that sit at the heart of network and IT convergence. For example, a universal messaging application may use presence information to deliver a message either through IM (instant messaging) if the user is logged into an IM client or through SMS if the user is available on a mobile device, but is not logged into any IM client. Indeed, the characterization of presence as the next generation dial-tone where every application can depend on the availability of presence information to perform intelligent routing is widely accepted today.

The presence enabler in the Oracle Service delivery platform performs two major functions. First of all, it is responsible for aggregating presence information from a number of different sources in a persistent, scalable manner. This is attractive for application developers who simply wish a summary view of the different ways a given user can be reached without wishing to interrogate the various sources of presence themselves. The presence enabler accomplishes this objective by providing Java and web service based APIs that an application developer can use to obtain a summary view of the presence information for a given user. As presence information is highly dynamic, the presence server is able to update the presence information on the fly (e.g., when a user logs out from an IM client and goes mobile) – this allows the application developer to depend on the presence enabler as the “source of truth” regarding dynamic presence information for a user.

A secondary function the presence enabler provides is the ability to publish the presence information to a number of different sources. In this scenario, the

presence server does not wait to be interrogated about the presence status of a user by an application – instead, it is able to asynchronously update the presence information of a user when that information is updated. As a result, the application developer can use standard J2EE and EDA constructs to make an application immediately more responsive to presence information. For example, an employee portal that immediately updates the presence status of one’s peers based on a change in status is quite helpful when someone is making a decision on how best to reach their colleague for an urgent issue. If the application is well designed, the employee in this case does not have to refresh their employee portal page that displays presence information for their peers – instead the page is automatically refreshed and updated with every change to presence status for the colleagues an employee cares about (similar to how buddy lists work in an IM client for example).

### **Media Server Control**

Media server capability is central to a number of telephony-related applications. For example, a conferencing application needs to integrate with a media server to implement floor control (e.g., the ability to mute or un-mute certain lines) or for regulatory reasons (e.g., to record a conference call). A media server is also required when integrating with voice capability – e.g., the ability to capture the key press events when a user is using DTMF.

Unfortunately, the media server industry has not yet converged on a set of industry standards on controlling media server functionality. There are two leading contenders for this role – MSML, pioneered by Convedia and supported by Convedia as well as Pactolus and MSCML, pioneered by Brooktrout/Snowshore (currently Cantata). The difference between MSML and MSCML is not just syntactic – the abstractions they provide for controlling a media server are quite different and developers have to master both. Since there is no clear leader in the media server marketplace today, it is difficult for a developer to build a media-centric telephony application without supporting both these standards. There is also a third standard called MOML that is used by some media servers.

The media server control enabler solves this problem by providing a set of standard north-bound APIs for controlling media server functionality across all these different control technologies. The media server control implements drivers that target MSML, MSCML and MOML and hides the differences in semantics between these control languages from the application developers. As a result, the application developer has a reasonable expectation of application portability across media servers. Oracle is also putting in place an interoperability testing program that would allow arbitrary media server vendors to certify with the media server control enabler.

### **Messaging Enabler**

The messaging enabler provides out of the box support for sending and receiving messages in A2P (application to person), P2A (person to application) and A2A

(application to application) scenarios using SMS and MMS. SMS (short message service) and MMS (multimedia messaging service) along with its variant EMS (Enhanced Messaging Service) are universally deployed on mobile networks. However, connectivity to the specific messaging gateways (an SMSC or MMSC) is accomplished through a protocol interface that is quite vendor dependent. A number of different standard protocols and their variations are in use today, including UCP, CIMD (v1 and v2) and SMPP. This makes it challenging for an application developer to integrate SMS and MMS messaging into their application since they have to implement multiple protocol drivers to connect to the various types of messaging gateways for achieving sufficient coverage. Hosted messaging aggregators address this problem somewhat, but don't address the business model issue when an enterprise may wish to contract directly with an operator for better rates and higher quality of service.

The messaging enabler provides an elegant solution to this problem by providing a set of northbound Java and web service APIs that an application can use to send and receive messages. An application can register to receive and process incoming messages using standard Java/J2EE constructs and can invoke the outbound APIs for sending messages. As a result, the application developer can view messaging as an abstract service that can be implemented to connect to an arbitrary provider without having to understand or depend on what specific protocol is used by that provider's messaging gateway. As a result, the developer is able to integrate messaging into their applications in a more portable manner.

Messaging (particularly combining P2A and A2P scenarios to implement P2P messaging) is an area of active interest in the OMA. A number of operators and service providers have expressed interest in standardizing the P2P messaging scenarios to enable a number of different services, including content screening, content categorization, privacy and spam protection. The combination of P2A and A2P messaging implemented on a standards-based platform coupled with a flexible policy enforcement layer has the potential to revolutionize P2P messaging as we know it.

