Oracle® Reference Architecture

User Interaction

Release 3.0

E16349-03

April 2011

ORACLE®

ORA User Interaction, Release 3.0

E16349-03

Primary Author: Bob Hensle

Contributing Author: Cliff Booth, Dave Chappelle, Jeff McDaniels, Mark Wilkins, Steve Bennett

Contributor: Tim Breeden, Ron Schweikert

Contents

List of Figures

Send Us Your Comments

ORA User Interaction, Release 3.0

E16349-03

Oracle welcomes your comments and suggestions on the quality and usefulness of this publication. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this document?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the documentation and the chapter, section, and page number (if available). You can send comments to us at its_feedback_ww@oracle.com.

Preface

This document describes concepts, standards, principles, and a reference architecture for a modern user interface in an IT environment. The reference architecture is described in a product agnostic way since the architecture does not require any specific products; rather, it is based on industry and Oracle best practices for user interaction. The document describes the key capabilities of a modern user interface and how this differs from more traditional user interface approaches. The principles that are essential to a modern user interface are listed and described. These principles provide the foundation and requirements for the reference architecture which is then illustrated and described. Finally, Oracle products are mapped onto the architecture to illustrate how to realize the architecture using Oracle products.

Audience

This document is intended for Enterprise Architects and Solution Architects who want to understand modern user interaction and how to create an architecture that supports a rich, user-centric system interaction. Some level of understanding of user interface concepts and design is required since this document is not intended to provide a primer for user interface design.

How to Use This Document

This document is intended to be read from start to finish. However, each section is relatively self contained and could be read independently from the other sections. This document can be used to understand modern user interfaces or as a blueprint for creating an architecture to support modern user interactions.

Document Structure

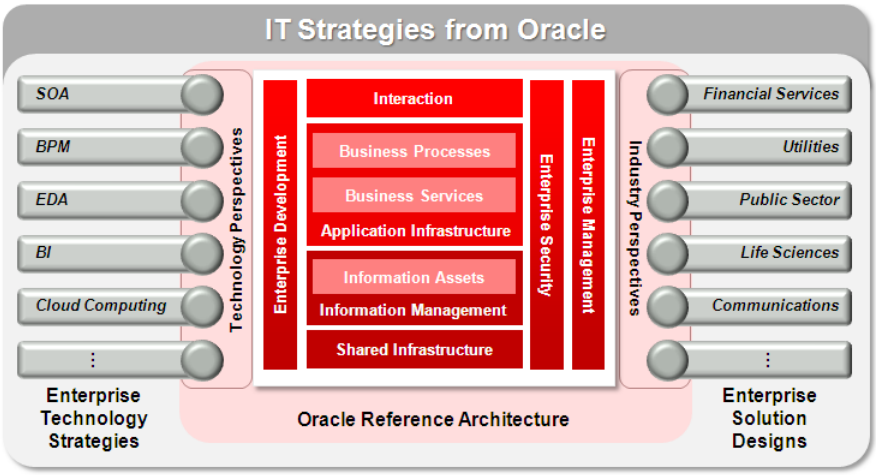
This document is organized in sections that first introduce and describe user interaction and then describes an architecture that is based on and supports a modern user interface. Specifically,

- Chapter 1 provides a description of user interaction and how this differs from more traditional user interface approaches.
- Chapter 2 identifies the principles that should be met by any architecture that purports to support modern user interfaces.
- Chapter 3 identifies the standards that are of particular relevance to a user interface architecture.

- Chapter 4 describes a reference architecture for modern user interfaces from various viewpoints.
- Chapter 5 maps Oracle products onto the architecture to illustrate how Oracle products can be used to realize the architecture.
- Chapter 6 is a brief summary of the document contents.

Related Documents

IT Strategies from Oracle (ITSO) is a series of documentation and supporting material designed to enable organizations to develop an architecture-centric approach to enterprise-class IT initiatives. ITSO presents successful technology strategies and solution designs by defining universally adopted architecture concepts, principles, guidelines, standards, and patterns.



ITSO is made up of three primary elements:

- **Oracle Reference Architecture (ORA)** defines a detailed and consistent architecture for developing and integrating solutions based on Oracle technologies. The reference architecture offers architecture principles and guidance based on recommendations from technical experts across Oracle. It covers a broad spectrum of concerns pertaining to technology architecture, including middleware, database, hardware, processes, and services.
- **Enterprise Technology Strategies (ETS)** offer valuable guidance on the adoption of horizontal technologies for the enterprise. They explain how to successfully execute on a strategy by addressing concerns pertaining to architecture, technology, engineering, strategy, and governance. An organization can use this material to measure their maturity, develop their strategy, and achieve greater levels of adoption and success. In addition, each ETS extends the Oracle Reference Architecture by adding the unique capabilities and components provided by that particular technology. It offers a horizontal technology-based perspective of ORA.
- **Enterprise Solution Designs (ESD)** are industry specific solution perspectives based on ORA. They define the high level business processes and functions, and the software capabilities in an underlying technology infrastructure that are required to build enterprise-wide industry solutions. ESDs also map the relevant application and technology products against solutions to illustrate how capabilities in Oracle's complete integrated stack can best meet the business, technical, and quality of service requirements within a particular industry.

This document is one of the series of documents that comprise Oracle Reference Architecture. ***ORA User Interaction*** describes the important aspects of a modern user interface and provides a reference architecture for delivering a robust, feature-rich user interface .

Please consult the ITSO web site for a complete listing of ORA documents as well as other materials in the ITSO series.

Conventions

The following typeface conventions are used in this document:

Convention	Meaning
boldface text	Boldface type in text indicates a term defined in the text, the ***OFRA Master Glossary***, or in both locations.
italic text	Italics type in text indicates the name of a document or external reference.
underline text	Underline text indicates a hypertext link.

"SOA Service" - In order to distinguish the "service" of Service-Oriented Architecture from the wide variety of "services" within the industry, the term "SOA Service" (although somewhat redundant) will be used throughout this document to make an explicit distinction for services that were created as part of an SOA initiative; thus distinguishing SOA Services from other types of services such as Web Services, Java Messaging Service, telephone service, etc.

1

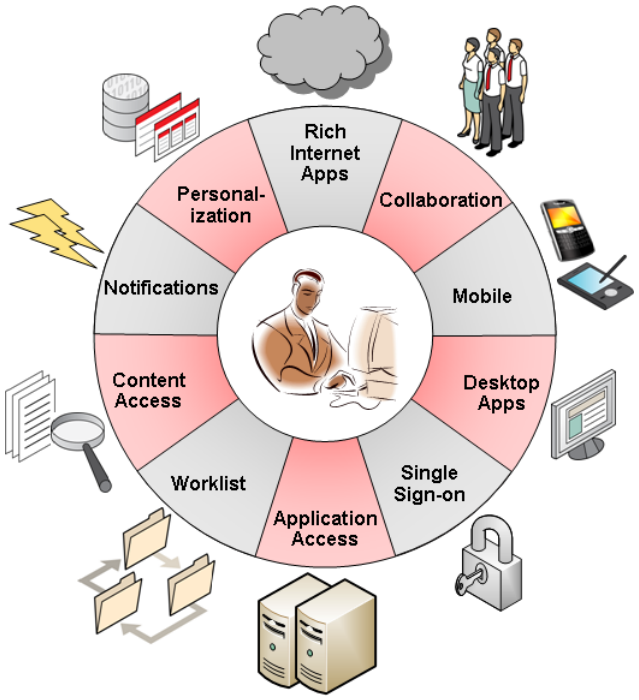
User Interaction Overview

This section describes the core concepts of user interaction and explains why modern user interaction brings new and expanded capabilities to the user than traditional user interfaces provided.

1.1 User Interaction

The primary goal of user interaction is to provide a user-centric, feature-rich, intuitive interface to end users tailored to the specific needs of each end user; thereby bringing maximum productivity to the work environment. Realizing this goal requires a flexible, extensible architecture since different types of end users will have very different needs. The user-centricity and breadth of capabilities that a modern user interaction requires are illustrated in Figure 1–1.

Figure 1–1 User Interaction Capabilities



The capabilities described here are not intended as the definitive list of all possible capabilities, nor must all of these capabilities be provided by a user interface to be considered "modern." Nonetheless, the capabilities described here are the capabilities that have become generally associated with state-of-the-art user interaction, and

therefore should be considered when defining an architecture to support modern user interaction.

As the various capabilities are described, it should become clear that the user interface surfaces the capabilities, but the underlying functionality may be provided by one or more separate and distinct systems. This is a fundamental concept of the user-centric nature of modern user interaction. The end user should be insulated from this complexity. As far as the end user can tell, it is one consistent, integrated interface that provides access to all the functionality required to complete their daily tasks.

1.1.1 Rich Internet Apps

A Rich Internet Application (RIA) is a Web application that provides the type of sophisticated user interface that has traditionally required a dedicated desktop client. There are a variety of technologies being applied to create RIAs, but all share the commonality of providing a sophisticated user interface delivered through a browser. Using a browser as the delivery platform leverages the infrastructure of the internet to reduce client deployment and maintenance costs that became the achilles heel of the dedicated desktop client. Thus, RIAs provide the richness of a dedicated desktop client with the ease of delivery provided by the Web.

Over the last few years Web applications have undergone dramatic improvements in user interaction capabilities. Initially Web applications were relatively simple static HTML pages. Dynamic HTML allowed Web applications to better respond to user requests but did little to improve the richness of the user interface. Today, Web applications frequently incorporate JavaScript or browser plug-ins to deliver sophisticated user interaction capabilities.

Here the term RIA is used to describe user interaction capabilities that include the latest technologies as well as the older, less sophisticated Web capabilities such as static and dynamic HTML. All of these technologies and techniques should be viewed as tools in the tool box and the tools should be chosen to best fits the needs of the end user.

1.1.2 Collaboration

The fundamental driver for the creation of the internet was to facilitate the collaboration between people. For example, HTML was created to allow researchers at CERN to share documents and the World Wide Web infrastructure was created to allow sharing across the entire globe. Thus, it is no surprise that a key capability of user interaction is collaboration.

Initially the Web was used to publish information i.e. the owner of the Web site provided the content for the site. This is no longer the case. Many Web sites today are designed to host content provided by other Web citizens. Examples abound and include Wikipedia (reference), Facebook (social networking), eBay (commerce), Blogger (personal commentary), GMail (email), etc. Even Web sites that are still publication focused (e.g. newspaper sites) usually include the ability for visitors to post comments. Thus, the Web has dramatically expanded the number of contributors or participants. This new “participatory Web” is commonly referred to as Web 2.0.

Collaboration as used here encompasses all of the current collaboration focused capabilities including email, calendar, contacts, blogs, wikis, etc. This requires that the user interface provides access to all of these types of collaboration capabilities. As stated above, the underlying functionality may be provided by another system, but the user interface must still incorporate these capabilities into a unified user experience.

1.1.3 Mobile

The rollout of third and fourth generation (3G and 4G) mobile networks coupled with the increasing capabilities of mobile devices has enabled the use of mobile devices for tasks that traditionally required a desktop or laptop computer.

The display capabilities (screen size, resolution, etc.) for mobile devices is markedly different than for desktop or laptop computers. Therefore, providing a user friendly interface for a mobile device usually entails some development to create a mobile device appropriate interface. Since the differences are focused on usability not functionality, the development effort should be isolated to the display aspects of the solution.

Transcoding is a technology that may be part of the mobile support for a solution. Transcoding automatically makes the necessary adjustments to the user interface to support the unique properties and idiosyncrasies of different mobile devices. The mobile market appears to be coalecing around a small number of mobile platforms (e.g. iPhone, Android) so trancoding has become less important than it was just a few years ago when there was a plethora of mobile device platforms.

1.1.4 Desktop Apps

The widespread adoption of desktop computers (and later laptop computers) was primarily driven by the productivity gains resulting from the use of desktop applications. Desktop applications can be widely applicable personal productivity tools (e.g. speadsheets, word processors) or may be custom applications targeted to specific employee roles and tasks.

With the advent of Web applications and more recently with the growing deployment of RIAs, the preeminence of desktop applications has faded, especially for custom applications. Nonetheless, there is little risk that desktop applications will vanish any time soon if at all, especially the extremely widespread applications such as spreadsheets, word processors, etc. Therefore, user interaction must interact with these desktop applications in a user friendly manner.

For example, it should be easy for the user to pull data that is stored remotely into a spreadsheet so that the data can be analyzed locally on the user's desktop. This removes the need to copy data (with the attendant issues of staleness, consistency, security, etc.) by allowing a single source of the data to be analyzed locally.

Obviously there are many other types of interactions with desktop applications that are possible. Fundamentally, modern user interaction must allow data and functionality to be integrated with desktop applications to allow end users to work more efficiently.

1.1.5 Single Sign-on

A user-centric interaction provides access to data and functionality from a variety of sources to simplify and improve the user experience; thereby increasing end user productivity. The sources of data and functionality are usually protected by some type of security to ensure that only legitimate users get access. Requiring that the end user authenticate individually to each source system not only burdens the user with repetitive logon procedures, but actually annoys the end user.

Single sign-on is the solution to the problem of multiple source systems each protected by security. Using single sign-on the end user provides their authentication credentials (e.g. user name and password) once and the user interaction system propagates the security credential as necessary so the end user can seemlessly access data and functionality from a variety of source systems.

1.1.6 Application Access

In order to perform their assigned tasks, frequently employees must access data and functionality provided by multiple different applications. Traditional application-centric user interfaces require the end user to learn multiple different application interfaces to perform their job function. This can be especially onerous when the user tasks only require a small subset of the functionality provided by the application. Adding to the user frustration, common data requirements may exist between the various applications that require the end user to perform a similar action in multiple applications e.g. entering a customer address several times into different applications.

A user-centric approach (rather than an application-centric approach) to user interaction provides the user with a single interface that pulls together the data and functionality from the source applications into an interface that is tailored to the specific needs of the end user. This user-centric interface can surface multiple different source applications, eliminate repetitive user input, and give the user a unified experience that hides the complexity of the underlying applications. In addition to increasing user productivity, this approach also reduces training costs and data entry errors.

1.1.7 Worklist

A common technique people use to stay organized and productive is to create a to-do list that identifies the tasks the person needs to complete. The worklist is essentially an automated to-do list. An item is added to the worklist when the end user needs to complete a task, and an item is removed from the worklist when the end user completes the associated task.

The worklist provides the organization with the productivity advantages of a personal to-do list, but also, and more importantly, organizes the collaborative efforts of multiple people into a measurable and manageable process. Items that stay in the worklist too long may get automatically escalated. Metrics (e.g. task duration, worklist length, escalations) can be captured that can be used to improve the overall process.

Integrating the worklist into the user-centric interaction can be much more than a list of tasks to complete. The items in the list may include links that take the end user to the exact screen that the user must use to complete the task. The screen may surface data from multiple source applications to provide the information the end user needs to complete the task. And, once the task has been completed, the worklist can be automatically updated to reflect that completion.

1.1.8 Content Access

In the current Information Age, it is essential that workers have access to the information they need. Businesses achieve competitive advantage by making relevant information readily available to their employees so that the employees can make timely, accurate decisions. Thus, a user-centric interaction must provide ready access to any type of content that the end user might need. The types of content that an end user might need are varied and include documents, presentations, graphics, images, recordings, etc.

Not only must the interface provide access to a wide variety of content types, it must also provide the means for the user to find the desired content. Therefore, the interface should provide search capability across all content sources based on user defined search criteria such as content type, topic, key words, date, etc. Requiring a user to constantly search for content can quickly become onerous; therefore it must also be

possible for the user to organize the content using categories, and to “tag” content once it has been found so that it can be easily retrieved.

Providing comprehensive content access:

- increases employee productivity since less time is spent looking for the needed content,
- improves the quality of work since the work product is based on complete information,
- reduces re-work since previously completed efforts are readily available.

Comprehensive content access is a foundational element for building organizational competency.

1.1.9 Notifications

Instead of a user interaction that is totally passive that simply waits until the user takes action, notifications provide the means to push information to the end user. Notifications come in a variety of types and for a wide variety of reasons. Some notifications are based on a subscription approach where the end user subscribes to receive notifications. Web feeds (e.g. RSS, Atom) are an example of this type of notification. Subscription notifications allows the end user to remain up to date on topics of interest without having to regularly visit the topic site(s) to look for updates.

Unsolicited notifications are also important to providing a user-centric interaction. Unsolicited notifications are notifications that may be of interest to the end user based on information known about the end user (i.e. their profile). Many e-commerce sites are experts at providing these unsolicited notifications. For example, Amazon provides unsolicited notifications (via email) of new books that may be of interest to the user based on similarities to other users or past book purchases. End users should be able to "opt out" of these unsolicited notifications.

Organizations can also leverage unsolicited notifications to improve corporate communications. Examples of using unsolicited notifications for corporate communication include notification of benefits open enrollment periods, corporate news, relevant industry news, employee review periods, etc.

Notifications remove the burden from users of having to regularly check for new information or updates. Notifications improve the distribution of information and opportunities by proactively informing the end user.

1.1.10 Personalization and Customization

No two people work in exactly the same way, even people in the same role within an organization. Thus, to be truly user-centric, the user interaction must be tailorable to desires of the individual end user.

Customization allows the end user to change the interface (within allowed parameters) to better suit their needs or individual preferences. In addition to customization that is done by the end user, a modern interface should be readily customizable by administrators. This allows the administrators to change the interface based on organization, department, role, etc. needs and desires. Customization is usually a static change i.e. once configured, the interface stays that way.

Personalization tailors the interface to the end user based on user profile information (e.g. location, role, demographic), past behavior (e.g. articles accessed, previous purchases), and current context (e.g. time of day, open enrollment). Personalization differs from customization in that the interface is tailored to the end user without the

end user taking any action. Personalization is generally dynamic i.e. the interface changes based on decisions made at runtime.

Interface personalization and customization increases user productivity by allowing administrators and end users the ability to create a user interaction that best meets the needs of the end user. The days of a one-size-fits-all user interface are gone.

1.2 Difference from Traditional User Interfaces

IT departments have been building user interfaces for decades. Many different technologies have been used over the years to create user interfaces. This section compares and contrasts traditional user interfaces with a modern user interface defined by this document.

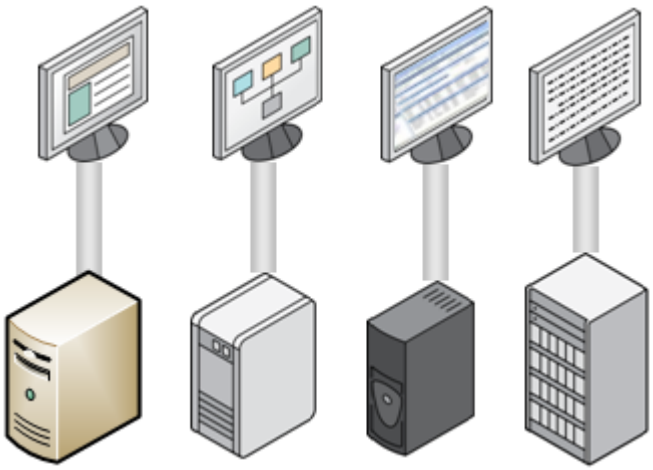
1.2.1 Traditional User Interfaces

Early user interfaces were text based with very limited display capabilities, usually driven by obtuse key stroke sequences, and each screen contained very limited information. Client server technologies introduced much more capable display devices and graphical user interfaces that significantly improved the user experience. While these dedicated, client-server, desktop user interfaces are the gold standard for rich user interfaces, they have three major disadvantages:

1. The client application is specific to the desktop device. This requires the development of a different client application for each type of desktop that will be supported; thus increasing development and support costs.
2. The dedicated desktop client must be installed on each client device. This increases initial deployment effort and costs. Additionally, whenever a new version of the client is released each desktop device must be updated to the new version; thereby increasing ongoing maintenance costs.
3. The dedicated desktop client has a relatively narrow focus for the capabilities provided i.e. the dedicated desktop client is tied directly to a single application. While this limitation is not inherent in dedicated desktop clients, it is how dedicated desktop clients have been architected, designed, and built. The dedicated desktop client has not been developed into a general purpose user interaction delivery platform.

Figure 1–2 illustrates the traditional dedicated client where each client is connected to a single backend system.

Figure 1–2 Traditional User Interface



Many of the maintenance issues associated with the client-server desktop were overcome with the advent of web technologies and moving the user interface into a standard web browser. The standardization and reduced cost of maintenance brought by the web browser interface was paid for by a much less rich user interface: HTML did not provide the richness of the dedicated client.

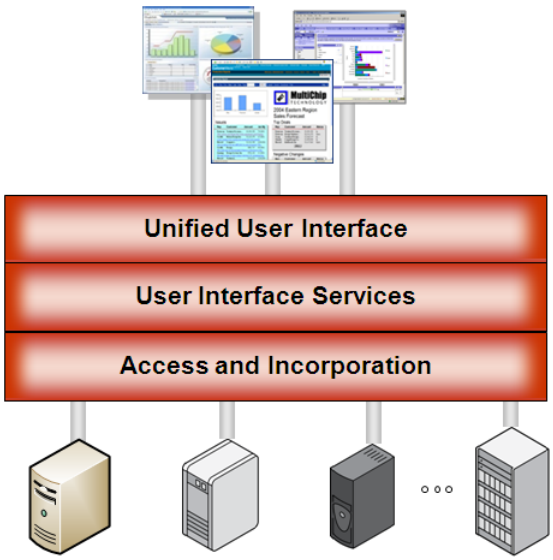
Over time the capabilities of the browser interface have significantly improved due to browser improvements and from various technologies such as JavaScript, browser plug-ins (e.g. Java applets, Adobe Flash), and expanding standards (e.g. new versions of HTML). Browser based interfaces are nearing parity with dedicated clients.

However, switching to a browser-based client did nothing to change the topology illustrated in Figure 1–2. Each backend system still had its own unique browser-based interface. Nonetheless, the standardization brought about by web technologies removed the barrier to providing a single, unified user interface that is core to the modern user interface.

1.2.2 Modern User Interface

The Conceptual View of the modern user interface is shown in Figure 1–3.

Figure 1–3 Conceptual View



While Figure 1–3 is a relatively simple diagram, there are a variety of concepts that are illustrated, as described below.

1.2.2.1 Display Devices

The figure shows multiple screens but no specific display devices; thus illustrating that the type of display devices is indeterminate, i.e. the architecture supports any type of display device. Additionally, whereas the traditional user interface illustrated in Figure 1–2 shows one screen per backend system, this figure shows that the number of screens is independent from the number of backend systems. The screen contents are defined by the needs of the user, not by the number or types of backend systems.

1.2.2.2 Unified User Interface

The Unified User Interface layer provides the control and visual elements that define the interaction the user has with the system. This layer separates the way the user interacts with the system from the underlying functionality provided by the system. This has many advantages including allowing different display devices to be supported via control and visual elements specialized for the device since, for example, mobile devices do not have nearly the screen real estate of a desktop computer.

1.2.2.3 User Interface Services

The User Interface Services layer provides a set of functionality that can be used and reused in a variety of ways to deliver various user interfaces specialized to the needs of the end user. This illustrates that the underlying functionality is separated from the visual and control elements built into the user interface. The services provided by this layer may come from a variety of sources located anywhere that is network accessible.

1.2.2.4 Access and Incorporation

The Access and Incorporation layer provides the capability to incorporate data and functionality from any number of backend systems into the user interface. Generally, there are two types of backend systems that need be incorporated into the user interface: systems that are designed for use with user interface (e.g. LDAP, dedicated database) and systems that are not (e.g. legacy applications). The former type of

systems can be access directly by the user interface architecture. Ideally the latter type should be accessed via a robust integration architecture rather than relying on point-to-point integrations.

This distinction is the reason that the term “incorporation” is used in this Conceptual View instead of the term “integration.” A suitable integration architecture is described in the ***ORA Service-Oriented Integration*** document.

1.2.2.5 Backend Systems

Four different graphics are used to represent backend systems. This is done to illustrate that many different types of systems may be incorporated into the user interface. Additionally, no attempt is made to define the types of backend systems that might be incorporated since the number and types of backend systems which could be incorporated is virtually limitless.

1.3 Complementary Technologies

This section discusses types of technologies that are associated with user interaction. Categories of technologies are discussed not specific products. The categories of technologies are not user interaction technologies *per se*, but rather they are technologies that impact or are impacted by a user interaction architecture.

1.3.1 Enterprise 2.0

Enterprise 2.0 is a nebulously defined term that refers to bringing Web 2.0 technology into the enterprise IT environment. A common theme for Enterprise 2.0 is collaboration: collaboration within the enterprise as well as collaboration between the enterprise and customers or partners.

As discussed above, the primary distinction between Web 1.0 and Web 2.0 is the change from a publishing paradigm to a participatory paradigm. There are a variety of technologies that have become associated with Web 2.0 in that these technologies have been used to create a rich user experience that facilitates participation. However, it is the capabilities provided to the end user that distinguishes Web 2.0 from Web 1.0, not the technologies themselves.

This document will discuss Web 2.0 technologies where they apply to the user interaction architecture. However, this document does not address Web 2.0 or Enterprise 2.0 directly or completely. Rather, Enterprise 2.0 is an Enterprise Technology Strategy (ETS) and will be covered by dedicated set of collateral (ORA perspective, foundation document, maturity model, etc.).

1.3.2 Content Management

Content Management technologies are used to manage content that is made available to an end user by the user interface. As such, Content Management is very complementary to user interaction. However, Content Management is far more that just making content available through a user interface. Content Management also includes capabilities such as content lifecycle management, archival, approval processing, document capture, etc.

This document will discuss the interaction between Content Management and the user interface. However, this document will not detail Content Management capabilities; rather Content Management will be treated as a source system that is integrated into the user interaction. A complete description of Content Management will be provided in a separate set of documents contained in the Content Management ETS.

1.3.3 Service Oriented Architecture

Service Oriented Architecture (SOA) is a strategy for constructing business-focused, software systems from loosely coupled, interoperable building blocks (called SOA Services) that can be combined and reused quickly. The user interaction architecture defined within this document follows the principles of SOA. Specifically, the user interface may surface data and functionality that is provided by external SOA Services, and conversely, the user interaction architecture provides access to internal capabilities via SOA Services.

Creating a SOA requires a holistic approach and encompasses far more than just the user interaction elements that are the topic for this document. Thus, this document will touch on some SOA topics but only where directly applicable to the user interaction. A comprehensive set of collateral for SOA is available from the SOA Eterprise Technology Strategy.

1.3.4 Business Process Management

Business Process Management (BPM) technologies are used to automate and manage business processes. Frequently business processes include task that are completed by a human participant. This human interaction is a type of user interaction that the user interaction architecture must support. However, BPM includes many capabilities that are not directly visible to the business process participant. These capabilities include process modeling, state management, process monitoring, integration to source systems, etc.

This document will discuss the interaction between BPM and the user interface, specifically how the end user participates in a business process that is being managed by BPM technologies. However, this document will not detail BPM capabilities; rather BPM will be treated as a source system that is integrated into the user interaction. A complete description of BPM is provided by a separate set of material collectively called the BPM Enterprise Technology Strategy..

1.3.5 Interactive Voice Response

Interactive Voice Response (IVR) is technology that allows a computer to detect and respond to voice (voice recognition) and touch-tone (telephone keypad) inputs. IVR can be used to control almost any function where the interface can be broken down into a series of simple menu choices. Thus, IVR is a type of user interaction. However, due to the limitation of simple menu choices, IVR addresses a narrow segment of the desired user interaction spectrum.

For this document, IVR is viewed as a specialized case of user interaction that requires specific technologies which are not covered by this document. However, IVR does address specific needs and should not be ignored totally when discussing user interaction. Thus, this document does discuss IVR integration to the user interaction architecture.

1.3.6 Identity Management

Identity Management (IdM) can be viewed as the integration of multiple identities and identity stores within an enterprise. The user interaction architecture can leverage IdM solutions where they exist. IdM provides a single source for authentication and authorization that can significantly simplify security concerns, especially single sign-on. In the context of this document, IdM solutions are considered source systems for security capabilities and user profile information.

2

User Interaction Principles

This section defines the principles of modern user interaction. These principles provide the guidance for creating a sound architecture that will fully support a modern user interface.

The principles defined below provide sound guidance for creating an architecture for user interaction. Each organization should evaluate the principles listed and derive their own set of principles that match the specific environment and goals of their organization.

The principles are:

2.1 Display Device Support

Principle	**Display Device Support**
Statement	The architecture must support multiple different display devices.
Rationale	An architecture that supports only a single display device is prohibitively limiting in today's connected world. Even if only a single display device (e.g. personal computer) is the initial focus for a business solution, the architecture must be designed to readily support additional display devices; otherwise the cost of supporting another display device constrains the future flexibility of the solution.
Implications	■ The display device specific aspects of a business solution should be minimized. ■ Applicable standards should be followed to maximize interoperability. ■ The architecture must support multi-channel delivery of content and functionality.

2.2 User Interaction Adaptability

Principle	**User Interaction Adaptability**
Statement	The architecture must support the ability to adapt to the end user needs and desires.

Rationale	The user interface should adapt to the needs of the end user rather than the user having to adapt to the interface. This adaptability has many different aspects including language support (internationalization and localization), the ability of end users to modify the interface (customization), and the interface changing implicitly to provide a better user experience (personalization).
Implications	■ The architecture must provide multi-lingual support. ■ The architecture must include both customization and personalization.

2.3 Unified User Experience

Principle	**Unified User Experience**
Statement	The architecture must support pulling together information and functionality from multiple sources into a single unified user experience.
Rationale	The days of silo'ed, single purpose user interfaces are past. Ready access to information and functionality is a competitive advantage in today's information driven economy.
Implications	■ The architecture must provide a means to incorporate multiple sources of information and functionality. ■ The user interface must support modular display to allow multiple components to display simultaneously.

2.4 Physical Separation

Principle	**Physical Separation**
Statement	The architecture must support physical separation of the user interface from the sources of information and functionality.
Rationale	The architecture must make no implicit assumptions about the relative locations of the front-end user interface and the back-end resources. The front end may be a mobile device or a computer on the public internet. The network connection between the front end and back end may be slow, unreliable, and unsecure.
Implications	■ The user interface should minimize the frequency of communications between the front end and the back end. ■ The front-end technology platform may be different from the back-end technology platform. ■ Other front-end components (e.g. WSRP portlets, mashup widgets) may also by physically separated from the user interface.

2.5 Secure Interactions

Principle	**Secure Interactions**
Statement	The architecture must provide secure interaction between the end user and the server-side resources.

Rationale	Security breaches can be costly to the business from a financial perspective, legal perspective, or brand perspective; therefore, the architecture must provide end-to-end security.
Implications	■ The back-end resources must be protected from a compromised front end. ■ The front end must be protected from a malicious back end. ■ The architecture must provide secure communications over unsecure networks.

2.6 Development for Disparate Display Devices

Principle	**Development for disparate display devices**
Statement	The architecture must provide for the development of user interfaces for a variety of display devices.
Rationale	Just as the architecture must be able to support multiple types of display devices, it also must facilitate development of user interfaces hosted on different types of display devices. This includes both browser-based (i.e. "thin client") interfaces as well as device-specific (i.e. "fat client") user interfaces.
Implications	■ The development environment should support different development frameworks for the different device types. ■ Device emulation may be required to support development for mobile devices.

2.7 Rapid Development

Principle	**Rapid Development**
Statement	The architecture must provide rapid, guided development of the user interface without needlessly exposing the developer to implementation details.
Rationale	Reducing the time and resources required to deliver business solutions is required for IT to better support and align with business needs. Low level development is both time consuming and error prone.
Implications	■ The architecture should provide an integrated development environment with integrated help and wizards to speed and simplify development. ■ The developer should have access to the lowest level code and configuration details when necessary, but the need to access this level of detail should be rare. ■ The development environment should facilitate code debugging to rapidly identify and fix defects.

2.8 Access to Functionality

Principle	**Access to Functionality**
Statement	The architecture must support end user access to a wide variety of server-side functionality.

Rationale	Ultimately the goal of the user interface to provide the end user access to all the necessary information and functionality regardless of the source of that information or functionality.
Implications	■ The architecture should support the hosting of functionality desired by the end users. ■ The architecture should provide mechanisms to integrate with existing sources of information and functionality. ■ The end user should be isolated from the particulars of the actual source of information and functionality.

2.9 Meta-data Support

Principle	**Meta-data support**
Statement	The architecture must support separating configuration and other types of meta-data from the source code.
Rationale	Source code modification is the least efficient mechanism to modify behavior of a system. The ability to change the behavior of the system by changing externalized meta-data provides much quicker response to business needs.
Implications	■ The meta-data should be accessible both within the development environment (to support developers) and from outside the development environment (to support others). ■ Externalizing meta-data should to be incorporated into the development process.

2.10 Performance

Principle	**Performance**
Statement	The architecture must provide performance acceptable to the end users.
Rationale	Poor performance, especially high latency, is a common complaint from end users. Poor performance results in less productivity and is a main contributor to end user dissatisfaction. The end user will blame the user interface for the poor performance even if the actual source of the problem is one (or more) of the backend systems.
Implications	■ The architecture should, where necessary, decouple the backend system response from the user interface response e.g. asynchronous access to backend systems. ■ Performance must not be an afterthought; but rather must be considered throughout the development lifecycle. ■ The performance of the system should be tested in real-world scenarios. Frequently testing is only done on local area networks with low latency and high bandwidth. This type of testing is insufficient for a user interface deployed to the public internet or a wide area network.

3

Industry Standards

This chapter identifies and briefly describes the industry standards that are applicable to the topic of user interaction. Some of the standards focus on defining how components interact while other standards define how individual components behave.

3.1 W3C Standards

The World Wide Web Consortium (W3C) defines the standards that help ensure the World Wide Web works for all. There are a wide variety of web standards under the W3C banner and only a few of the key standards of particular relevance to the architecture described in this document are listed below. Further information, including the full specification, for each of the listed Web standards can be found at http://www.w3.org.

3.1.1 HTTP

The HyperText Transfer Protocol (HTTP) and the related Secure HyperText Transfer Protocol (HTTPS) provide the protocols upon which the Web is based. It is the widespread adoption of these protocols that has made the ubiquitousness of the Web possible. The HTTP standard is a joint effort between the W3C and the Internet Engineering Task Force (IETF).

HTTP(S) is but one of many Internet communication protocols (e.g. SMTP, FTP, Telnet), but it is by far the most important protocol for the user interface.

3.1.2 URI

A Uniform Resource Identifier (URI) is a string of characters used to identify a resource on the Internet. Such identification enables interaction with representations of the resource over a network (e.g. Web) using specific protocols (e.g. HTTP). A Uniform Resource Locator (URL) is a specialization of a URI. In addition to identifying a resource, a URL specifies the means of acting upon or obtaining the representation of the resource. For example, the URL provided above for W3C (http://www.w3.org) specifies that the HTTP protocol be used to access the resource.

The primary functionality for a user interface is to allow the end user to interact with various resources. Hence, the URI specification is essential in that it provides the means for a user interface to find and interact with a wide variety of resources accessible via the Internet.

3.1.3 HTML, XHTML, CSS

The HyperText Markup Language (HTML) specification defines how web browsers use the markup tags to interpret the content of a web page. Cascading Style Sheets (CSS) allow common formatting to be defined once and then used across multiple HTML pages; thereby providing uniformity to the look and feel of a particular web site.

eXtensible HyperText Markup Language (XHTML) is based on the eXtensible Markup Language (XML) (also a W3C specification). Because XHTML is base on XML, XHTML can be parsed by (widely available) XML parsers which provides improvements for searching, indexing, and parsing.

These standards provide the *lingua franca* (at least from a technology point of view) for the Web and, therefore, are essential to browser based user interfaces.

3.1.4 AJAX

Asynchronous JavaScript and XML (AJAX) is not a W3C standard *per se*, but rather is a group of interrelated W3C standards that facilitates the creation of interactive web sites. Using AJAX, a web page can retrieve data from the server asynchronously (in the background) without interfering with the display and behavior of the existing page.

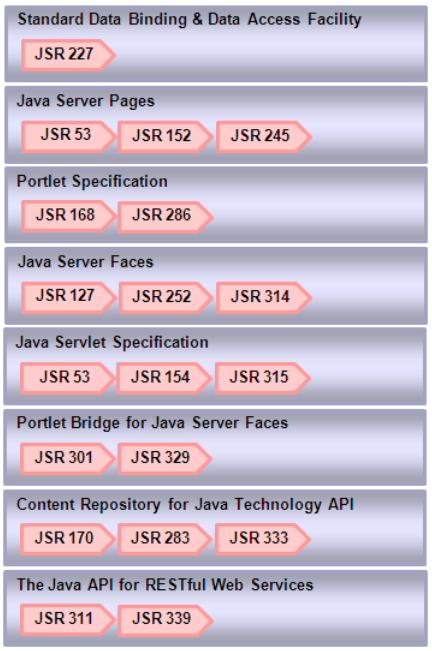
The primary W3C standards incorporated into AJAX include JavaScript, XML, and DOM (Document Object Model). It is also important to note that, despite the name, AJAX does not require XML, but may be based on other data interchange formats, for example, JavaScript Object Notation (JSON).

AJAX (and similar technology) allows the web page to behave (from an end user perspective) more like a rich, desktop user interface i.e. AJAX is a key enabler for Rich Internet Applications (RIAs).

3.2 Java Standards

Java standards are defined by the Java Community Process (JCP) and each standard is numbered. The Java standards that are of particular relevance to the architecture described in this document are listed below. The Java standards are frequently refered to by the number assigned to the Java Specification Request (JSR). Some of the Java standards have undergone several revisions, and most have undergone at least one revision. Figure 3–1 illustrates the Java standards and the associated JSR numbers.

Figure 3–1 Java Standards



The standards shown above are listed below by the latest JSR number and, where applicable, the previous version of the standard has been identified to provide traceability. Further information, including the full specification, for each of the listed Java standards can be found at `http://jcp.org/en/jsr/all`.

3.2.1 JSR 227: A Standard Data Binding & Data Access Facility for J2EE

This specification defines a framework that standardizes the interactions between typical user interface components and classes that publish data objects and provide methods that manipulate the data objects. The framework decouples the user interface from the data portions of the application.

In the nomenclature of the MVC pattern (see Section 4.2.1), this specification standardizes metadata and behavior for binding the model to both the view and the controller.

3.2.2 JSR 245: Java Server Pages 2.1

This standard provides interoperability between Java Server Pages (JSPs) and the container hosting the JSPs. JSPs are focused on simplifying the creation of dynamic content on web pages. This JSR is an update to **JSR 152: Java Server Page 2.0** to better support Java Server Faces.

In web applications, JSPs are frequently used to implement the view in the MVC pattern (see Section 4.2.1).

3.2.3 JSR 286: Portlet Specification 2.0

This standard provides interoperability between portlets and the portal framework by defining a set of APIs between the portal and portal framework. The specified APIs address lifecycle, aggregation, personalization, presentation, and security. This JSR is an update to **JSR 168: Portlet Specification**.

Embracing this standard provides portability between portals and the portal framework into which the portal is deployed. A portal can be developed once and then be deployed to any compliant portal framework. Following this standard also prevents vendor lock-in by allowing the portal framework to be replaced without requiring modification to all the portlets.

3.2.4 JSR 314: Java Server Faces 2.0

This standard provides a server-side, component framework for developing web applications with rich user interfaces. This standard defines a framework of JSP tags and Java classes that simplify building graphical user interfaces and is specifically designed to support graphical development tools. By providing a component framework, this standard facilitates the creation and use of a wide variety of prebuilt, JSF compliant components. This JSR is an update to **JSR 252: Java Server Faces 1.2** which was a minor update to **JSR 127: Java Server Faces 1.0**.

The Java Server Faces standard is based on the MVC pattern (see Section 4.2.1).

3.2.5 JSR 315: Java Servlet 3.0 Specification

This standard provides interoperability between Java Servlets and the container hosting the servlet. Java Servlets were created to standardize programming when following a request-response paradigm. While applicable to any request-response programming, Java Servlets are most widely used in web applications using the HTTP(S) protocol. This JSR is an update to **JSR 154: Java Servlet 2.4 Specification**

In web applications, Java Servlets are frequently used to implement the controller in the MVC pattern (see Section 4.2.1).

3.2.6 JSR 329: Portlet 2.0 Bridge for JavaServer Faces 1.2 Specification

This standard provides the ability to expose a Java Server Faces (JSF) application as a portlet including the ability to remote the portlet using WSRP (see Section 3.3.1). This standard allows the JSF based web interface to be easily incorporated into a portal framework. This JSR is an update to **JSR 301: Portlet 1.0 Bridge for Java Server Faces 1.2**.

3.2.7 JSR 333: Content Repository for Java Technology API Version 2.1

This standard provides a way to access content from a content repository in a standardized way. The specified API focuses on transactional read/write access, binary content (stream operations), textual content, full-text searching, filtering, observation, versioning, and handling of both hard and soft structured content. This JSR is an update to **JSR 283: Content Repository for Java Technology API 2.0** which was an update to **JSR 170: Content Repository for Java Technology API**.

Since ready access to a variety of content is a primary concern in many user interaction scenarios, having a standard way to interact with content repositories is a big advantage. It significantly reduces the effort required to incorporate a content repository and provides a consistent mechanism to access multiple content repositories.

3.2.8 JSR 339: JAX-RS 2.0: The Java API for RESTful Web Services

This standard provides a high-level, easy-to-use API for developing RESTful web services running on the Java platform. The standard provides a declarative style of programming using annotations and also enables low level access in the cases where

needed. This JSR is an update to **JSR 311: JAX-RS: The Java API for RESTful Web Services**.

RESTful web services provide an alternative to SOAP based web services. RESTful web services are more aligned with the stateless request-response paradigm used in the majority of HTTP based web applications.

3.3 OASIS Standards

The Organization for the Advancement of Structured Information Standards (OASIS) has also published some standards that are of particular relevance to the architecture described in this document and are listed below. Further information, including the full specification, for each of the listed OASIS standards can be found at http://www.oasis-open.org.

3.3.1 WSRP 2.0

The Web Services for Remote Portlets (WSRP) specification defines a set of interfaces and related semantics designed to simplify incorporating user interface components exposed via web services; thereby providing a portion of the overall user interface without writing any code. The WSRP standard is layered upon the web services stack and therefore leverages the existing web service standards.

The WSRP specification is a key enabler to federated portals (see Section 4.2.3).

3.3.2 CMIS

The Content Management Interoperability Services (CMIS) specification defines a web service interface providing greater interoperability of content management systems. CMIS defines web services interfaces to enable rich information to be shared, in standardized formats, between document systems, publishers, and repositories.

This standard plays a role similar to the Content Repository for Java Technology standard listed above with the notable difference that it is based on web services rather than Java; therefore it is equally applicable to non-Java environments.

3.4 Other Standards

There are a few other standards that are of particular interest to the user interaction architecture. These standards are listed below:

3.4.1 MIME Types

Multipurpose Internet Mail Extensions (MIME) is an IETF standard that supports: text in non-ASCII character sets, not-text attachments, message bodies with multiple parts, and header information in non-ASCI character sets. The MIME standard is a set of IETF RFCs available on the IETF web site (http://www.ietf.org).

The MIME types standard allows multiple different types of content (e.g. graphics, sounds) to be transmitted and displayed correctly by web browsers and other user interface applications.

3.4.2 Atom

The Atom standard is actually two related standards, the Atom Syndication Format standard and the Atom Publishing Protocol standard. Together these two standards provide the mechanism to allow applications to check for updates of published

content (e.g. blog entries, news items) on a website. The Atom standard is composed of two IETF RFCs available from the IETF web site (http://www.ietf.org).

The Atom standard is one of the approaches used to provide syndicated content (see Section 4.1.4.10) to a user interface.

3.4.3 RSS 2.0

The Really Simple Syndication (RSS) standard provides the mechanism to allow applications to check for updates of published content (e.g. blog entries, news items) on a website. The RSS 2.0 standard is available at http://cyber.law.harvard.edu/rss/rss.html.

The RSS 2.0 standard is a direct competitor to the Atom standard and is also one of the approaches used to provide syndicated content (see Section 4.1.4.10) to a user interface.

4

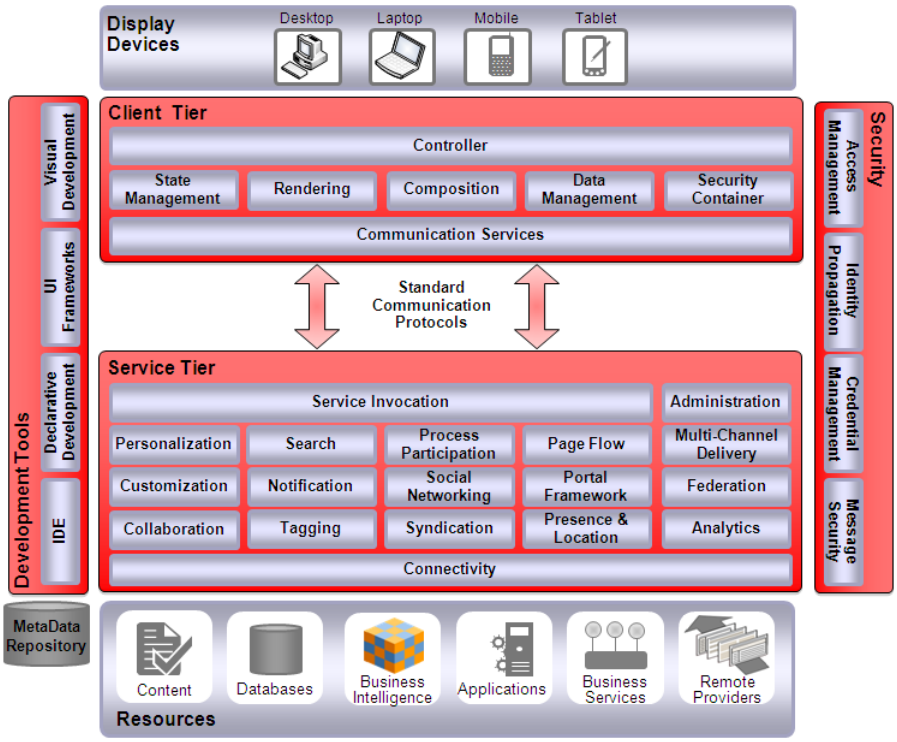
User Interaction Reference Architecture

This section describes a high-level architecture for user interaction. Each section provides a view of the architecture from a particular perspective.

4.1 Logical View

The Logical View of the architecture describes the various layers in the architecture. Each layer encapsulates specific capabilities for the overall architecture. Upper layers in the architecture leverage the capabilities provided by the lower layers. Generally, upper layers call lower layers in the architecture and the reverse (i.e. lower levels calling upper layers) is prohibited. Upper layers are allowed to call capabilities provided by any lower layer and, therefore, may skip any intermediate layers. The Logical View of the architecture is shown in Figure 4–1.

Figure 4–1 Logical View



Each layer in the architecture beginning at the top and progressing to the bottom is described in subsequent sections. The description of each layer does not attempt to

enumerate every possible capability provided by the layer; rather, only the key (essential) capabilities provided by the layer are listed and described.

4.1.1 Display Devices

The display devices are used by the end user to interact with the user interface. Which types of display devices to support is an important design decision for any user interaction architecture. This architecture is designed such that the vast majority of provided capability is display device independent; therefore adding support for additional display devices is not particularly difficult. Display devices include desktop and laptop computers, mobile phones, tablets, etc. The architecture supports both browser based interfaces and client hosted applications.

This is not formally a layer in the user interaction architecture. Rather the entire purpose of the architecture is to provide a feature-rich user interface that can be accessed by a variety of display devices. Conversely, a single display device may host multiple different user interfaces and may access multiple different Service Tiers.

4.1.2 Client Tier

The Client Tier is hosted on the display device. As mentioned above, this may be a browser or an thick client specific to the display device. A browser based interface may be a simple HTML web site, a complex rich-internet application (RIA), or a hybrid of the two. The design of the Client Tier is driven by the requirements of the end user interface with the other layers in the architecture equally supporting any Client Tier choice. In fact, the other layers in the architecture are intended to concurrently support many different types of interfaces in the Client Tier.

Regardless of the choice for the Client Tier, there are standard capabilities provided by this tier in the architecture. These key capabilities are described below.

4.1.2.1 Controller

The Controller accepts input from the user and performs actions based on that input. Nearly all of the actions will result in calls being made to other components in the Client Tier. Many of the actions will result in calls to the Communication Services which will, in turn, result in actions being invoked on the Service Tier of the architecture.

4.1.2.2 State Management

The State Management component is responsible for maintaining the current state of the user interface. For browser interfaces, this is frequently implemented via cookies. For native applications this component may be more complex and may maintain, for example, the current step being executed in a series of user input actions.

The State Management component is not used to maintaining the state of data entities that the user interface operates upon. Data entity state management is handled by both the Data Management capability in the Client Tier and by capabilities provided by the Service Tier.

4.1.2.3 Rendering

The Rendering component is responsible for delivering a view of the interface suitable for the end user. Rendering is a capability provided by the display device via a standard programming interface. Rendering may be provided by a browser (e.g. HTML), a browser plug-in (e.g. Flash), or a native application (e.g. Java Platform, Micro Edition).

4.1.2.4 Composition

Composition provides the end user with the ability to combine different interface content in an ad hoc manner. For browsers, this capability is usually referred to as "mashups" and has gained widespread acceptance as an effective mechanism to empower end users. Ultimately, the goal of Composition is to allow the end user to adjust the way information is combined and presented to better support the needs of the end user.

4.1.2.5 Data Management

Data Management is focused on providing temporary storage of data entities so that they may be efficiently manipulated by the user via the user interface without requiring incessant calls to the Service Tier. Once all necessary changes have been made, the data entities are persisted via calls to the Service Tier. Using client-side Data Management makes the user interface much more responsive by dramatically reducing the number of calls to the Service Tier.

An example of extensive use of this Data Management capability is Asynchronous Java and XML (AJAX). AJAX fetches data from the Service Tier as XML (or another format such as JSON), temporarily stores the data while it is being manipulated via JavaScript, and then posts the modified data back to the Service Tier.

Sophisticated Data Management also facilitates occasionally connected computing (OCC). When used within an OCC architecture, the Data Management component stores all modified data entities until a connection between the Client Tier and Service Tier is (re)established, at which point the changes are propagated to the Service Tier.

4.1.2.6 Security Container

The Security Container provides client-side security services. The Security Container is responsible for supplying user credentials to the Security Tier to facilitate user authentication and authorization. Credentials may be supplied by the end user (e.g. user name and password) or they may be associated with the Client Tier itself (e.g. X.509 certificate).

The Security Container is also responsible for validating any credential provided by the Security Tier. For example, the Security Container must check the certification chain and the root certificate for an X.509 certificate provided by the Security Tier to establish a secure HTTP session.

Additionally, the Security Container is responsible for protecting the Client Tier (and the display device) from malicious attacks originating from the Service Tier. This is vitally important since the Client Tier may access multiple different Service Tiers, some of which might be malevolent. For example, a browser-based Client Tier connected to the internet can access any HTTP server also connected to the internet.

4.1.2.7 Communication Services

The Communication Services provide the means to access Service Tier capabilities. The Communication Services handle message marshalling and demarshalling as well as connection and reconnection as necessary to provide reliable communications with the Service Tier. The Communication Services use standard communication protocols as described below.

4.1.3 Standard Communication Protocols

The Standard Communication Protocols is not technically a layer in the architecture, rather it separates the Client Tier from the Service Tier. Architecturally it is very important for two primary reasons:

- It illustrates that the Client Tier and the Service Tier are separated by a network. The network may introduce many issues including unreliable transport, poor throughput, high latency, and malevolent interlopers.
- The Client Tier and the Service Tier may be completely different technology platforms owned by different authorities. Therefore, adhering to standards-based communication protocols is essential to ensure interoperability.

Even in the case where the Client Tier and the Service Tier are owned by the same authority and reside on an internal network, the architectural significance of separating the Client Tier from the Service Tier should not be ignored since frequently functionality, which was originally designed to support only internal users, ends up with an expanded role to support partners and even customers.

4.1.4 Service Tier

The Service Tier hosts the capabilities that satisfy the requirements of the end user. The Client Tier exists solely to provide a user-friendly interface to the capabilities hosted on the Service Tier. Since the Service Tier is accessed by multiple Client Tier applications concurrently, it is vital that the Service Tier be reliable, meet or exceed the availability and performance (both throughput and latency) requirements of the clients, and be scalable to meet any increase in the number of clients.

From the clients perspective, all functionality comes from the Service Tier. However, in reality, much of the functionality exposed by the Service Tier actually originates in other systems (the Resource Tier), and the Service Tier is only providing easy access to the functionality.

There is a wide variety of capabilities that the Service Tier might provide, and a multitude of ways the capabilities can be organized and quantified. This section describes the key capabilities at sufficient granularity to make the architecture requirements clear without overwhelming the reader with a laundry list of possible features. The key capabilities are:

4.1.4.1 Service Invocation

The Service Invocation capability provides easy access to the Service Tier capabilities. The Service Invocation capability is the counterpart to the Communication Services in the Client Tier. The Service Invocation capability must handle the communication with multiple clients simultaneously and ensure that each client request is routed to the appropriate Service Tier capability.

4.1.4.2 Personalization

Personalization uses specific, implicit and explicit characteristics of the end user (e.g. interests, past behavior, context) to modify the interface to provide a better user experience. Personalization is driven by three types of information:

- End user profile information such as organization, role, demographic, zip code, etc.
- Past behavior of the end user such as previous purchases, articles accessed, etc.
- Current behaior such as responding to an image or clicking a link.

- Current context independent of the end user such as day of the week, time of day, current specials, open enrollment period, etc.

Personalization is generally based on a deterministic, rules-based process. The totality of rules applied to determine the personalization may be arbitrarily complex. Personalization may also be based on collaborative filtering where the personalization for an end user is determined by the actions and interests of "similar" users.

4.1.4.3 Customization

Customization is based on end user selections and explicit information provided by the end user. For this architecture, localization is considered a type of customization i.e. the end user selects the desired local and the interface changes to reflect the chosen local. Customization allows the end user to select particular display themes; move, remove, or add display elements; set default display settings; etc.

In addition to end user customization, customization may provide an easy mechanism to deliver a custom interface via administrative configuration. Private labelling is an example of this type of customization. Private labelling takes a standard set of functionality but provides a custom interface by changing the layout, color schemes, etc. to create a seemingly unique experience.

4.1.4.4 Collaboration

Collaboration encompasses all of the current collaboration focused capabilities including email, calendar, contacts, forums, discussions, blogs, wikis, instant messaging, announcements, etc. A unified, integrated collection of collaboration capabilities simplifies and enhances the user experience. For example, what starts out as an email exchange might transition into a discussion to include more participants and record the exchange for future reference. Collaboration capabilities should allow the end users to work together effortlessly without having to jumping from interface to interface.

4.1.4.5 Search

The Search capability provides the ability to search for specific content items across all content sources based on user defined search criteria such as content type, topic, key words, date, etc. The Search capability needs to be integrated with the Security Tier since end users must only be allowed to search over content they are entitled to access.

4.1.4.6 Notification

Notifications provide the means to push information to the end user. Instead of supporting only the simple request-response paradigm, notifications allow the Server Tier to initiate actions on the Client Tier. For example, a chart or other visual display of information can update automatically when new data is received and processed by the Service Tier.

It should be noted that the push is from the end user perspective i.e. the end user receives a notification without taking any action to find the notification. The actual implementation may require that the Client Tier regularly poll the Service Tier to find pending notifications. While polling is a possible implementation for notifications, a Service Tier push is a much more efficient solution and, therefore, is the preferred approach.

4.1.4.7 Tagging

The Tagging capability allows the end user to "tag" content so that it can be easily retrieved at a future date. The Tagging capability includes "tag clouds" where content

tagged by others can also be found based on the tags that have been applied to the content. Tag clouds are similar to key words except that tags are supplied by consumers of the content; whereas key words are supplied by the providers of the content. For this architecture, content rating is considered a type of tag i.e. the end user "tags" the content with a particular rating (e.g. four out of five stars).

4.1.4.8 Process Participation

The Process Participation capability allows the end user to interact with managed business processes (i.e. business processes that are under the control of a business process management (BPM) system). Using this capability the end user may initiate a business process or participate in an existing business process. An example of this capability is a "worklist". The worklist displays a list of tasks that the end user is expected to perform as part of a larger business process. The Process Participation includes the worklist capability but also includes other types of interaction with the business process including the ability for the end user to monitor and manage business processes. The actual functionality exposed to each end user is determined by the entitlements for each end user.

4.1.4.9 Social Networking

The Social Networking capability allows end users to locate and establish relationships with other end users based on similar interests, activities, locations, organizations, etc. Once a relationship is established, the connected end users usually employ the Collaboration capabilities to communicate and collaborate on items of interest.

4.1.4.10 Syndication

The Syndication capability makes it possible to easily share material between multiple sites. Syndication requires both a sending site and one or more receiving sites. The receiving site(s) must subscribe to the content desired from the sending site(s). The two most common forms of web site syndication are Really Simple Syndication (RSS) and Atom Syndication Format.

4.1.4.11 Page Flow

The Page Flow capability determines the order in which interface elements are presented to the end user. For example, when completing a multi-part form, the Page Flow capability orders the steps that the end user goes through to fill out the form. The flow of information to the user or the flow of input from the user are both handled by this capability. Generally, the flow is determined at development time with each possible end user action, and error conditions, built into the prescribed flow. An example browser-based implementation of this capability is Apache Struts, where the struts.xml file determines the flow of actions taken based on user inputs. Java Server Faces (JSF) is another exmple that also uses an XML file to define the flow of actions.

This capability is also sometimes called "task flow", which, when viewed purely from the UI perspective, is probably more accurate since, for example, the configured flow may be displayed within a portlet inside a larger page. However, this document is part of a larger architecture (Oracle Reference Architecture) which uses the term "task flow" as part of the business process management descriptions. Hence, the term "page flow" is used here to keep an explicit distinction between a flow associated with the user interface and a flow associated with a business process.

4.1.4.12 Portal Framework

The Portal Framework capability provides the structure and layout of the interface at runtime. The Portal Framework supports page templates as well as dynamic page

creation. Templates allow for easy reuse across pages and sites while dynamic page creation allows a page to change based on the content that needs to be displayed. The Portal Framework also provides inter-portlet and inter-component communication; thereby allowing display elements to react to changes or user input in other display elements.

4.1.4.13 Presence & Location

Presence and Location provide information about the end user of the display device. Presence determines whether or not the end user is actively using the device by detecting user actions (e.g. key strokes). Location determines the current location of the display device. For mobile devices this is particularly useful and is generally based on GPS capability built into the display device. For laptops and desktops, a less accurate determinations of location can be based on the IP address of the device.

Presence and Location are shown as Service Tier capabilities even though the actual determination of presence and location is done via the display device. The display device informs the Presence and Location capabilities of the current state. This allows other Service Tier capabilities (e.g. Notification, Search) to make use of the information to create a better user experience.

4.1.4.14 Administration

The Administration capability allows users who are assigned “administrator” privileges to perform a variety of functions including: add or remove resources from pages, re-arrange page layouts, set various page and component properties, and delegate administration functionality to other users. The Administration capability provides the ability to make runtime changes to the user interface, effectively eliminating the need to engage in development activity for configuration changes to the interface.

4.1.4.15 Multi-Channel Delivery

The Multi-Channel Delivery capability facilitates the delivery of shared content and functionality to multiple different types of display devices without requiring specific development for each type of display device. The need to support delivery of content and functionality to multiple channels has grown as the mobile devices have become more powerful and prevalent. Multi-channel delivery also encompasses multiple applications such as browser, email, instant messaging, short message service (SMS), etc.

For mobile devices, transcoding is frequently used to support the wide variety of device types. Transcoding is used when the target device does not support the original content format or has limited storage capacity that mandates a reduced file size, or to convert incompatible data to a supported format.

4.1.4.16 Federation

The Federation capability allows display elements hosted elsewhere (i.e. hosted remotely) to be incorporated into the user interface. Web Services for Remote Portlet (WSRP) is an example of a standard created to support federation of portlets in web pages. Web clipping is also sometimes used to incorporate content from remote sites into a web page, for example, when WSRP is not an option or when the remote site was not designed with federation in mind.

4.1.4.17 Analytics

The Analytics capability tracks how end users navigate the use interface and provides the ability to analyze this information. Analyzing this information allows the interface designer(s) to improve the user experience.

4.1.4.18 Connectivity

The Connectivity capability allows the other capabilities in the Service Tier to access to backend resources for data, content, functionality, etc. The Connectivity capability must handle the connection to backend resources on behalf of multiple concurrent end users. Generally, this is handled by providing connection pools that allow connections to backend resources to be used and reused as necessary without burdening the backend resources with either too many connections or repeatedly instantiating and removing connections.

4.1.5 Resources

The Resources Tier contains the backend sources that are accessed by the Service Tier to provide needed content, data, functionality, etc. The Resources Tier is not formally part of the User Interaction architecture because these sources exist independent from the User Interaction capabilities. Nonetheless, access to these resources is frequently the primary reason that the User Interaction architecture is being constructed i.e. to provide a user friendly interface to these existing resources.

4.1.5.1 Content

Providing access to a variety of content is generally a requirement for user interfaces. The content may be text, images, sound, documents, etc. Usually the content is managed within a content management system (CMS), but the content may also be stored in a file system with little management or structure imposed.

4.1.5.2 Databases

Almost without exception, one or more databases supply the data that the end users interact with via the user interface. The database(s) provide create, read, update, and delete (CRUD) functionality for the data. Databases also help to ensure that the data remains uncorrupted via transaction control, referential integrity, etc.

4.1.5.3 Business Intelligence

Business intelligence (BI) has been around for years and is continually gaining importance as more and more data is collected, stored, and managed. BI provides the means to derive value from the vast quantities of data that can now be captured. BI used correctly provides insights into the business that allows the business to better address the needs of their customers.

4.1.5.4 Applications

Frequently desired functionality that is to be exposed via the user interface already exists within one or more existing applications. There are a variety of applications that may contain desired functionality including legacy applications, packaged applications, and custom applications.

Integration to existing applications is a common requirement for user interfaces and a continuing struggle for IT departments. The most common approach to integrating applications is via point-to-point integration. While this solves the immediate needs of the user interface, it inevitably results in future maintenance issues; therefore, a

service-oriented approach (as described in ***ORA Service_Oriented Integration***) is the preferred approach to application integration.

4.1.5.5 Business Services

Business Services also provide desired functionality to the user interface. Business Services are specifically designed and built to be easily used and reused across the IT environment; therefore, they provide an easy means to incorporate functionality into the user interface.

The ***ORA SOA Foundation*** document describes different types of SOA Services that might be created as part of an SOA including Business Process Services, Business Activity Services, Data Services. This document uses the term Business Service generically to represent any of these types of SOA Services. The term Business Service is used instead of SOA Service to make it clear that the real value to the user interface architecture is business functionality (as opposed to lower-level technical functionality).

4.1.5.6 Remote Providers

Remote Providers refers to sources of user interface elements that are hosted separately from the user interface architecture. This may be interface elements that are specifically designed to be remotely accessed (e.g. WSRP portlets) as well as existing interface elements that were never intended to be incorporated remotely (e.g. screen scraping of an existing application or website).

Remote Provides differ from the other types of resources listed in this layer because the presentation elements of the remote system are directly incorporated into the user interface. Remote Provides are instrumental in the creation of federated portals. The Presentation Service as defined in the ***ORA SOA Foundation*** document is an advanced form of Remote Provider since it is specifically designed and built for remote use and reuse.

4.1.6 Development Tools

This layer provides the tools that enables designers and developers to create user interfaces. While focused on creating the user interface, it is imperative that these tools also provide the ability to create new functionality and, more importantly, incorporate existing functionality. This section describes the key development capabilities that are of particular relevance to this architecture, but does not attempt to identify all the capabilities required for a enterprise development environment. Please refer to the ***ORA Software Engineering*** document for a description of a complete development environment.

4.1.6.1 IDE

An integrated development environment (IDE) is the primary tool used by designers and developers to create a user interface. The IDE includes designing, source code editing, compiling, debugging, build automation, integration with source control, etc. An important IDE capability for user interface tools is that content displayed during editing matches the final output. This capability is frequently referred to as what-you-see-is-what-you-get (WYSIWYG) editing.

4.1.6.2 Declarative Development

Declarative development allows the developer to specify what should happen without having to write the code that actually makes it happen. There are many functional areas where declarative development applies. Source code containers allow the

developer to user the container provided services without coding that functionality. Source code containers provide transaction control, security, connection pooling, thread pools, etc.

Likewise, the developer defines what and where visual elements should appear without writing any graphics code. For example, HTML is a mark up language that allows the developer to "declare" what should appear without providing any graphics code. For HTML, the graphics code is provided by the browser. Cascading style sheets (CSS) allow a designer to create and modify the look-and-feel (i.e. colors, fonts, text sizes, border widths, etc.) of an interface without changing the contents of the interface.

4.1.6.3 UI Frameworks

The IDE is generally a multi-purpose development environment that is not solely dedicated to UI development. The UI framework is added into the IDE to provide capabilities specific to the type of UI being developed. If the UI is being developed for a specific device (e.g. iPhone, BlackBerry, Android) then the UI framework provides the extra capabilities to easily develop for that platform.

Many different UI frameworks also exits for browser based user interfaces. Some UI frameworks provide little more than a UI design pattern (e.g. Struts) while others provide a rich set of display components [e.g. Oracle Application Development Framework (ADF), Adobe Flash].

4.1.6.4 Visual Development

Visual development is an extension of the WYSIWYG editing mentioned above. Whereas WYSIWYG allows the developer to readily see what the interface will look like, visual development allows the user to work with display elements directly by simply selecting and positioning display elements on the screen. Visual development hides the underlying details of the visual elements, allowing the developer to more easily and rapidly create the user interface.

While visual development is a great productivity enhancer, it is important that, when necessary, the developer has access to the configuration/code that is created by the visual development environment. This access is frequently called the "source code view" and allows the developer to investigate and fix problems in the user interface.

4.1.7 MetaData Repository

The metadata repository allows configuration data to be stored and manipulated separately from the source code for the user interface which allows for rapid configuration changes without the need to re-compile and re-deploy source code. There is a wide variety of metadata that can be stored in the metadata repository, and the metadata stored is entirely dependent on the implementation characteristics of the user interface. The configuration for access to server side resources is a common type of metadata. For example, the location of SOA Services accessed by the user interface should be stored as metadata in the repository.

Using the metadata repository to store configuration data facilitates support for different IT environments (e.g. development, test, production) since the differences between the environments can be stored as metadata and then easily changed when moving from one environment to another.

This logical representation of the metadata repository does not make a distinction between design time and run time for metadata access. The term "repository" is often associated with design time whereas "registry" is often associated with run time. The

metadata repository as defined here is equally applicable to both design time and run time.

Frequently the metadata repository uses a database as the backing store. However, this is not required: the metadata may be persisted via another mechanism, for example, as files on a standard file system.

4.1.8 Security

The Security Layer provides end-to-end security for this architecture. This section describes the key security capabilities that are of particular relevance to this architecture, but does not attempt to identify all the security capabilities required for a secure enterprise IT environment. Please refer to the ***ORA Security*** document for a description of a complete security architecture.

4.1.8.1 Access Management

Access management allows authorized end users access to the appropriate data and functionality from the user interface. This includes both the initial authentication of the end user via some type of security credential (e.g. user name and password, biometrics) as well as the authorization to access specific components based on attributes of the user (e.g. roles, groups).

4.1.8.2 Credential Management

Credential management is responsible for managing the credentials that are used by the access management to authenticate the end user. In most IT environments there is a centralized credential management that allows all end user security credentials to be managed by a central authority. The credential management is also responsible for enforcing any security policies associated with the credentials such as password aging or password restrictions (e.g. number of characters, must include upper case, lower case, and numbers)

4.1.8.3 Identity Propagation

Identity propagation allows an end user to authenticate once and have their security identity propagate to other backend systems. This allows the data and functionality contained in the backend systems to be securely accessed by only end users with the appropriate security entitlements. It is still the responsibility of the backend systems to enforce security restrictions based on the identity provided, but the end user is insulated from the fact that these additional security checks are taking place.

4.1.8.4 Message Security

The Client Tier frequently accesses the Service Tier via a public network; thus, ensuring that all messages between the two tiers are securely transmitted is essential to the security of the entire architecture. Providing message security entails providing message integrity and confidentiality. Message integrity ensures that the message has not been modified during transit. Message confidentiality prevents intermediaries from inspecting the message contents by encrypting the message during transit. Message security is most commonly provided via transport layer security (TLS) which is also the basis for secure HTTP (HTTPS).

4.2 Development View

The Development View of the architecture describes aspects of the architecture that are of interest to developers building assets that conform to and leverage the architecture.

There are three primary concerns that developers following this architecture should strive to conform with: the model-view-controller (MVC) pattern, modular development, and federation.

4.2.1 MVC Pattern

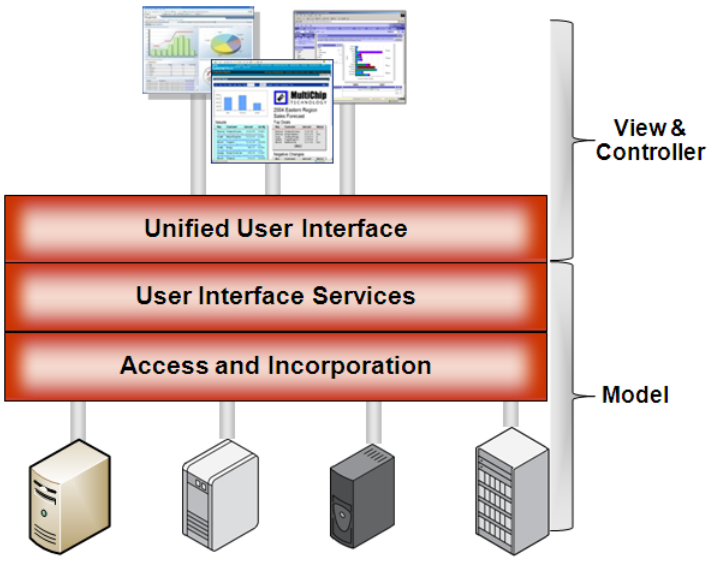
The model-view-controller pattern separates the three major elements in the user interface; thereby providing separation of concerns which results in code that is more easily understood, reused, modified, and maintained. The three major elements in the user interface are: model, view, and controller.

- The model manages the behavior and data for the system. The model responds to requests for information about its state from the view, and responds to instructions from the controller to change state or execute behavior. The model may also notify the view when information changes so that the view can update the effected interface elements.
- The view renders data from the model into a suitable interface element. Multiple different views may exist for a single model element. Each view element is designed to support a particular end user need; whereas the model elements are shared by all users of the system.
- The controller receives input from the user and makes the corresponding calls on model elements. The response to the user input is displayed via the view elements. A single user input may result in changes to none, one, or many view elements.

When the MVC pattern came into prominence, client-server was the system architecture *de rigueur*. In a client-server architecture, the mapping of model, view, and controller is fairly straight forward: the model is mapped to the server (usually a database) while the controller and view map to the client. With the advent of multi-tier systems and web technologies, the mapping for the model, view, and controller became much more complex.

For this architecture, mapping the model, view, and controller to the Conceptual View (Figure 1–3) is still simple as illustrated below:

Figure 4–2 MVC Mapping to Conceptual View



However, mapping model, view, and controller to the Logical View (Figure 4–1) is much more difficult. In general, the model maps to the Service Tier. The view and controller map to the Client Tier for a thick client but map to the Service Tier for thin clients or simple browser based interfaces. Rich internet applications (RIA) move some (but usually not all) of the view and controller functionality to the Client Tier. However, the mapping is not that clean for many reasons. Some examples why the mapping is more complicated:

- When in a disconnected state, the Data Management capability in the Client Tier may act temporarily as the model allowing the user interface to function even when the Service Tier is not available.
- Remote Providers in the Resources Tier include all three aspects (model, view, and controller) in a single item on the diagram. A Remote Provider is a complete user interface, it is just incorporated (as is) into a larger user interface.

Although the mapping of model, view, and controller is complicated, this does not lessen the desire to maintain a separation between these three concerns; in fact, the complexity of a modern user interface requires greater emphasis on encapsulation and separation of concerns or the system can rapidly become unmaintainable. Therefore, it is incumbent upon developers to understand and follow the tenets of the MVC pattern to help create a maintainable system. Many UI frameworks incorporate the MVC pattern which helps developers follow this important construct.

4.2.2 Modular Programming

The concept behind modular programming is to create separate, interchangeable, reusable pieces (modules) of software. As applied here, modular programming is specifically focused on interface elements. Although each user interface is unique, there is rarely a need to create entirely new interface elements to implement the user interface. There are a wide variety of interface elements that can be developed once and used repeated in various user interface designs. Reusable interface elements include tables, charts, text fields, drop-down menus, etc.

There are two ways in which modular programming applies to user interface design. The first is for developers to take full advantage of existing interface elements; thereby reducing development time and effort. Using existing interface elements also dramatically reduces the number of software defects since the interface elements are already fully tested.

Modular programming should also be considered in the event that a new interface element is required. The new interface element should be constructed with (eventual) reuse taken into consideration. When creating a new interface element, it is essential that the MVC pattern described above be fully employed.

4.2.3 Federation

Whereas the MVC pattern and modular programming are relatively old concepts (at least as far as software development is concerned), federation is a relatively new concept closely related to service orientation. Applied to user interfaces, federation is the concept that parts of the user interface are created and controlled by an organization (authority) that is separate from the organization (authority) creating the user interface.

Much as in modular programming, there are two ways that federation applies to user interface design: as either consumer or producer of federated interface elements. As a consumer the developer is responsible for incorporating existing interface elements into the user interface. The Remote Providers capability (Section 4.1.5.6) shown in the

Logic View (Figure 4–1) is the concrete manifestation of these existing interface elements.

If the developer is creating an interface element that is expected to be (or could be) accessed remotely (i.e. a Remote Provide for a different user interface), the developer must take that remote access into account, making sure that the interface element is truly separate and independent so that remote access does not cause unintended consequences.

Federation is a powerful capability when applied to an company-wide initiative. For example, a company may have multiple independent lines of business. A customer may do business with many of the distinct lines of business. Using federation, the company can provide a user interface that incorporates elements from each of the lines of business into a single view for the customer of all their business with the company. Yet, each line of business can provide (and be responsible for) the specific interface elements that detail the customer's state for that particular line of business.

4.3 Process View

The Process View describes the computer processes incorporated into the architecture and illustrates the interactions between the various components in the architecture.

4.3.1 Simple User Request

Figure 4–3 illustrates a simple request from the end user that retrieves data from a database and displays the result.

Figure 4–3 Simple User Request

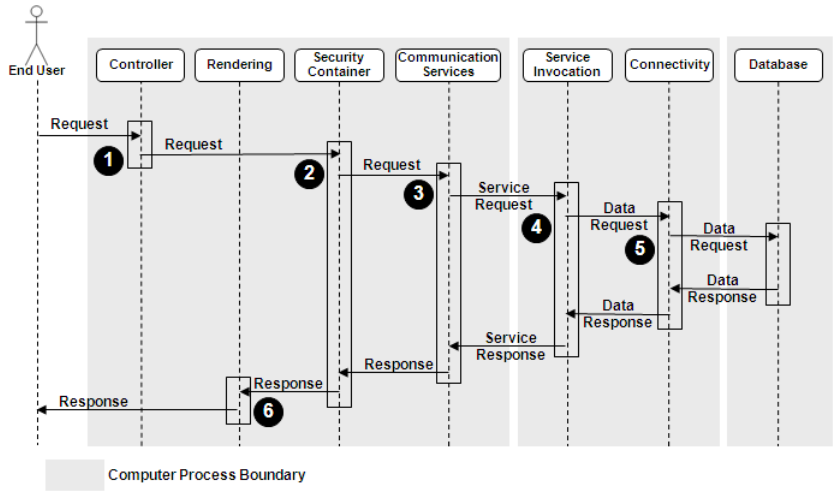


Figure 4–3 illustrates an example of how the various components interact to service a user request and does not correspond to any particular product. Descriptions for the numbers shown in Figure 4–3 are as follows:

1. The user request is handled by the Controller capability. The Controller directs the request to the appropriate Service Tier capability. This follows the MVC pattern.
2. All interactions with the Service Tier flow through the Security Container to ensure that appropriate security is maintained for all user requests.
3. All communication with the Service Tier is handled by the Communication Services capability in the Client Tier.

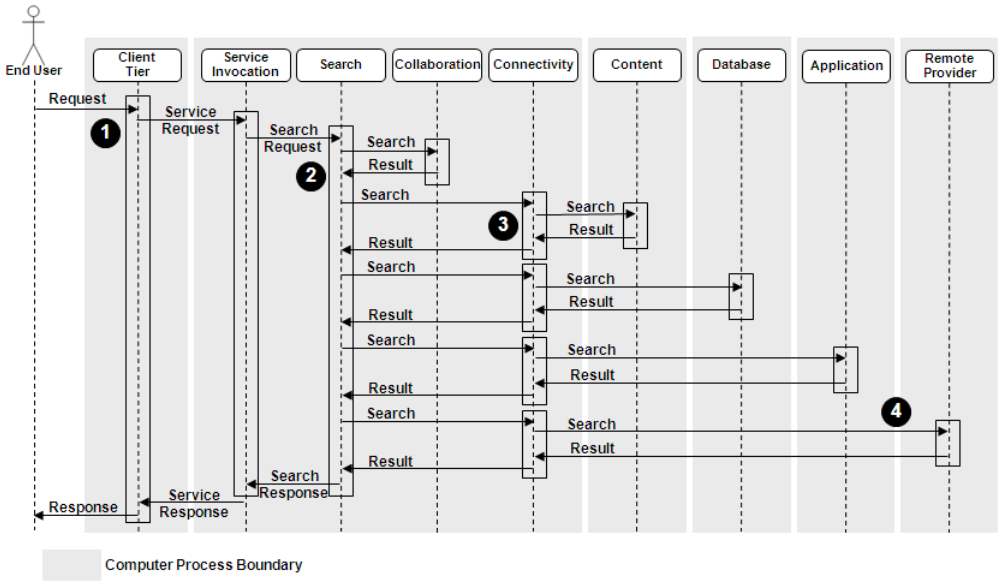
4. The request from the Client Tier is handle by the Service Invocation capability on the Service Tier. This is the model portion of the MVC pattern.
5. Since this user request requires assessing a database, the Connectivity component is invoked to provide a connection to the database instance.
6. Ultimately the response to the user request is displayed in a visual element by the Rendering capability on the Client Tier. This completes the view portion of the MVC pattern.

Figure 4–3 also shows the process boundaries that must be crossed in order to service the user request. In this example there are three processes involved: the Client Tier process, the Service Tier process, and the process for the database instance.

4.3.2 Search Request

Figure 4–4 illustrates a more complex Service Tier interaction. The figure illustrates the interactions required to fulfill a user search request.

Figure 4–4 Search Request



Descriptions for the numbers shown in Figure 4–4 are as follows:

1. For this example the internals of the Client Tier have been encapsulated into a single item since the Client Tier interactions are the same as for the previous example.
2. To handle the user search request, the Service Invocation capability invokes the Search capability. The Search capability then evaluates the user request and begins to search across all the necessary (and searchable) resources. In this example, the Collaboration capability is included in the search to find blogs, wikis, discussions, etc. that match the search criteria.
3. The Connectivity capability is used to access resources for searching.
4. There are a wide variety of resources that may be included in the search. This example shows a content repository, a database, an application, and a remote provider as the targets of the search.

The process boundaries are also illustrated for this example interaction. The number of processes involved will depend on the number of different resources that the search request must cover.

4.4 Deployment View

There are a myriad of ways that the architecture can be deployed within an enterprise. The types and number of physical servers is determined based on company preferences and expected computational load. The actual products that are used to realize the architecture may also have significant impact on the deployment choices.

4.4.1 Physical Deployment

Figure 4–5 provides a high-level view of how the architecture might be deployed to hardware.

Figure 4–5 Physical Deployment

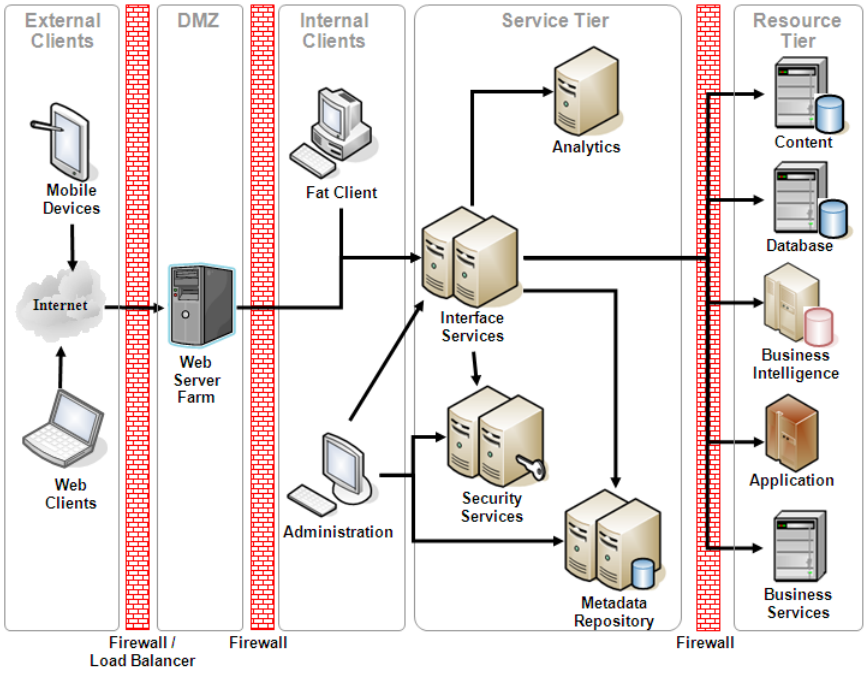


Figure 4–5 shows the architecture deployed to multiple hardware boxes to provide scalability and reliability. In a modern data center where server virtualization is widespread, the architecture would be deployed to virtual servers. The number of virtual servers hosting each component in the architecture would be determined by the actual load on the component and could be dynamically provisioned as necessary.

The example deployment shows both external and internal clients accessing the interface. The external clients use the public internet to access the interface via a web server farm deployed to a demilitarized zone (DMZ). External clients access the web server farm through a firewall and load balancer. The firewall protects the web servers from malicious “netizens”, and the load balancers distribute the load across the web servers.

Internal clients are inside the DMZ and access the interface directly. The internal client shown is a fat client, but could just as easily be a browser-based client. This example

also shows an administration client that is used to perform a variety of administration functions.

This example shows a separate server for the Analytics capabilities with the Interface Services pushing usage data to the analytics server for subsequent, off-line analysis. This is but one example of how different Service Tier capabilities might be deployed to dedicated servers to off load the compute needs for that capability and isolate those compute needs from the rest of the interface. Some capabilities that might be deployed to dedicated servers include Analytics, Search, and Collaboration.

In this example, the Resource Tier is separated from the Service Tier by another firewall. This is a common network topology used to provide greater protection to the resources in the Resource Tier since these resources are generally business critical systems containing company confidential information.

As mentioned above, the detailed hardware deployment decisions may be heavily influenced by the specific products and technologies used to realize the architecture. A more detailed, product specific hardware deployment illustration is contained in Section 5.4.

4.4.2 Federated Deployment

Federation was discussed in the Developer View as an important consideration for this architecture. Figure 4–6 illustrates a conceptual deployment for a user interface based on federation.

Figure 4–6 Federated Deployment

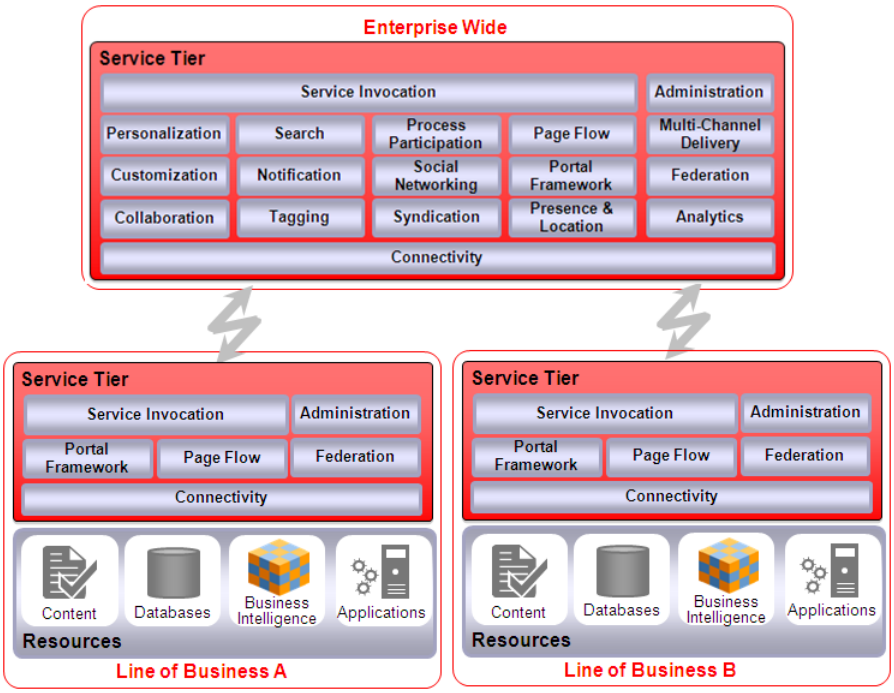


Figure 4–6 shows two separate lines of business (A and B). Each line of business has its own resources that are unique to the line of business and are controlled by that line of business. The enterprise wants to provide a single user interface that, at least from the user's perspective, unifies the separate lines of business.

In this example, the enterprise wide user interface deployment is a full featured user interaction architecture (i.e. it contains all of the capabilities defined in the Logical

View). Each line of business deploys limited functionality since the only functionality required is the functionality to create interface elements exposing the resources of that line of business. The enterprise wide user interface then uses the interface elements provided by the lines of business to create a unified user experience.

The interface elements provided by the lines of business are Remote Providers (Section 4.1.5.6) to the enterprise user interface. This deployment allows the lines of business to maintain control of their respective resources since the only access to the resources is via the interface elements that they create.

5

Product Mapping

This section describes how Oracle products can be utilized to realize the User Interaction Architecture defined in Chapter 4.

5.1 Products Included

The following Fusion Middleware[1] products are included in the product mapping:

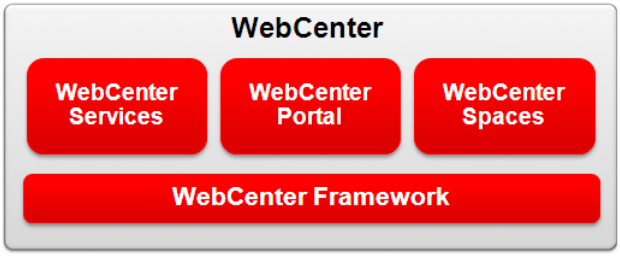
- **Java Platform, Micro Edition (Java ME)** - meets the needs of developers creating applications for the consumer and embedded markets. No other technology provides such robust applications across so many types of size-constrained wireless and wireline devices, from mobile phones and PDAs to set-top boxes and vehicle telematics.
- **Java Platform, Standard Edition (Java SE)** - enables developers to create secure, portable, high-performance applications for a wide range of desktop computing platforms including Macintosh, Linux, Microsoft Windows, and Oracle Solaris.
- **Oracle Access Manager (OAM)** - provides an identity management and access control system that is shared by all applications. It offers a centralized and automated single sign-on (SSO) solution for managing who has access to what information across IT infrastructure.
- **Oracle HTTP Server (OHS)** - provides a HTTP listener for Oracle WebLogic Server and the framework for hosting static content, dynamic content, and applications over the Web.
- **Oracle Internet Directory (OID)** - serves as the central user profile information repository, providing a scalable, secure, high-performance LDAP data store for enterprise applications.
- **Oracle JDeveloper** - is a complete IDE for Java solution development. It provides visual and declarative tools for JSF, JSP, EJB, ADF, Oracle Fusion Middleware, and Oracle application development. Oracle JDeveloper provides the visual editing, wizards, and declarative tools necessary for developers to rapidly create modern user interfaces.
- **Oracle Meta Data Services (OMDS)** - stores customization, personalization, and other metadata in a repository. The repository can either be stored in a database or in file-based storage.
- **Oracle Platform Security Services (OPSS)** - provides security APIs for Java SE and Java EE platforms that insulate application developers from underlying identity systems and allows them to focus on business logic. OPSS is an

[1] The product mapping is for the 11g release of Oracle Fusion Middleware.

enterprise-grade, standards-based, and portable security platform used by all Oracle Fusion Middleware 11g components and Oracle Fusion Applications.

- **Oracle Secure Enterprise Search (OSES)** - provides search access to data sources (e.g. Web sites, file servers, content management systems, applications, business intelligence, and databases) while protecting sensitive data from unauthorized users.
- **Oracle WebCenter (OWC)** - provides the foundation for delivering a modern user experience for Oracle Fusion Middleware as well as Oracle Fusion Applications. OWC is composed of four main components as illustrated in Figure 5–1:

Figure 5–1 WebCenter



  - **Oracle WebCenter Framework (OWCF)** - provides the ability to create composite applications bringing together transactional applications, content management, and collaboration. OWCF enables developers to compose federated user interface components directly into their web applications by providing support for portal producers that can be accessed, deployed, and managed centrally.
  - **Oracle WebCenter Portal (OWCP)** - is a high performance portlet engine that provides the ability to layout a site structure, secure site resources, provide multi-level delegation model, and deliver personalized user experience built directly into Java Server Faces. OWCP is the convergence of the best capabilities from WebLogic Portal, Sun Java Portal Server, and Oracle Portal and is designed to provide an upgrade path for each of the portal products.
  - **Oracle WebCenter Spaces (OWCSp)** - provide a set of templates and site building and management tools to enable micro-sites tailored by business users to target a specific group of users for information sharing, collaborative team projects, and social connections.
  - **Oracle WebCenter Services (OWCS)** - provides ready-to-use components and portlets enriching applications with content, communication, and collaboration capabilities including document libraries, wikis, blogs, forums, presence, etc. Example services include:
    - **WebCenter PageLet Producer** provides integration of non-standard web sites and includes a robust reverse proxy capability. This is a rebranding of WebCenter Interaction Ensemble.
    - **WebCenter Personalization Server** provides industry leading declarative and dynamic personalization of web sites including recommendations based on past activities.
    - **WebCenter Composer** allows business users to customize any page using using visual page editing.

  - **WebCenter Social Computing Services** provide a wide range of social computing capabilities including announcements, discussions, syndication, tagging, wiki, etc.
  - **WebCenter Analytics** provides metrics for pages, portlets, content usage, response times, etc. which are displayed flexible reports. This is a rebranding of WebCenter Interaction Analytics

- **Oracle WebCenter Anywhere (OWCA)** - brings the benefits of a unified work environment to all types of mobile technologies. **ADF Mobile** adds mobile-specific extensions allowing developers to build JSF-based applications with the rich component set rendered appropriately for small-screen mobile devices. Additionally, OWCA provides access to WebCenter applications directly from Windows Desktop tools.
- **Oracle WebCenter Real-Time Collaboration (OWCRTC)** - improves efficiency and productivity by enabling users to connect and collaborate with others via web and voice conferencing, instant messaging, presence, and chat rooms. It complements collaboration services available in Oracle WebCenter by connecting users who require direct interaction and immediate response.
- **Oracle WebCenter Intelligent Collaboration (OWCIC)** - automatically discovers people with relevant expertise and securely and privately brokers business connections between parties to ensures the right people participate in key business activities at the right time.
- **Oracle WebLogic Suite (OWLS)** - fully implements the latest Java EE standards, provides industry leading reliability and performance, and includes comprehensive management capabilities. OWLS can be used to host the Oracle WebCenter products.

Each of the listed products provides key capabilities to the User Interaction Architecture. However, not every organization will require all the capabilities these products provide. The user interaction requirements for each organization must be evaluated and the proper products included in the architecture.

5.2 Products as Resources

In addition to the products that provide the key capabilities described for the User Interaction Architecture, there are also many Oracle products that could, and frequently are, resources used by this architecture i.e. products that fit in the Resources Tier as described in Chapter 4.1.5.

5.2.1 Content

Oracle Universal Content Management (OUCM) is a unified enterprise content management system that provides centralized document management, Web content management, digital asset management, and records retention functionality. Through user-friendly interfaces, roles-based authentication, and security models, OUCM allows users throughout the enterprise to create, view, update, or retire content; thus ensuring that all published information is secure, accurate, and up-to-date.

5.2.2 Databases

Oracle Database provides comprehensive features to easily manage the most demanding transaction processing, business intelligence, and content management applications. There are a variety of versions of the Oracle Database ranging from **Oracle Database Standard Edition One** focused on affordability for small user groups

up to the **Oracle Exadata Database Machine** providing extreme performance for both data warehousing and online transaction processing.

5.2.3 Business Intelligence

Oracle provides a comprehensive range of BI capabilities in a family of products. A few of these products are listed below:

- **Oracle Business Intelligence Foundation Suite** is a complete, open, and architecturally unified business intelligence solution for the enterprise that delivers best in class capabilities for reporting, ad hoc query and analysis, OLAP, dashboards, and scorecards.
- **Oracle Business Intelligence Enterprise Edition** is a comprehensive business intelligence platform designed for scalability, reliability, and performance that delivers a full range of analytic and reporting capabilities.
- **Oracle Essbase** is the industry-leading multi-dimensional online analytical processing (OLAP) server, providing a rich environment for effectively developing custom analytic and enterprise performance management applications.

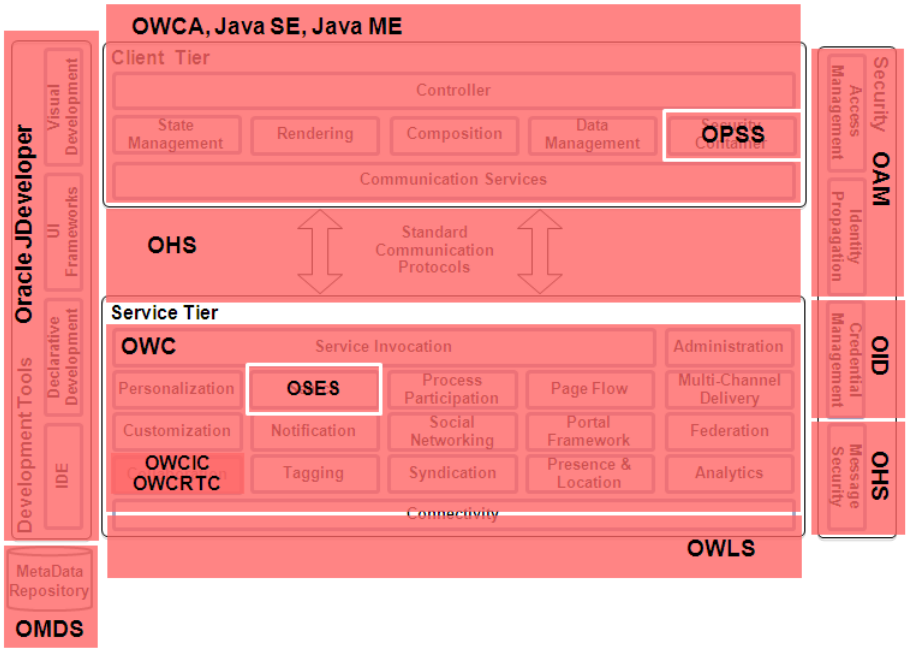
5.2.4 Applications

Oracle provides a wide variety of applications which are commonly incorporated into the user interface. Both the next-generation **Oracle Fusion Applications** as well as the **Oracle Applications Unlimited** (e.g. **Siebel**, **PeopleSoft**, **JD Edwards**) can be readily integrated into the User Interaction Architecture.

5.3 Product Mapping

Figure 5–2 illustrates how the products listed in Section 5.1 can realize the User Interaction Architecture described in Chapter 4.

Figure 5–2 Oracle Product Mapping



The Client Tier can be browser-based, desktop, or mobile applications. Java SE applies to desktop and browser-based applications. Java ME applies to mobile applications. OWCA applies to browser-based, desktop, and mobile applications. While Java ME and Java SE provide Java security capabilities, OPSS provides security capabilities that extend the core Java security capabilities and tie the security into the enterprise security capabilities.

OHS provides the standard communication protocols (e.g. HTTP) between the Client Tier and the Service Tier as well as the Message Security between the Client Tier and Service Tier.

The OWC provides the majority of the capabilities required for a modern user interface. OWC provides some higher-level Connectivity capabilities (e.g. connectivity to content management systems, WSRP). OWLS provides the base Connectivity (e.g. JDBC, JCA) to connect to the Resource Tier.

While OWC provides Collaboration capabilities, both OWCIC and OWCRTC provide enhanced Collaboration capabilities to the architecture. The Search capability is provided by OSES and the MetaData Repository capability is provided by OMDS.

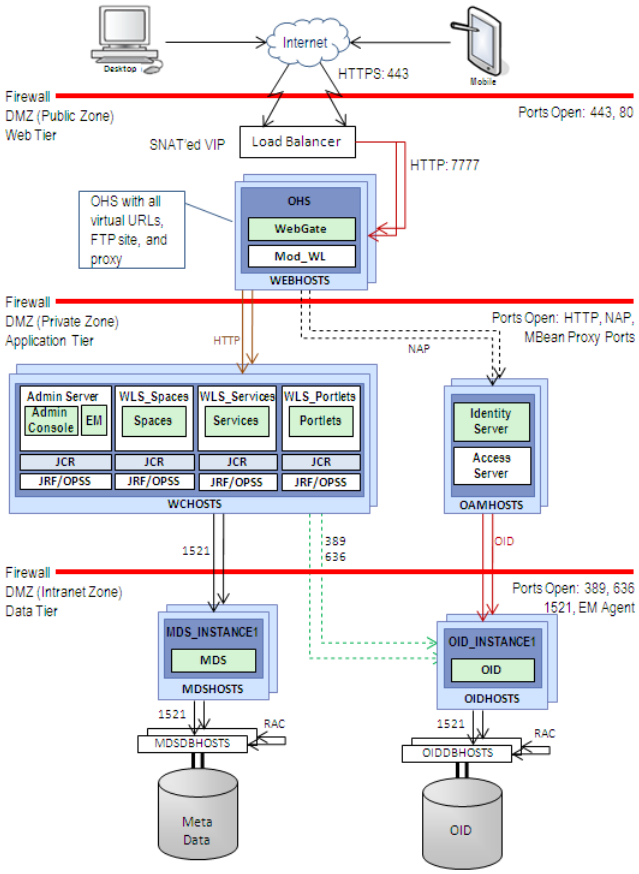
OAM provides the Access Management and Identity Propogation capabilities to the architecture. OID provide the Credential Management capability. Although not illustrated in Figure 5–2, it should also be noted that OPSS is incorporated into OWLS which ties the entire Service Tier into the enterprise security architecture. Please refer to the ***ORA Security*** document for a complete set of Oracle security products.

Oracle JDeveloper provides for the architecture. While Oracle JDeveloper is a powerful development tool, there are other Oracle products that are important for an enterprise software engineering environment. Please refer to the ***ORA Software Engineering*** document for a description of these products.

5.4 Deployment View

The sample deployment shown in Figure 5–3 illustrates the positioning of infrastructure elements across four separate network zones.

Figure 5–3 Example Product Deployment



The client devices are in the public network zone i.e. on the public internet. OHS is deployed in the Web Tier behind a firewall open only to HTTP and HTTPS network traffic.

The WebCenter products are deployed in the Application Tier network zone separated from the Web Tier by another firewall. The OAM product is also deployed to the Application Tier and is accessed by OHS to provide single sign-on capability to end users.

The OMDS products as well as the identity management products are deployed in the Data Tier separated from the Application Tier by another firewall.

OID is shown in this deployment diagram since the WebCenter products are most commonly deployed with access to an LDAP directory as the source for user profile information (e.g. organization, roles) and security credentials. This allows the security and personalization with the user interface to be based on existing user profile information.

6

Summary

Providing a user-centric, feature-rich, intuitive interface to end users requires a flexible, extensible architecture which incorporates all the necessary capabilities allowing the interface to be tailored to the specific needs and preferences of the individual end user. This document describes an architecture appropriate for delivering a best-in-class user interaction experience based on industry standards and architecture principles.

Oracle offers a complete set of products that provide the foundation for a modern user interface. This document includes a product mapping view which illustrates how the Oracle products can be deployed to realize the capabilities required for a modern user interface.

This document covered only the core concepts and key capabilities of a modern user interface. Complementary technologies and products are covered separately and in greater depth in the other Oracle Reference Architecture documents.

A

Further Reading

The ***IT Strategies From Oracle*** series contains a number of documents that offer insight and guidance on many aspects of technology. In particular, the following documents pertaining to ***ORA User Interaction*** may be of interest:

ORA Application Infrastructure Foundation - Underpinning enterprise applications and infrastructure is a computing platform that provides reliability, availability, scalability, and performance qualities for enterprise-class computing. This document describes these concepts and capabilities and defines an appropriate solutions platform.

ORA Software Engineering - Rapidly developing robust business solutions requires a feature-rich and sophisticated engineering platform that allows one to model, design, develop, test, and deploy business solutions. This document defines the core capabilities and best practices required for effective solutions development built and deployed on Oracle products.

ORA Management and Monitoring - This document describes important management and monitoring capabilities and a reference architecture to address the needs for the modern IT environment.

ORA Security - This document describes important aspects of security including identity, role, and entitlement management, authentication, authorization, and auditing (AAA), and transport, message, and data security required to secure the modern IT environment.

ORA Service-Oriented Integration - This document examines the most popular and widely used forms of integration, putting them into perspective with current trends made possible by SOA standards and technologies. It offers guidance on how to integrate systems in the Oracle environment, bringing together modern techniques and legacy assets.

In addition, the following materials and sources of information relevant to ***ORA User Interaction*** may be useful.

Oracle WebCenter Suite 11g: The Modern User Experience Platform for the Enterprise and the Web - an Oracle White Paper.

Oracle WebCenter 11g Release 1: New Features Overview - an Oracle White Paper.

Accelerating Collaboration with Oracle WebCenter Intelligent Collaboration - an Oracle White Paper.

Oracle Secure Enterprise Search - an Oracle White Paper.

Web 2.0 Architectures, by James Governor, Dion Hinchcliffe, and Duane Nickull, 2009, O'Rielly Media, Inc. ISBN 978-0-596-51443-3.