## **ENABLERS AND APPLICATION DEVELOPMENT**

The concept of enablers as reusable components that speed up application development is well understood in the IT domain and is starting to be well understood in the service provider and telecom domains. By exposing network-specific functionality using standard IT technology (Java and Web service APIs for example), the enabler framework brings SOA to the telecom world – it is possible to now treat common network or OSS/BSS functionality as a building block in an overall application or service and apply SOA principles to the overall application, including the enabler. All the principles and tools of SOA – reuse, delegation, composition and service orchestration – are available to an application or service that uses enablers as building blocks. In contrast, an application that works directly with the underlying network elements (e.g., using SMPP directly to integrate

messaging instead of using the messaging enabler) is less able to seamlessly integrate SOA capabilities such as delegated authentication or service orchestration.

## **CONCLUSION**

The Oracle Service Delivery Platform is a J2EE-based realization of the OMA Service environment (OSE). Enablers, which are key building blocks of reusable functionality that expose standard interfaces, are a central architectural principle underlying the implementation of the Oracle SDP.

Enablers bring to the Telecom and service provider domain a number of well understood lessons in the IT domain, such as encapsulation, abstraction, standard APIs and reuse. As such, relevant out of the box enablers along with a general purpose enabler framework are absolutely central to getting traditional IT and enterprise developers excited about network-centric applications. The Oracle SDP provides both such enablers as well as an enabler framework that allows 3<sup>rd</sup> parties to build their own enablers.

Figure 1

## Oracle Service Delivery Platform

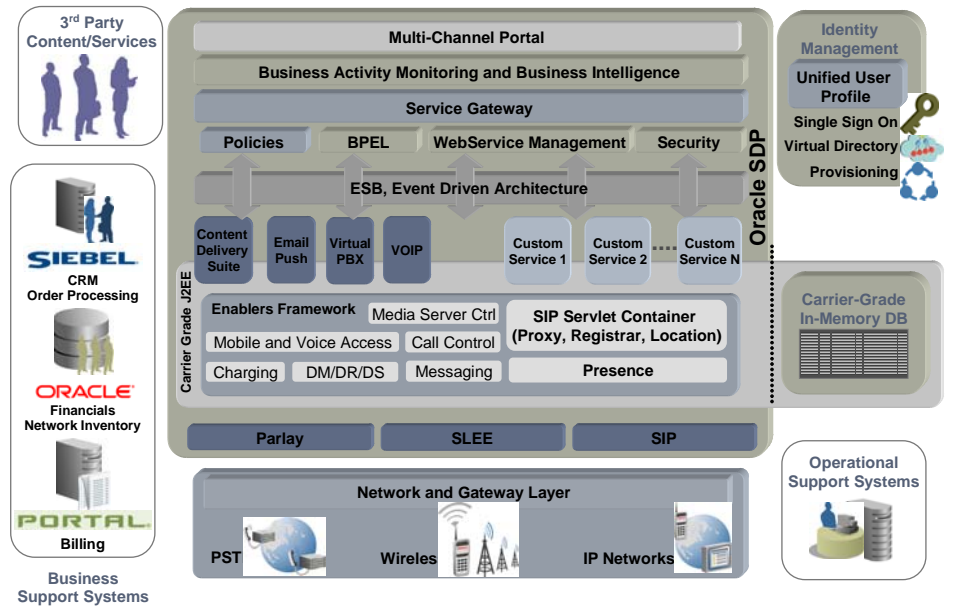
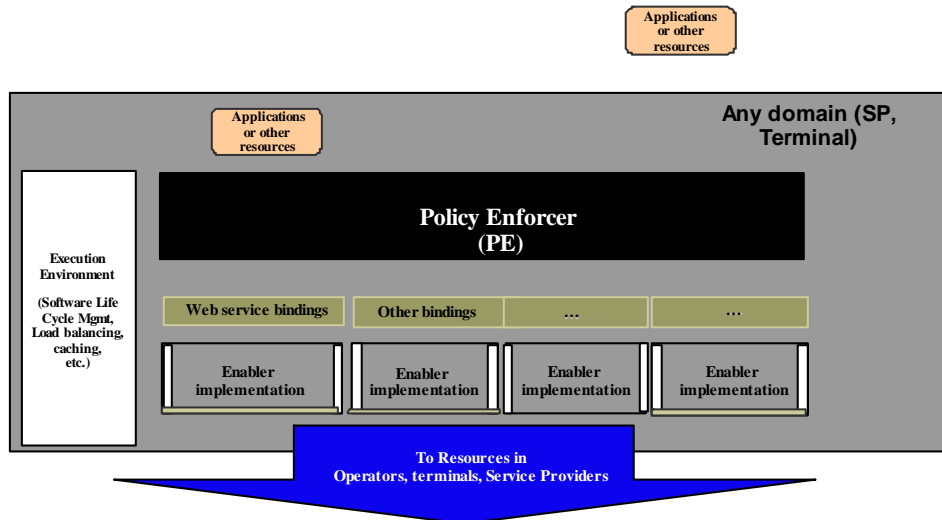


Figure 2

## OMA Service Environment (OSE)



ORACLE

Approved - [http://www.openmobilealliance.org/release\\_program/ad.html](http://www.openmobilealliance.org/release_program/ad.html)

## **ORACLE FUSION MIDDLEWARE**

January 2006

Author: Stéphane H. Maes

Contributing Authors: Indu Kodukula

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 © 2006, Oracle. 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, JD Edwards, PeopleSoft, and Retek are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners