Oracle® Reference Architecture

Software Engineering

Release 3.0

E16772-03

September 2010

ORACLE®

ORA Software Engineering, Release 3.0

E16772-03

Primary Author: Anbu Krishnaswamy

Contributing Authors: Sharon Fay, Shaun O'Brien, Stephen G. Bennett, Dave Chappelle, Bob Hensle, Mark Wilkins, Cliff Booth, Jeff McDaniel

Contributor: Joe Fernandes

Contents

3 Engineering Logical View

List of Figures

Send Us Your Comments

ORA Software Engineering, Release 3.0

E16772-03

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

Developing robust business solutions requires a feature-rich and sophisticated engineering platform that allows one to model, design, develop, test and deploy them. The engineering platform should provide the capabilities, tools and frameworks for developing solutions rapidly.

Innovations in business and technology result in organizations struggling to find skills to develop solutions using new tools and approaches. In order to overcome this challenge, organizations need a unified and integrated engineering environment where solutions are developed end-to-end rapidly and reliably.

Most organizations take an isolated approach to engineering when developing solutions and shared enterprise assets. The assets developed are managed locally without an enterprise perspective. Even shared services groups manage their assets in their own local repository, leading to redundancy and sprawls. One of the core principles of ORA Engineering is integrated asset management. It emphasizes the need for managing the asset metadata in a central repository so that the assets are not only properly tracked and reused but also a holistic functional view of the enterprise is built over time with corresponding relationships and dependencies. This type of integrated asset management also helps implement effective governance.

Document Purpose and Scope

ORA Engineering defines the core capabilities and best practices required for effective engineering of solutions built and deployed on the Oracle Fusion platform.

The focus of this document is the core elements of the ORA Engineering that cover the basic design, development and quality management aspects. This document does not cover the unique details of any specific technology. For example, this document does not cover the modeling and design of business processes. Business Process Analysis (BPA) is covered in the BPM Enterprise Technology Strategy (ETS).

This document describes the architecture aspects of ORA Engineering and is not intended to address the engineering or development methodology. It focuses on the architectural components of the design-time environment that enables the design and development of enterprise solutions.

Business Process Analysis and Modeling is out of scope for this document as the audience for Engineering and Business modeling are different. This document covers business process modeling from the perspective of the technical users. The *BPM Enterprise Technology Strategy* covers the Business Process Analysis and modeling in detail.

This document covers asset management as an engineering discipline but does not cover operational management and monitoring. Management and monitoring is covered in the *ORA Monitoring and Management* document.

This document covers the architecture concepts pertaining to the following Oracle products/tools.

- JDeveloper and ADF
- Oracle Enterprise Pack for Eclipse (OEPE)
- Oracle BPM Studio
- Oracle BPA Suite
- Oracle SQL Developer Data Modeler
- Oracle Enterprise Repository (OER)
- Oracle Service Registry (OSR)
- Oracle Application Express (APEX)
- Oracle Application Testing Suite (ATS)
- Oracle Real Application Testing

Audience

This document is intended for enterprise architects, application architects, project managers and developers. The material is designed for a technical audience that is interested in learning about the intricacies of engineering infrastructure and how the infrastructure can be leveraged to satisfy the solution development needs. This document also provides useful insights for those that architect shared services and manage shared services competency centers.

Document Structure

This document is organized into chapters that introduce ORA Engineering concepts, standards, and architecture views.

Chapter 1, "Introduction" - provides an introduction to ORA Engineering, asset-centric engineering and related standards.

Chapter 2, "Engineering Conceptual View" - provides a conceptual model for ORA Engineering and describes the capabilities required for an engineering infrastructure.

Chapter 3, "Engineering Logical View" - The logical view shows the logical components of the engineering environment and show how they are connected to each other.

Chapter 4, "Deployment View" - The deployment view describes packaging and deployment related aspects of ORA Engineering.

Chapter 5, "Product Mapping View" - This section describes how Oracle products fit on to the logical model to realize the engineering infrastructure.

Chapter 6, "ORA Engineering Best Practices" - discusses some of the basic best practices with respect to ORA Engineering.

Chapter 7, "Summary" - summarizes the ORA Engineering document.

Chapter , "Glossary" - provides a list of terms specific to this document. *ORA Master Glossary* provides the terms that are common across the documents.

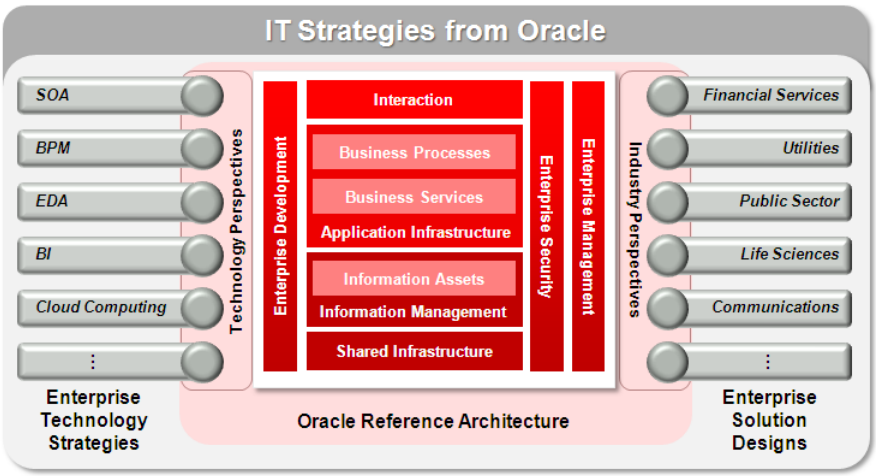
Chapter A, "Further Reading" - provides a list of documents and reading resources for further information and reference.

How to Use This Document

This document is designed to be read from beginning to end. Those that are already familiar with engineering concepts and standards may wish to skip the initial chapters and proceed with the reference architecture definition that begins with Chapter 2, "Engineering Conceptual View".

Related Documents

IT Strategies from Oracle (ITSO) is a series of documentation and supporting collateral 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 success and adoption. 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.

ORA Engineering is one of the series of documents that comprise Oracle Reference Architecture. ORA Engineering describes important aspects of the Enterprise Development layer including asset-centric engineering and architectural components of the design-time environment that enable the design and development of enterprise solutions.

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 ***ORA 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.

1 Introduction

Most of the traditional engineering approaches are based on silo-ed organizational models where the software delivery teams are aligned with individual lines of businesses. Projects funded by a business unit tend to confine their scope narrowly to the requirements of that particular business unit. This approach inhibits reuse and diminishes business value. Even within a single project, information and assets are sometimes not shared across various phases. This may lead to misalignment of the requirements and implementations.

1.1 Asset-centric Engineering

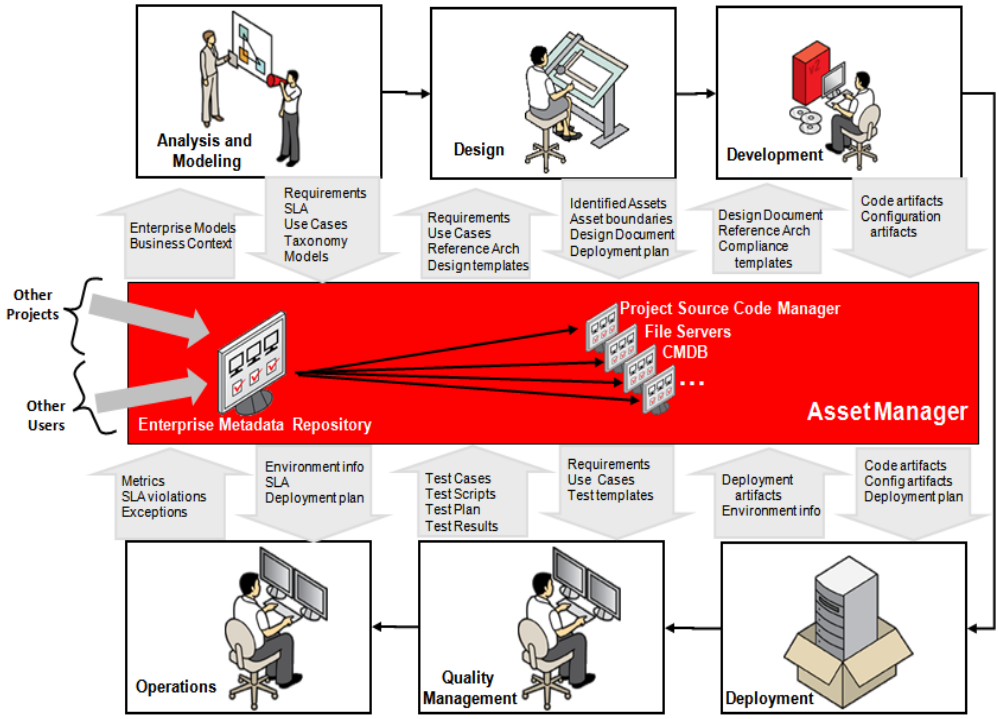
The underlying core principle of ORA Engineering is asset sharing and enterprise development through an integrated asset management approach. Most organizations use a Software Configuration Management (SCM) or Version Control System (**VCS**) for managing the code and configuration assets. These tools are great for managing the versioning of assets produced but they don't maintain the metadata of the assets. Without metadata assets are not organized in context and it is hard to discover them. ORA recommends an asset-centric engineering process as shown in Figure 1–1, where an Asset Manager is used to address the challenges posed by the traditional approaches. The Asset Manager is typically an enterprise-scoped Metadata Repository working in concert with SCMs and other types of asset repositories.

The traditional approach to engineering that does not encompass an integrated asset management discipline has several challenges including the following.

- As assets are managed at the project level, the development is basically silo-ed. Projects are run in isolation without ever taking an enterprise perspective or contributing to the enterprise asset base.
- Traditional approaches do not realize the full benefit of sharing and reuse. They depend on manual approaches to identifying and taking advantage of reuse opportunities. Sharing and reuse start from requirements analysis and most traditional processes do not provide a framework to facilitate it.
- When development efforts are not coordinated, it leads to duplication and asset sprawls. It is not uncommon to see duplication of functionality and business capabilities in organizations that are heavily fragmented.
- Enforcement of governance is honor-based and largely uses manual processes in a traditional model.
- Isolated development and localized maintenance of assets lead to lack of visibility into the enterprise asset base.

- In order to realize business value, assets developed must support the business goals. Traditional approaches lack the ability to trace the line of sight between the assets and business goals.
- Measuring how the assets perform and linking those measurements back to the goals and objectives of the assets is a critical factor in continuously improving business value. Failing to gather and correlate these metrics results in lack of insight and diminished value.

Figure 1–1 Asset-centric Engineering



With ORA's asset-centric engineering approach, the Asset Manager forms the logical component that links various engineering activities. It facilitates it by providing a central point for sharing and collaboration.

Since the Metadata Repository is enterprise-scoped, the project requirements can be refined into enterprise requirements allowing identification of reuse opportunities across projects. As more projects start using the Metadata Repository, the quality and completeness of the information it manages improves and it caters to other audience such as business users and portfolio managers.

Asset-centric engineering provides several other benefits including:

- **Enterprise Visibility**: Provides role-based visibility into all assets, regardless of the source for assets like business processes, SOA Services, applications, components, models, frameworks, and policies. Visibility into assets under development minimizes redundancy and promotes collaboration and reuse.
- **End-to-End Traceability**: Allows one to display and navigate asset-to-asset and asset-to-project relationships and interdependencies to simplify impact analysis. This ensures business alignment by enabling users to organize and link various assets to associated business processes and requirements. For example, the link between business processes, business process models, process execution, service

orchestration and SOA Service components are maintained so that they can be traced from one another.

- **Governance**: Governance requires a structured approach to defining standards, principles and policies and enforcing compliance to them through active advisory and monitoring. The Asset Manager helps implement it by facilitating workflows, standards, certification, policies, compliance, and SLAs. It allows best practices, templates and standards to be archived, prescribed and monitored.
- **Asset Lifecycle Management**: Supports the entire asset lifecycle with governance controls and automation that reach from the planning, design, and development of assets to their deployment and consumption in production. In addition, it enables impact analysis, planning and prioritization to manage the lifecycle of assets and to retire redundant assets.
- **Architecture alignment:** shows how requirements and logical models propagate to physical implementations and enables the identification of exceptions and violations.
- **Analytics for Measuring Value**: The Asset Manager is in a unique position to track the use and business value of the assets and produce valuable reports to target a range of stakeholders with key metrics, including business value, ROI, predicted versus actual reported savings from reuse, asset usage, and compliance.

Figure 1–1 shows various activities in a typical engineering process and how an Asset Manager helps the flow of information and assets across the lifecycle of the assets. More importantly, it gives visibility into the rest of the enterprise to promote reuse and sharing. The physical asset may lie in the SCM, other repositories or in the runtime environment but the Metadata Repository maintains the metadata along with a pointer to the physical asset in the SCM.

The responsibilities of the repository and SCM are very distinct. The SCM manages the micro-versioning of individual constituents of major assets and the Metadata Repository manages the macro-versioning of composite assets. For example, the Metadata Repository plays a key role in maintaining the versioning of a SOA Service asset as a whole. It shows the current versions of the SOA Services and their relationships with the older versions and other SOA Services. In contrast, the SCM manages the versioning of the components that constitute the SOA Service. The contract documents, interfaces (e.g. WSDL), XML schemas, code and configuration artifacts are some of the examples of the assets versioned by the SCM.

Figure 1–1 shows a sample process that might appear somewhat similar to the Unified Process (UP) but the concept of asset-centric engineering is applicable to any type of development methodology including Waterfall, Unified Process, **Extreme Programming (XP)**, **Scrum**, etc. Figure 1–1 illustrates the following activities.

- **Analysis and Modeling**: Project requirements and the business context are captured and the requirement documents are produced. Use cases are created and the requirements are classified against the enterprise functional model. Other artifacts such as solution architecture models and documents are also created in this phase.
- **Design**: The solution is designed based on the project requirements and enterprise standards available in the Asset Manager. The produced deliverables including asset definitions, design documents and deployment plan are fed back into the Asset Manager along with their inter-dependencies.
- **Development**: The development team consumes the project resources and generates the code and configuration artifacts that are versioned in the SCM along with the metadata defined in the Metadata Repository. In addition, the assets can

be harvested into the Enterprise Repository so that there is visibility into the deployed endpoint.

- **Deployment**: Once the basic assets are in place, the runtime artifacts and environment plan are built. The details of the environment (hardware, network, topology, platform etc.) and the details of what are going to be deployed on these environments are captured in the Asset Manager as well.
- **Quality Management**: Quality Management activities may run parallel to the design and development activities. Test case development is driven by the requirements or contracts. In a Test Driven Development (TDD) approach, the test cases and test scripts are first developed even before the code is developed. Regardless of the methodology, the asset-centric engineering approach streamlines the flow of assets across the activities through the Metadata Repository and other asset management systems.
- **Operations**: When the project goes live, the operations team uses the deployment plan, environment information and other related assets for OA&M (Operations, Administration and Management). The operational metrics and other statistics collected are fed back into the Metadata Repository for closed loop governance.

1.2 Standards

A number of standards are relevant and applicable for ORA Engineering. The objective of this document is not to cover all the engineering related standards but just to highlight the most prominent ones related to the Oracle environment. Most of these standards are covered in detail in the other ORA documents.

1.2.1 Asset Management Standards

The standards related to asset management are

- OMG Reusable Asset Specification (RAS)
- OMG Meta Object Facility (MOF)
- OMG XML Metadata Interchange (XMI)

1.2.1.1 OMG Reusable Asset Specification (RAS)

This specification is a set of guidelines and recommendations about the structure, content, and descriptions of reusable software assets. It identifies some categories, or rather types or profiles and provides general guidelines on these profiles. RAS addresses the engineering elements of reuse. It attempts to reduce the friction associated with reuse transactions through consistent, standard packaging.

1.2.1.2 OMG Meta Object Facility (MOF)

MOF is an extensible model-driven integration framework for defining, manipulating, and integrating metadata and data in a platform independent manner. MOF-based standards are used for integrating tools, applications and data.

1.2.1.3 OMG XML Metadata Interchange (XMI)

XMI is a model-driven XML Integration framework for defining, interchanging, manipulating and integrating XML data and objects. XMI-based standards are used for integrating tools, repositories, applications and data warehouses. XMI provides rules by which a schema can be generated for any valid XMI-transmissible Meta Object Facility-based meta-model.

1.2.2 Modeling Standards

There are several software engineering modeling standards in existence. This section lists some of the prominent ones.

1.2.2.1 Business Process Modeling Notation (BPMN)

BPMN is a standard that defines process modeling and graphical representation for capturing business processes. The objective of BPMN is to support business process management for both technical users and business users by providing a notation that is intuitive to business users yet able to represent complex process semantics. More details about BPMN can be found in the *ORA BPM Foundation* document.

1.2.2.2 Business Process Execution Language (BPEL)

Business Process Execution Language (BPEL), short for Web Services Business Process Execution Language (WS-BPEL) is an OASIS standard executable language for specifying interactions with Web Services. Processes in Business Process Execution Language export and import information by using Web Service interfaces exclusively. BPEL is covered in detail in the *ORA BPM Foundation* document.

1.2.2.3 XML Process Definition Language (XPDL)

XPDL is a standard designed to exchange the process definition, both the graphics and the semantics of a workflow business process. XPDL contains elements to hold graphical information, such as the X and Y position of the nodes, as well as executable aspects which would be used to run a process. This distinguishes XPDL from BPEL which focuses exclusively on the executable aspects of the process. Please refer to the *ORA BPM Foundation* document for further discussion on XPDL.

1.2.2.4 Unified Modeling Language (UML)

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed by the Object Management Group. UML includes a set of graphical notation techniques to create visual models of software-intensive systems.

1.2.2.5 OMG Systems Modeling Language (SysML)

This specification defines a general-purpose modeling language for systems engineering applications. SysML supports the specification, analysis, design, and verification and validation of a broad range of complex systems. These systems may include hardware, software, information, processes, personnel, and facilities.

1.2.2.6 OMG SOA Modeling Language (SoaML)

This specification defines a meta-model and a UML profile for the specification and design of SOA Services within a service-oriented architecture.

ORA defines a meta-model for SOA Services. More information about it can be found in the *ORA SOA Foundation* document.

1.2.3 Implementation Standards

This section describes the standards related to implementation.

1.2.3.1 Java Enterprise Edition - JEE

Java Platform, Enterprise Edition or Java EE is a widely used platform for server programming in the Java programming language. The Java platform (Enterprise Edition) adds libraries, which provide functionality to deploy fault-tolerant, distributed, multi-tier Java software, based largely on modular components running on an application server.

Java EE is considered informally to be a standard since providers must agree to certain conformance requirements in order to declare their products as Java EE compliant. Java EE includes several API specifications, such as JDBC, RMI, JMS, Web Services, XML, etc., and defines how to coordinate them.

Java EE also features some specifications unique to Java EE for components. These include Enterprise Java Beans (EJB), Connectors, Servlets, Portlets (following the Java Portlet Specification), Java Server Pages and several Web Service technologies. This allows developers to create enterprise applications that are portable and scalable, and that integrate with legacy technologies. A Java EE application server can handle transactions, security, scalability, concurrency, and management of the components that are deployed to it, in order to enable developers to concentrate more on the business logic of the components rather than on infrastructure and integration tasks.

1.2.3.2 Java Community Process - Java Specification Requests (JSR)

The Java Community Process or JCP is a formalized process that allows interested parties to get involved in the definition of future versions and features of the Java platform.

The JCP involves the use of Java Specification Requests (JSRs) - the formal documents that describe proposed specifications and technologies for adding to the Java platform. A final JSR provides a reference implementation that is a free implementation of the technology in source code form and a Technology Compatibility Kit to verify the API specification.

Currently there are over 300 JSRs in the process. A link to the JCP page is provided in the references section (Appendix A) of this document.

1.2.3.3 Web Services Standards

The core Web Services standards are

- Web Services Description Language (WSDL),
- Universal Discovery and Description Language (UDDI) and
- Simple Object Access Protocol (SOAP).

A number of other Web Services related standards and extensions are available to cover various technology areas and use cases. These include

- WS-* Specifications
- Web Services Remote Portlets

These standards are discussed in the *ORA SOA Foundation* document.

1.2.3.4 Other standards

Some of the other notable engineering related standards are:

1.2.3.5 Service Component Architecture (SCA)

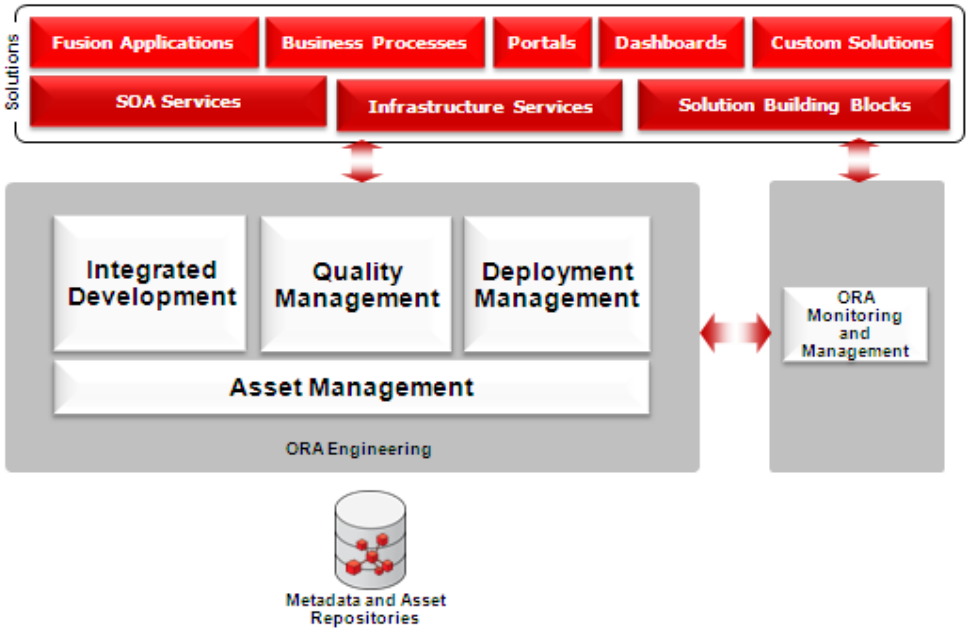
Service Component Architecture (SCA) provides a programming model for building applications and systems based on a Service Oriented Architecture. It is based on the idea that business function is provided as a series of SOA Services, which are assembled together to create solutions that serve a particular business need. These composite applications can contain both new SOA Services created specifically for the application and also business functions from existing systems and applications, reused as part of the composition. SCA provides a model both for the composition of SOA Services and for the creation of service components, including the reuse of existing application functions within SCA compositions.

2

Engineering Conceptual View

This chapter of the reference architecture introduces the basic capabilities of ORA Engineering infrastructure. It builds on the concepts and standards described in the previous chapters and provides context for the next chapter which presents a logical view.

Figure 2–1 ORA Engineering Categories



The high-level logical categories are shown in Figure 2–1. The broad categories that define ORA Engineering are:

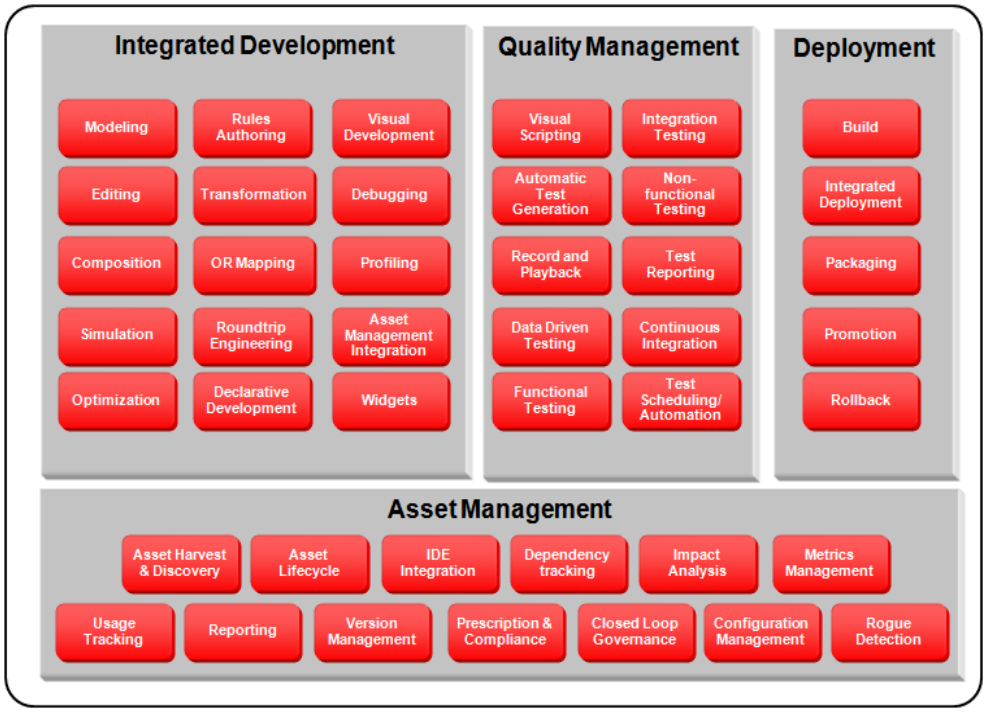
- Integrated development
  - This covers a wide range of engineering capabilities required to model, design and build solutions. These capabilities go beyond simple editing and include advanced capabilities to support round-trip engineering, integrated testing, deployment, and asset management.
- Quality Management
  - Quality Management capabilities ensure that the developed solution meets the enterprise standards and pass the exit criteria. Quality Management covers testing, defect management, and continuous integration.

- Deployment Management
  - Deployment Management deals with building, packaging, migration, and deployment of assets.
- Asset Management
  - Asset Management deals with the visibility, management and governance of assets and asset metadata. It covers the capabilities required to effectively manage enterprise assets.

These categories of capabilities support the development of the business solutions, business services, infrastructure services, and solution building blocks. Engineering infrastructure plays a key role in influencing business competitive advantage by building business capabilities faster and ensuring high quality through excellence in engineering process. ORA Engineering also has touch points with other components of ORA such as *ORA Monitoring and Management* (M&M). There is a two-way exchange between these subsystems. Engineering infrastructure uses the management capabilities for deployment of runtime and detecting rogue services and the M&M infrastructure provides the overall and individual asset operational metrics to the design-time decision components for closed loop governance.

Figure 2–2 expands the previous diagram by adding the capabilities to the logical categories. The details of these are discussed in the subsequent sections.

Figure 2–2 ORA Engineering Conceptual View



2.1 Integrated Development

The focus of integrated development is to improve developer productivity by streamlining the development tasks and by reducing context switching between tools. This section explores the integrated development capabilities.

2.1.1 Modeling

Most effective engineering processes begin with modeling and continue to leverage modeling throughout the development lifecycle. Graphical models narrow the communication gap between various IT and business stakeholders. Modeling capabilities include

- Graphical tools and notations with import/export capabilities
- Support for industry standard notations
- Simulation and what-if analyses
- Round-trip modeling

2.1.2 Editing

Writing and modifying code is one of the most basic features in engineering. Although not architecturally significant, these features play a key role in improving developer productivity. Following is a list of key code editing features:

- Code completion
- Syntax highlighting
- Block highlighting
- Multi-way editing where the source code and the other views like design canvas are kept in synchronization.
- Code refactoring allows automatic restructuring of the code to optimize the design and execution.

Most software platforms also support some form of scripting. Scripting is generally simpler to develop compared to a full featured programming language. Java Script, Groovy, Python, Jython and Ruby are some of the popular scripting languages. Auto completion, syntax highlighting and script testing harness are some of the desirable capabilities for scripting.

In asset-centric engineering, developers have access to proven assets. They can use an existing asset, modify the source, and submit their revisions, as in the Open Source community.

2.1.3 Composition

Composition allows existing assets to be quickly combined to create value-added assets. Composition is an important architectural construct that promotes faster time-to-market through sharing, reuse, and improved agility. Composition may be in many forms including the following

- **Functional composition**: enables complex functionality to be built by combining simple functional blocks.
- **Service composition**: Composite Services are built by orchestrating atomic SOA Services.
- **Object composition**: is a key principle of Object Oriented Analysis and Design and allows simple objects and data types to be combined into types that are more complex. Compositions are a critical building block of many basic data structures such as linked list and binary tree.

In an asset-centric engineering model, composition is done with pre-engineered assets. These have been tested and qualified, which ensures the quality of the composition, reduces risk, and speeds delivery.

2.1.4 Simulation

Simulation is a very powerful capability in an engineering environment. It allows various real-life scenarios to be tried and tested in the system by changing the key parameters. With simulation, what-if scenarios can also be tested and the results provide essential input to the optimization process.

2.1.5 Optimization

Most models are not optimal the first time they are created. They need to go through a few rounds of optimizations to increase efficiency and to maximize the business benefits. Optimization is often done in concert with simulation as the outputs and results of simulation lead to the refinement of that model. The goal of optimization is to adjust the asset parameters to maximize throughput and efficiency, and to minimize the cost.

For example, business process models are used to simulate business processes that reveal areas of the business process (Business Process Reengineering) and process implementation (technical aspects) that can be optimized.

2.1.6 Rules Authoring

An essential part of engineering is to author rules and policies. The development infrastructure must provide the ability to easily model and create them. The ability to test and troubleshoot the content is an added feature. Most authoring use cases also require support for crafting regular expressions using XML query languages such as XSLT or XQuery.

2.1.7 Transformation

It is quite common to encounter use cases that require transformation of information from one format to another, especially in the area of enterprise integration. Source systems and target systems may use very different representations of data and in some cases, a canonical data model might be used as a common intermediate format. In some cases, the transformation is a simple field-to-field mapping whereas in other cases it is a complex manipulation and conversion of data. It should be possible to visually map the source and target representations with the ability to enrich the elements to support both simple and complex data transformations.

2.1.8 Object-Relational Mapping (ORM)

Object Relational Mapping (ORM or O/R Mapping) allows the entities from a relational database to be fetched and manipulated in object oriented programming languages.

Almost all enterprise software applications involve accessing the database in some form or the other, which makes it critical to support ORM. ORM often reduces the amount of code needed to be written and reduces the chance of manual errors, making the software more robust.

Most tools create a one-to-one transformation from the data model to object model. The data model should be normalized to ensure that the granularity of the object model reflects the business entities appropriately.

2.1.9 Round-trip engineering

Forward engineering is the ability to derive artifacts from a previous lifecycle activity. It allows code artifacts to be generated from models and ensures that the implementation code is in alignment with the models and requirements.

Reverse engineering is a useful feature for discovering and refactoring existing code. It is the ability to generate the artifacts backwards in the lifecycle. In practice, two main types of reverse engineering emerge.

- In the first case, source code is already available for the software, but higher-level models and documentation are produced.
- In the second case, there is no source code available for the software, and any efforts towards discovering one possible source code for the software are regarded as reverse engineering. In some cases, binary code may need to be ported back to source code and in others, source code may need to be reconstructed by studying the application behavior.

Round-trip engineering is the ability to effectively perform both forward and reverse engineering to seamlessly transform the lifecycle assets. Forward engineering is very useful the first time that code is generated from a model. It saves much of the mundane work of keying in classes, attributes, and methods. Reverse engineering is very useful both to transform code into a model when no model previously existed, as well as to resynchronize a model with the code at the end of a change.

During an iterative development cycle, once a model has been updated as part of an iteration, another round of forward engineering should allow code to be refreshed with any new classes, methods or attributes that have been added to the model. Source code generally contains much more than the model and tools must be very adept at reconstructing the source code that existed prior to the new round of forward engineering. At minimum, the modeling tool should successfully support forward engineering the first time and reverse engineering throughout the process.

2.1.10 Declarative development

Development tools have evolved from primitive editors to user-friendly, intelligent and sophisticated development platforms. In order to avoid repetitious and tedious coding, most editors support declarative development where annotations drive the logic flow and control.

2.1.11 Visual development

Visual development significantly improves developer productivity. A combination of visual editors, property inspectors, structure panes, and editing dialogs simplify or eliminate tedious coding. These visual features provide a simpler way to define the components and the code is always accessible for direct manipulation as well. Changes could be made either visually or directly in the code and should be reflected in both views simultaneously.

2.1.12 Debugging

Sound design and best practice based development improve the quality of code developed. There is no replacement for sound design but integrated debugging acts as the first level of defense against bugs in the code. There are several debugging related capabilities such as:

- Break points

- Step-through code
- Snapshot view and watch view of variables
- Dynamic update of instance variables

2.1.13 Profiling

Profiling is a form of dynamic program analysis that studies a program's behavior using information gathered as the program executes. The usual purpose of this analysis is to determine which sections of a program to optimize either by increasing its overall speed or by decreasing its memory requirement or sometimes both. Code instrumentation improves the accuracy of the analysis by providing additional statistics.

2.1.14 Asset management integration

A successful asset-centric engineering approach makes effective use of asset management. Interfacing with the asset management should not be an isolated task; rather, it should be an integrated task within the engineering activities. This requires that the design and development tools be seamlessly integrated with the asset management infrastructure.

Assets are generated and consumed at every step of the engineering process. For example, the modeling step produces an archivable model and the design step produces design documents. If the modeling and design tools are integrated with the asset management infrastructure, these deliverables can be transparently submitted and organized in the asset management system.

Integration with the Metadata Repository allows the metadata to be defined and archived on the fly, as the assets are being developed. It allows assets to be searched, discovered and downloaded for consumption. It also enables the repository to gather vital statistics about the usage and compliance status, leading to effective governance.

Integration of the development environment with the SCM allows faster, frequent, and effective version management of the assets. This reduces the chances of conflicts and manual errors.

2.1.15 Widgets

Widgets are out-of-the-box utilities that improve developer productivity. Widgets must be interoperable and easily integratable. Calendar, charts, and other data visualization components are examples of useful out-of-the-box widgets. The architecture should allow the widgets to be plugged into the code.

2.2 Quality Management

Quality management is a critical part of an engineering process. In order to ensure that 100% of the code is tested and quality assured, the testing process should be automated and checkpoints should be introduced to assess the quality of the code. Testing frameworks and tools should be chosen prudently and used to ensure that the functional and non-functional criteria have been thoroughly met.

Concepts like Test Driven Development (TDD) emphasize the need for quality solutions that meet the end user requirements by developing the end user test cases before developing the code. Testing should be driven by requirements or service contracts rather than design or implementation. Test cases should be derived from the

requirements and use cases. Test readiness can and should happen in parallel to design and development.

The following are some of the key capabilities with respect to quality management.

2.2.1 Visual Scripting

Visual scripts offer the fastest and easiest way to define tests for Web applications. Visual scripts capture the interaction of the user with the application under test. These scripts serve as the baseline and can be replayed against new versions of the application to detect functional, regression, scalability, and post-deployment defects.

Visual scripts offer the power and flexibility to address all the testing challenges. A subset of the Visual scripts that depict different profiles of usages of the application can be re-used for load testing and scalability testing. All of the functional testing scripts that depict correct behavior of the application can be re-used after deployment to monitor the application any time.

2.2.2 Automatic test generation

The ability to automatically generate the tests from finished code ensures that all parts of the solutions are tested from the perspective of the user. It also reduces the laborious task of manual test generation. Although this is a useful capability, the tests generated automatically may not be sufficient to cover all scenarios. If the test scripts are automatically generated based on the interfaces, then manual tests must ensure that the interfaces are tested properly.

2.2.3 Record and playback

To avoid laborious manual testing, the testing framework should allow record and playback of tests. With this capability, tests can be recorded as they are run manually. These scripts can be played back unlimited number of times to conduct future tests.

2.2.4 Data driven testing

One of the best ways to perform functional testing is through data-driven testing, in which a databank is created to cover the various functional use cases and is used to drive the testing. This requires the ability to iterate through a list of data sets in the databank, substitute them for the input values, and run the tests.

2.2.5 Functional testing

Functional testing is the most basic but critical capability that ensures that the functionality delivered to the end user fulfills the business requirements in its entirety. Manual functional testing might become tedious especially if the requirements are complex and data intensive.

Automating the functional testing requires the ability to capture the user inputs either in the form of data inputs or as a sequence of activities that need to be performed or a combination of both. The next step is using the data to run the tests and capture the output of the transaction. This output should then be compared to the expected output and reports indicating the test results along with other related metrics should be generated.

The infrastructure should not only provide the ability to run the functional tests with different data but it should also allow the data inputs and expected outputs be recorded to be rerun at a later point of time.

2.2.6 Integration testing

Integrated testing is the ability to test the code modules together as they are being developed. In order to support such a feature the tool should also have integration with the container management and deployment. This would allow it to deploy the code directly from the IDE to the container and run the necessary tests. Integrated testing should also support automatic generation of input messages and examination of request traces for troubleshooting purposes.

2.2.7 Non-Functional testing

Non-functional testing ensures that the assets meet the reliability, availability, scalability and performance requirements of the business. Performing non-functional testing requires the ability to generate peak load that closely simulates the real world and validate the results against the Service Level Agreements (SLA).

Non-functional requirements are best tested through load testing. Load testing is a special case of non-functional testing that focuses on scalability and the effects of load on the system. Following essential capabilities are needed to perform non-functional and load testing.

- Load generation
- Simulation of think time and delay times
- Load ramp up and ramp down
- Multi-threaded execution
- Result capture and reporting

Load testing is also a means of testing reliability. The failure rate is an indication of reliability. Availability measures the uptime of the underlying hardware and software infrastructure. The tests should measure how well the system handles unplanned downtime through failover and recovery.

2.2.8 Test reporting

Test results must be reported appropriately with failures and exceptions highlighted to assist the troubleshooting process. Test reports may include any of the following types.

- Real-time dashboards that stream the current status of tests
- Custom reports that provide a view based on custom criteria.
- Graphical views to display user friendly snapshots and trend analysis

2.2.9 Continuous integration

The developer code should be integrated and tested early and often. The latest and greatest assets from the repository should be periodically pulled and the test cases should be run. Reports should be generated with failures and exceptions highlighted. This functionality benefits from test scheduling and test dashboard capabilities described later in this section.

2.2.10 Test scheduling and automation

Test job scheduling enables unattended and repeat testing. A test job scheduler may allow flexible and sophisticated calendar based scheduling. The scheduled tests should be run automatically and reports should be generated for later examination.

The infrastructure should also include a notification mechanism to notify the users of test status or exceptions.

2.3 Deployment Management

Deployment management deals with preparing the assets to a form suitable for provisioning.

2.3.1 Build

The code assets are compiled and organized using the build process. In a development environment, build is generally integrated into the IDE and can be executed from within. It should also be possible to build the code outside the development environment using build tools and scripts so that the administrators and the deployers can pull the latest source and build them independently for deployment.

This capability must support standard and custom build activities. Custom build activities allow the administrators to create custom scripts to build and test the assets.

2.3.2 Integrated deployment

Integrated deployment allows deployment and testing of assets as they are being developed in the development environment. This capability improves code quality and developer productivity. With integrated deployment, one can control the runtime container from the development environment and deploy the assets in an exploded form without explicitly packaging the code. Exploded deployment enables selective redeployment and it makes it easy to debug code.

2.3.3 Packaging

Packaging is the grouping and organizing of assets and asset components to flexibly deploy and administer the assets in a standards-based manner. Packaging should ensure that all non-functional criteria are met and there is room for extending the capacity without code, configuration, or packaging changes.

Packaging tools must support fetching, organizing, and archiving the assets in a flexible manner. Packaging wizards allow best practice based package structures to be created using a GUI based helper.

Packaging enables the following:

- Create, examine and update deployment descriptors
- Archive the binary and configuration assets
- Automatic creation of manifest files

Following are some of the considerations with respect to packaging:

- Packaging format: Assets may be packaged in an exploded format or archived format. Exploded format allows hot deployment and can load assets individually without having the need to redeploy the entire package.
- Classloader optimization: Assets must be packaged to optimize the way they are loaded in the runtime. Classloaders are components that are responsible for loading the classes.
- Static content optimization: Static content may be separated from the dynamic content so that it can be either cached or served independently to improve performance.

- Precompilation: Some of the assets may be precompiled to improve runtime performance.

2.3.4 Promotion

There are two primary means of asset promotion. The first method is the promotion of the asset archives through the build process. The second method is a dynamic promotion where the asset runtimes are directly transferred from one environment to another. Dynamic promotion or demotion of assets between environments is crucial for certain assets like SOA Services. This may require the application of a workflow process that includes a role-based approval to promote to the next environment. Sometimes it may be necessary to replace environment specific parameters in the assets. While some tools are capable of addressing this need, scripting support enhances the experience of the Operations and Administration personnel. See *ORA Management and Monitoring* for more details around configuration management.

The tools should also have the ability to maintain the integrity of the environment through all-or-nothing deployment of related assets. Since the Metadata Repository is the central source of all assets providing a snapshot view of the enterprise, the asset metadata should be updated with the appropriate environment information to indicate the current deployment status.

2.3.5 Rollback

When a build fails to meet the exit criteria to be moved to production, it is necessary to rollback to a last known good deployment configuration to preserve the consistency. A number of different mechanisms may be employed for performing a rollback. SCMs allow version labels to identify the assets that belong to a specific build and rollback can be performed by simply retracting to the previous version label. Some infrastructure tools support rollback to a previous configuration without the need for a rebuild.

2.4 Asset Management

In order to realize business value through governance and control, asset management should be made an integral part of the engineering process. The lack of well planned asset management will result in redundancy and asset sprawl.

This section describes the key asset management capabilities required for a successful enterprise implementation.

2.4.1 Asset Harvest and Discovery

A key requirement of asset management is the ability to harvest assets from various sources and discover assets in an easy and user friendly manner. An undiscoverable asset is a non-existent asset. Assets are harvested and associated with comprehensive metadata and are categorized with the appropriate taxonomy that characterizes them. This allows the assets to be discovered through metadata search or taxonomy navigation.

2.4.2 Asset Lifecycle

Assets go through a lifecycle of their own, from conceptualization to retirement. The status of the asset must be tracked and appropriate actions and checkpoints must be introduced in the lifecycle. Asset management infrastructure must support the tracking and management of an asset's lifecycle.

2.4.3 IDE integration

An integrated asset management would have a better rate of success with design-time governance. Integrated asset management not only allows harvesting and discovery of assets from the IDE but also enables assets to be prescribed and checked for compliance.

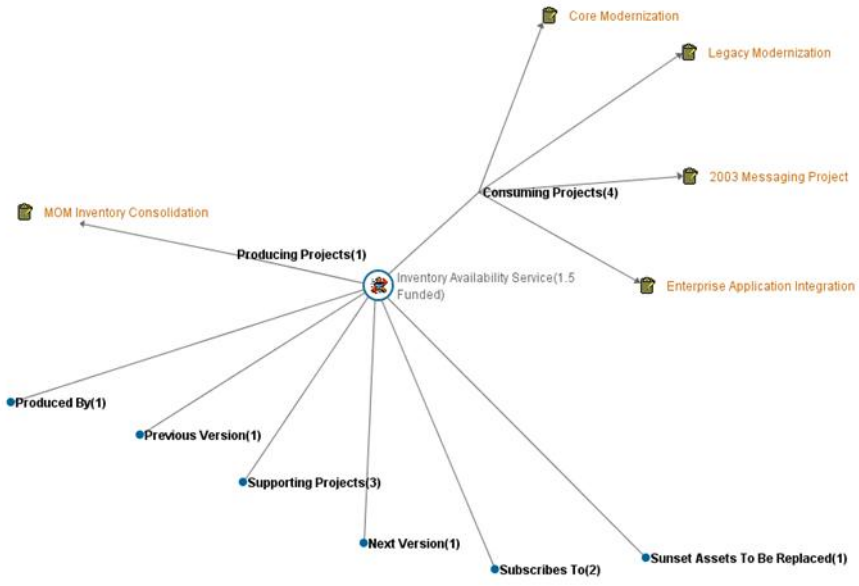
2.4.4 Dependency tracking

As the number of assets grow rapidly, the interdependencies become hard to track if not managed properly. Dependency tracking is beneficial as the number of assets increases, the relationships become more complex, and the need to revise or retire assets arises. Tracking the dependencies allows us to perform impact analysis when changes need to occur. This goes both ways - one may need to understand the dependencies an asset has, if the asset needs to be changed, moved, or virtualized; or one may need to understand what assets depend on a particular resource, if that resource needs to be modified, retired, or moved.

2.4.5 Impact analysis

A key part of asset management is the ability to assess the impact in development and operational environments as a result of a change to a given asset. Impact analysis helps with release planning and asset enhancement prioritization. Impact analysis is not to be confused with dependency tracking. Dependency tracking is the ability to track the relationships between assets and impact analysis is making use of those relationships to analyze the impact of a change.

Figure 2–3 Impact Analysis



For example, the dependency tree in Figure 2–3 shows that four other projects depend on the Inventory Availability Service that is being developed as part of the MOM Inventory Consolidation project. This service needs high priority to ensure on-schedule completion; otherwise four other projects will be negatively impacted.

2.4.6 Metrics management

Enterprise visibility and performance management depend on the ability to measure the business value added by individual assets. This requires metrics to be identified, defined and tracked for the assets. An asset management infrastructure provides a way of associating and tracking the metrics related to the assets.

2.4.7 Usage tracking

Tracking the usage of assets produces key statistics to assess the value of assets. Knowing which assets would have greater demand allows the portfolio managers to prioritize the development of assets and plan accordingly. Usage tracking comes in three forms:

- The registered intent to use an asset
- The interest in an asset resulting in a subscription to that asset
- The actual audit trail of consumers using the asset

The first two are design-time tracking mechanisms that can be achieved by surveying potential consumers. It is a great way of determining the value of the assets. The third is a run-time approach.

2.4.8 Reporting

The asset management system needs to be able to generate a variety of reports such as the following:

- Asset production, usage, ratings, value, status, rejections, deployments, and portfolio valuation reports
- Compliance reports
- User and security reports

2.4.9 Version Management

Assets may require changes based on a number of factors, such as code defects, functional updates, non-functional requirement changes, and resource modifications.

Versioning can be fairly simple for assets with only one consumer. In some cases, the consumer and provider may be updated simultaneously. This reduces (or eliminates) the need for concurrent versions.

Versioning becomes more difficult when multiple consumers use the assets. In these cases, every effort should be made to support multiple concurrent versions in production. This allows consumers to test and migrate to the latest version on their own release schedule.

An asset management system maintains artifacts that might be atomic or composite assets. Therefore, versions may need to be maintained at multiple levels as well. The metadata repository generally manages the macro-versioning or the versioning of composite assets, while the SCM manages the micro-versioning or the versioning of the atomic assets.

2.4.10 Prescription and Compliance

Enterprise architects and project architects need to provide guidance to the developers on standards that should be followed and assets that should be reused based on the project requirements. This type of prescriptive reuse is more powerful than voluntary

reuse. The list of prescribed assets and standards is captured in compliance templates which are associated with projects. A compliance template communicates asset requirements and asset solution sets to the project teams and serves as a governance mechanism to enforce and measure those standards.

2.4.11 Closed Loop Governance

In order to maintain competitive advantage, businesses must continuously improve the efficiency of their internal processes. Process improvement requires feedback - the metrics that measure how well the current process performs. The ability to design assets that fulfill the requirements and compare them against the runtime metrics to validate the performance is the focus of closed loop governance. With closed loop governance, the monitoring and management metrics, historical statistics, and performance information are acquired back into the asset management for better decision-making and process improvement.

Additional benefits include

- Better consumer reuse decisions due to the availability of performance metrics
- Reporting on consumer contract violations
- Producer summary for improvement - Asset producers use the feedback to improve the quality.
- Portfolio management decisions - the feedback allows portfolio managers to make educated decisions on asset prioritization, promotion and roadmap. For example, assets that don't perform well or are rarely used may be retired. Assets with a growing usage trend may be provided upgrade opportunities. New assets or new versions of existing assets may be created based on the correlation between asset usage metrics.

2.4.12 Configuration management

The asset management includes configuration management capabilities to maintain consistency of the assets. These capabilities help manage the changes to the overall system to ensure that the integrity is protected.

2.4.13 Rogue detection

A rogue asset is an asset put into production without proper governance. Rogue assets pose significant risk to the enterprise and diminish the business value. Rogue assets consume resources without accountability. Detecting and harvesting the rogue assets is a very valuable capability of an Asset Manager. Once harvested, these assets are added to the existing catalog of enterprise assets with corresponding dependency links. Rogue asset detection not only promotes visibility by making the asset discoverable but it also eliminates corporate risk by ensuring that regulatory compliance requirements are not violated.

2.5 Asset meta-model

This section describes a high level, abstract asset meta-model. This meta-model defines assets, classifies them based on high level categories and shows some examples of the types of assets. The SOA Service meta-model described in *ORA SOA Foundation* dives deeper into the specifics of the definition of service assets.

In an IT environment, everything can be considered an asset. The size and significance of the assets may vary a great deal but by definition they are still different types of

assets that constitute the architecture. Generally assets build on top of one another and relate to each other through complex interdependencies.

Maintaining the assets from the business processes all the way down to the implementation artifacts along with their relationships and dependencies enables line of sight and traceability back to the business goals and objectives. In order to achieve this benefit, it is important to have a common understanding of what constitutes an asset and how the assets are categorized.

In addition to the above-mentioned, a formal model in asset-centric engineering provides the following benefits.

- It supports automated harvesting of assets based on the standard and accepted models. Automated asset harvesting accelerates the process and improves the quality of asset metadata.
- It provides specific relationships between assets to jumpstart the asset harvesting and definition process. It also makes navigation and discovery of assets more intuitive.

2.5.1 Asset types

In a broad sense, assets are classified into atomic assets and composite assets.

- **Atomic assets**: Atomic assets are simple assets that are basic building blocks of solutions. Atomic assets are further classified into functional and non-functional assets.
- **Composite assets**: Composite assets are higher level, coarse grained, value added assets that are created by composing the atomic assets.

Figure 2–4 Asset Meta-model - Asset Types

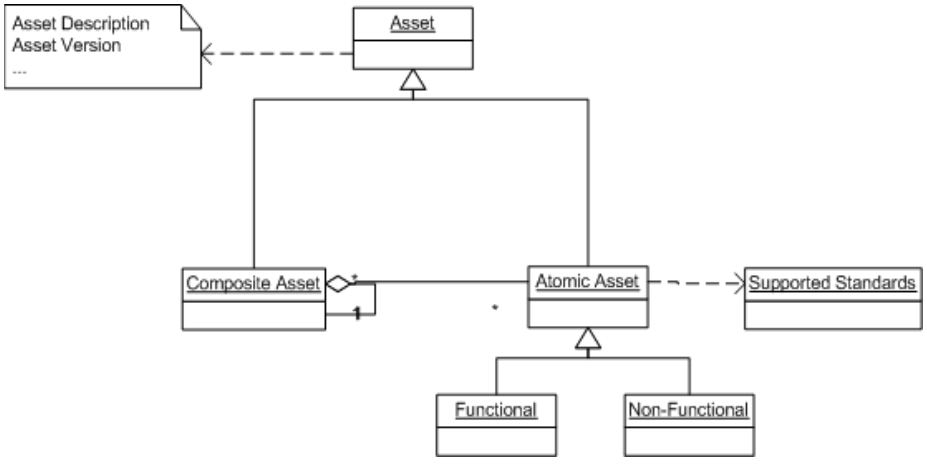


Figure 2–4 shows the high level classification of assets. Some assets are deployable and some are not. Deployable assets are deployed on a deployment platform that provides the end point, management and other runtime features like high availability and security.

2.5.2 Atomic Assets

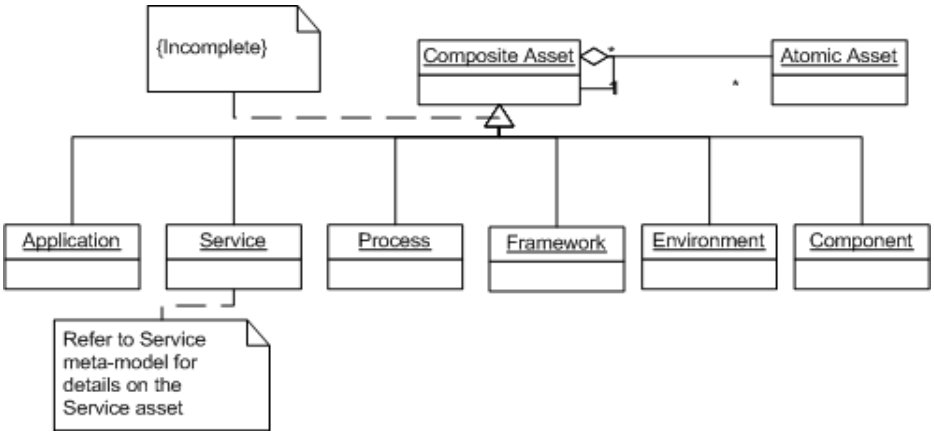
Atomic assets may be functional or non-functional. Some assets have supporting standards that describe how they are developed. Where applicable, assets should be

developed using these supporting standards. Atomic assets may be in a variety of forms including documentation, code, and configuration.

2.5.3 Composite Assets

Composite assets are made up of atomic assets or other composite assets. For example a (SOA) Service is a composite asset that might be made up of other Services and code assets. Some of the examples of composite assets are Service, Application, Process, Framework, Environment, and Component. As shown in the example in Figure 2–5, Web Service is a form of realization of a Service and this metaphor applies to the other asset types as well.

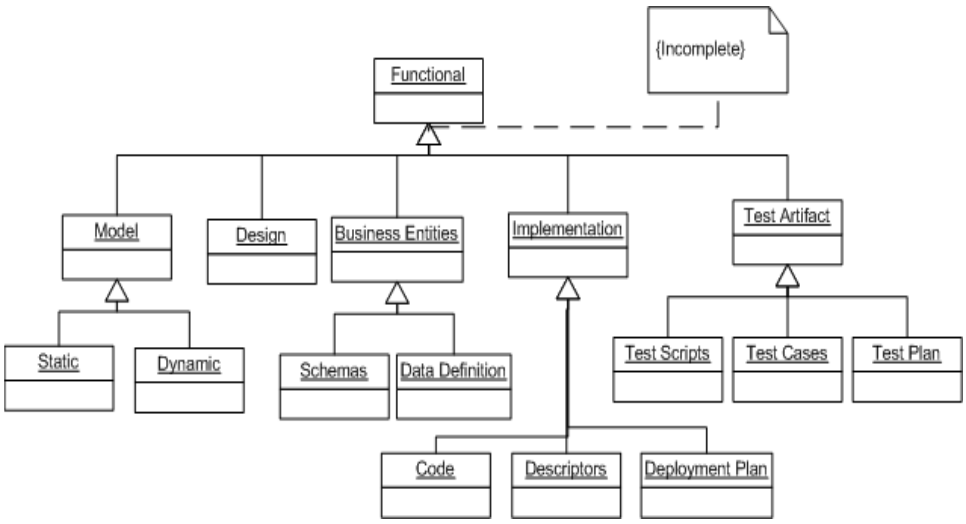
Figure 2–5 Asset meta-model - Composite Asset



2.5.4 Functional Assets

Functional assets are atomic assets that represent some piece of business functionality. They could be in a number of forms such as model, design, data, implementation or test artifacts. Figure 2–6 shows some of the examples of functional atomic assets.

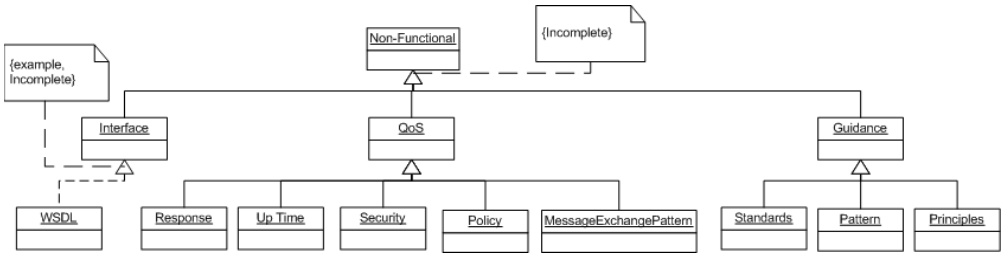
Figure 2–6 Asset Meta-model - Functional Asset



2.5.5 Non-Functional Assets

Non-functional assets are assets that support the architecture, development and RASP qualities of the infrastructure. As shown in Figure 2–7, some of the examples of non-functional assets are interface, policies that define Qualities of Service (QoS) and the architecture guidance artifacts like standards, patterns, and principles.

Figure 2–7 Asset Meta-model - Non-functional Assets



2.6 Engineering Principles

This section lists architecture principles related to ORA Engineering.

2.6.1 Asset-centric Engineering

Principle	**Asset-centric Engineering**
Statement	Asset-centric approach must be applied to engineering processes.
Rationale	Asset-centric engineering provides a governed engineering environment that improves visibility and promotes sharing and reuse.
Implications	▪ The development infrastructure must support asset-centric engineering. ▪ Assets must be associated with meaningful metadata that can be used to discover and interpret the asset. ▪ Existing assets must be reused to fulfill a whole or part functionality when available. ▪ All required functionality must be checked against the enterprise Metadata Repository for reuse opportunities, at the beginning of the project.

2.6.1.1 Traceability

Principle	**Traceability**
Statement	Asset functionality and architecture decisions must be traceable back to the business and system requirements.
Rationale	Functional assets and system architecture must support the goals of the business. The ability to trace the IT implementations back to the business and system requirements ensures that the required business capabilities are built properly and the business stays agile.

Implications	■ Design drives the development of assets. ■ Solutions developed must be traceable to specific business goals or objectives. ■ The architecture and business requirements must dictate what type of assets should be created for the given functionality.

2.6.2 Integrated Quality Management

Principle	**Integrated Quality Management**
Statement	Assets must be tested early and often to ensure highest quality possible.
Rationale	Asset quality must be assured at every step of the engineering process. Integrated quality management practices not only lead to improved quality of the assets but also ensure on-time delivery of business functionality.
Implications	■ Business requirements and service contracts drive testing of the assets. ■ Solutions developed must be integrated and tested early and often. ■ Assets (Code, components, SOA Services etc.) must always be unit tested. ■ 100% code coverage must be ensured.

2.6.3 Proactive Quality Assurance

Principle	**Proactive Quality Assurance**
Statement	An independent and formal QA must drive a proactive approach to asset quality.
Rationale	Asset-centric engineering provides a governed engineering environment that improves visibility and promotes sharing and reuse.
Implications	■ QA must have an active role and be aligned to the governance process. ■ QA is introduced early in the asset lifecycle and not just postmortem testing. ■ QA must establish processes, procedures, and automation that realize service completeness, consistency, and correctness. ■ QA must verify and validate completeness - assures that the technical and business requirements are met and satisfied. ■ QA must verify and validate consistency - assures that the standards are met and satisfied. ■ QA must verify and validate correctness - assures that the code performs per the consumer's specification. ■ QA must drive the configuration and change management process.

2.6.4 Reuse

Principle	**Reuse**
Statement	Architecture, infrastructure and design must support and enable reuse.
Rationale	Reuse is a fundamental principle that promotes business agility and cost savings. Reuse should be considered from the get-go and should not be an afterthought.
Implications	■ All software assets that have reuse capability (such as document templates, data models, for example) must be entered into the repository to maximize their reuse potential. ■ Reuse before Version before Buy before Build

2.6.5 Visibility

Principle	**Visibility**
Statement	Assets must be organized and exposed for visibility, easy navigation and discovery.
Rationale	Organizing assets in a way that enhances visibility would provide a holistic view of the enterprise from both functional and architecture perspectives. This promotes business agility by enabling discovery, sharing and reuse.
Implications	■ All assets and related metadata must be entered into a repository that supports asset discovery and asset lifecycle management. ■ Asset relationships and dependencies must be captured and tracked. ■ Assets must be assigned appropriate taxonomies. ■ Rogue assets must be discovered and catalogued.

2.6.6 Change Management

Principle	**Change Management**
Statement	Changes to assets must be managed in a controlled manner that supports the evolution and recovery of assets.
Rationale	The role of change control is critical in federated environments. In a shared environment, business requirements of the consumers may diverge from the original functionality of the assets, necessitating the creation and management of newer versions.

Implications	■ All Assets must be versioned. ■ All assets must be backed up. ■ Asset management must be used as a governance mechanism. ■ Impact of change to the assets must be assessed before performing the change. ■ All consumers must plan to move to the latest version of the assets. ■ The version of code and configuration must not change between environments.

2.6.7 Asset Packaging

Principle	**Asset Packaging**
Statement	Assets must be packaged using standards-based approaches with the goal of improving flexibility, reuse, and runtime performance.
Rationale	Applying packaging standards and best practices is a critical step in ensuring that the assets are deployed for the best quality and performance. It also accelerates the time-to-deployment.
Implications	■ Every reusable asset must contain at least one manifest file that self-describes the contents of the package. ■ Libraries and components in a package must not be duplicated. The classloader hierarchy must be used to design the packages to avoid duplication. ■ Common libraries must be placed outside the package to be loaded by a higher level classloader (e.g. System classloader). ■ Any components that can be precompiled must be precompiled in the package. ■ Libraries provided by the platform should not be included in the package. (e.g. Application Server system libraries) ■ Non-runtime artifacts must not be included in the deployment package. (e.g. build and test artifacts) ■ Packages must follow predefined industry or company standard naming conventions and structures. ■ Static content must not be included in the deployable package. They must be served separately in exploded format. ■ Packaging of components must be modular and all common components must be packaged as independent libraries that can be included in multiple packages.

2.6.8 Asset Deployment

Principle	**Asset Deployment**
Statement	The deployment process must be streamlined by automating the asset build, provisioning and promotion process.

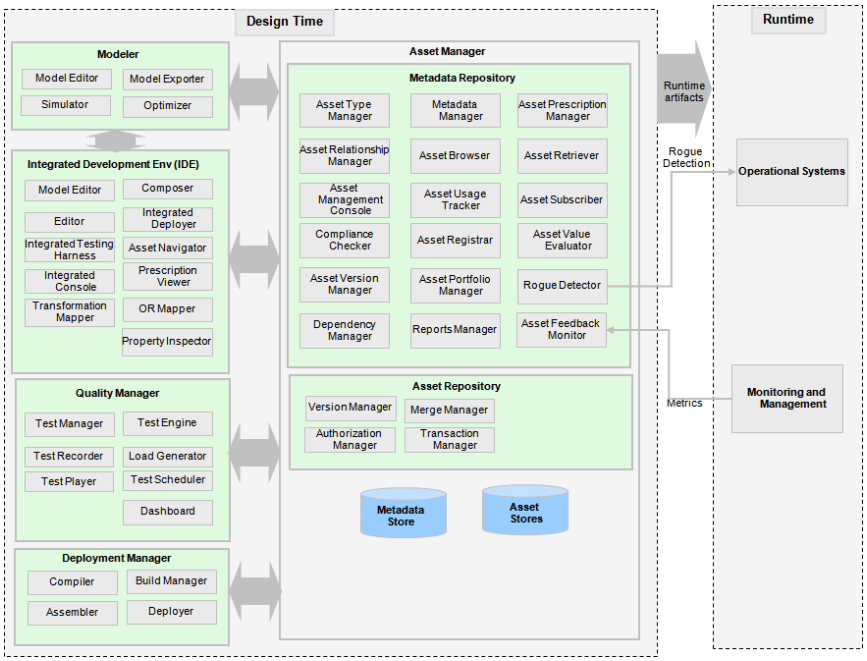
Rationale	Streamlining the deployment process helps minimize human errors and ensures that deployment best practices are followed.
Implications	■ An independent build environment must be used to organize and build the deployable artifacts. ■ The application must be tested in a production-like staging environment before production deployment. ■ The assets deployed in production must be the same build that was tested in the test environments. Untested or partially tested builds must not be deployed in production.

3

Engineering Logical View

The logical view shows the logical components of the Engineering environment and show how they are connected to each other. The primary logical components of the engineering environment are shown in Figure 3–1, which displays a generic logical model. Later in this section, a few scenarios are described using corresponding logical models.

Figure 3–1 Generic Logical View



The primary logical categories as shown in Figure 3–1 are:

- Modeler
- Integrated Development Environment (IDE)
- Quality Manager
- Deployment Manager
- Metadata Repository
- Asset Repository

Also shown are the inbound and outbound interactions between the design-time infrastructure and runtime infrastructure. Primarily the runtime artifacts are pushed to the operational systems for deployment and production.

The monitoring and management subsystem manages the operational systems, captures the runtime metrics and feeds them back into the design-time using various mechanisms for closed loop governance and continuous process improvement. Closed loop governance also involves harvesting assets throughout the lifecycle, not only at development, but also at build and deployment. This provides visibility throughout the lifecycle.

The monitoring and management subsystem primarily interfaces with the Metadata Repository to push the metrics that correspond to the design-time assets.

In addition, the rogue detection takes place in the production environment in order to avoid any accidental discovery of junk or orphaned assets.

The logical components shown in Figure 3–1 are described below.

3.1 Modeler

Modeling is a prime and foremost activity of the engineering process. Modeling bridges the gap between business and technology worlds through the language common to both sides. With the gaining popularity of Model Driven Architecture (MDA), modeling has taken a more critical role in engineering. Modeling proves useful in several parts of the enterprise engineering process including the following.

- Data modeling
- Functional modeling
- Process modeling
- SOA Service modeling
- Component modeling
- Use case modeling
- Architecture modeling
- Object modeling
- Event modeling

3.1.1 Model Editor

Model editor should support notations to capture static and dynamic behaviors of the systems. Static modeling focuses on capturing the instance attributes and snapshots of nodes and objects. Dynamic modeling generally refers to one or both of the following

- Behavior modeling that focuses on the internal state changes
- Interaction modeling that focuses on external collaborations.

3.1.2 Model Exporter

One key feature that is often overlooked is the ability to export models into a format suitable for viewing and reporting and that can be imported into other tools. When possible models must be exported using standards based interchange formats. Many industry standards such as Business Process Modeling Notation (BPMN) or XML

Process Definition Language (XPDL) have been created for modeling notation and representation.

3.1.3 Simulator

Simulator is responsible for running the models through real-world scenarios to observe the behavior and to make any necessary adjustments. Simulation may be based on conceived data or data captured from the live systems. The scenarios simulated should not only include the "happy path" but must also include the fault and exception scenarios. Simulator runs the scenarios and captures the behavior and results for further analysis and validation.

3.1.4 Optimizer

Models must be optimized at the source during the design-time. This practice avoids the need for last minute chaos and costly fixes. It also ensures that the runtime resources are sized appropriately at the beginning. Optimizer identifies and recommends performance and quality improvement opportunities.

3.2 Integrated Development Environment (IDE)

The Integrated Development Environment (IDE) provides the core capabilities required for development.

3.2.1 Model Editor

IDEs provide essential modeling features that are mostly developer based modeling. Modeling in the IDE enables round-trip engineering. IDEs include visual designers for modeling and models are synchronized with the code automatically.

3.2.2 Editor

The primary capability of the IDE is editing. The idea of editor is to provide an environment where code, configuration, and other assets can be developed quickly and efficiently. Editing features includes authoring, code editing, syntax checking, and code completion. Different types of assets require different editing capabilities and the editor should be versatile and extensible to support the needs of various asset development needs.

3.2.3 Integrated testing harness

Integrated testing harness allows the developers to perform unit testing of the code from the IDE. It also generates clients or client side stubs to assist with the testing. For example, SOAP clients and WSDL interfaces may be generated to test Web Services. Integrated testing harness generally provides request stack trace and performance metrics for troubleshooting and tuning.

3.2.4 Integrated console

Basic administration of the development server should be provided in the IDE. The minimum set of operations that are often used in the integrated console are start, stop and deploy. Integrated console also enables code debugging from the IDE.

3.2.5 Transformation Mapper

Transformation mapper provides a user friendly graphical interface for mapping source and target data or message representations. IDEs provide advanced XQuery capabilities to enrich and split the data elements.

3.2.6 Composer

Composer allows higher level composite assets to be created using existing atomic and composite assets such as SOA Services and components. Composer should have the ability to discover, examine and download the required interfaces. Composer should also support composite development standards like Service Component Architecture (SCA). SCA defines a generalized notion of a component and specifies how those components can be combined into larger composites structures.

3.2.7 Integrated deployer

Integrated deployer deploys and redeploys the code to the server in exploded or archived format from the IDE. When the application development completes, it undeploys it from the server. Exploded deployment allows testing of the updated code without a full deployment.

3.2.8 Asset Navigator

Asset navigator allows the assets to be discovered through find and search capabilities.

3.2.9 Prescription Viewer

Assets prescribed by the enterprise architects or other governance bodies are viewed through the prescription viewer. It may use technologies like Software Fingerprint Identification (SFID) to automatically track the usage of assets. This component also allows the developers to view and download what has been prescribed. Prescription viewer integration to the IDE allows the display of the prescribed list of assets in the IDE itself. Assets are usually prescribed through compliance templates that capture the assets that are required to be used by the projects to be compliant with the enterprise standards. Assets appearing in the compliance templates assigned to the project, and the details of which of the assets have been used by the user, are displayed by the prescription viewer.

3.2.10 Object Relational Mapper (ORM)

ORM enables the developers to explore the database schema and auto-generate the program objects and UI components through introspection of the data models. Most ORMs also generate basic inquiry and persistence operations for the entities defined in the data model.

3.2.11 Property Inspector

Property inspector allows inspection of the property values at design time for development and at runtime for testing.

3.3 Quality Manager

Quality manager includes the tools and frameworks that help assure that the generated assets meet the functional and non-functional criteria defined by the business requirements.

3.3.1 Test Manager

Test Manager organizes and manages the overall testing process. It provides a single unified platform for sharing information among team members.

Test Manager creates projects that group together and organizes test scripts, requirements that need to be tested, and issues resulting from the tests. Once created, the relationships among these items can be indicated, allowing quick and easy discovery of all information pertaining to a particular test script, requirement, or issue.

Test Managers typically offer the following capabilities for integrated requirements management and defect tracking for both manual and automated tests:

- **Requirements Management** - provides the ability to define and manage requirements for a specific project. One can specify details for each requirement, track the status of each requirement, and associate requirements with test cases to ensure testing coverage.
- **Test Planning and Management** - provides the ability to define and manage a test plan that incorporates both manual and automated test cases. The process can be automated by storing and executing the test scripts and capturing the test results automatically. Business requirements can be associated to the test cases to ensure testing coverage, and issues can be associated with the test cases so that they can be reproduced. Keeping track of how the issues were identified assists training and compresses troubleshooting times.
- **Defect Tracking** - provides the ability to create and manage defects, referred to as issues, for a specific project.
- **Reporting** - generates reports in standard formats for managing the overall testing process. Reports on requirements, tests, and issues are commonly generated.
- **Database Repository** - provides the ability to store test assets including test scripts, results, attachments, requirements, test plans, and defects in a common database.

Asset Manager links to the Test Manager to provide a connected view of requirements, project artifacts, test artifacts and test results.

3.3.2 Test Recorder

Test recorder allows the test scripts to be recorded for future playback. It records the user actions and data input. Test recorder also captures the think time and other delay times to simulate the real world patterns.

Visual Scripts record the interaction of the user with the application under test. These scripts serve as the baseline and can be replayed against new versions of the application to detect functional, regression, scalability, and post-deployment defects.

3.3.3 Test Player

Test Player plays back the scripts recorded by the Test Recorder. Test failures and interface differences are flagged by the player during playback. Test Players provide the ability to customize a variety of things including the following

- Add test cases
- Modify default tests
- Ignore specific test cases
- Connect Data Banks for feeding data sets
- Ignore specific failures
- Add custom programmatic tests

Sometimes there are system generated elements of data output that may vary between test runs. An example of this would be a system timestamp. Test players should allow these kinds of data to be ignored when comparing with the expected output.

3.3.4 Test Engine

The role of Test Engine is to execute the test scripts. It takes the input manually or from the test player and runs the test cases. It also interfaces with the test recorder to record the tests as they are run. Test engine should be able to run single threaded or multi-threaded.

3.3.5 Load Generator

Load generator generates load for performance or load testing. It simulates multi-user testing through multi-process or multi-threading. Load generator should also be capable of simulating complex think time patterns.

3.3.6 Test Scheduler

Test Scheduler is a test management tool that groups and runs multiple test scripts in sequence as a single job. Scheduler jobs can be scheduled to run automatically at specific times or be run manually at any time. Test Scheduler includes an editor to configure the test schedule information. Integration of the scheduler with the calendar interface is a desired feature that improves ease of use. Test scheduler's integrated viewer is used for viewing test schedules and results. Scheduler may also include a Notifier to notify the status and results of the tests run.

3.3.7 Test Dashboard

Test dashboard captures and displays a comprehensive summary of the test status and test results. It allows the stakeholders (Quality Assurance Managers and Project Managers) to make decisions on release management and quality control. Test dashboard displays the release status in various environments to show the progress from development to production and the quality improvements over releases.

3.4 Deployment Manager

Deployment manager manages the process of creating and archiving the binary representation to prepare them for deployment.

3.4.1 Compiler

The compiler compiles the source code and creates the binary representation. The implementation of the compiler depends on the underlying platform and the programming language used.

3.4.2 Assembler

Assembler comprises of a set of tools that help assemble the deployable archives. Certain parameters are best set during the assembly phase by the Assembler role. These parameters are typically set in deployment descriptors that can be edited independent of the code. Assembler tools allow these parameters to be set or edited without causing a total repackaging of the deployable artifacts.

3.4.3 Build Manager

Build Manager manages the builds, build related reporting, and documentation. It encapsulates the complexities of the build process and provides easy to use interfaces. Build Manager accelerates the build process by employing standard structures, conventions, and best practices for organizing artifacts and building applications.

Organizations should also harvest assets into their Metadata Repository at build time, leveraging Ant scripts or other tools. This ensures that there is visibility through the build phases of the lifecycle. The endpoints are updated in the Metadata Repository, and can be promoted to a runtime infrastructure like the Service Registry.

3.4.4 Deployer

Deployer is a set of deployment tools and frameworks that deploy the application to the application servers. Deployers provide a combination of script-based and GUI-based interfaces to target and deploy the deployable artifacts.

Assets should also be harvested at deployment. Updated endpoints appear in the Metadata Repository, and end users will have access to the deployed endpoints.

3.5 Metadata Repository

Metadata Repository is a collection of logical components that manage the metadata of the assets including the asset descriptions and asset dependencies.

3.5.1 Asset Type Manager

Before assets are added to the repository, asset types need to be defined and configured. Some asset types are standard and some are specific to the organization. Asset Type Manager is responsible for managing the asset types.

3.5.2 Asset Relationship Manager

An important use of the Metadata Repository is to analyze the impact of change. This requires the relationships between the assets to be defined and maintained. Asset Relationship Manager defines and maintains the asset relationship information.

3.5.3 Asset Management Console

Asset Management Console is the interface for managing the assets and the asset repository. It allows administrative tasks to be performed.

3.5.4 Compliance Checker

Compliance Checker checks the compliance to policies defined in the Metadata Repository. An example would be a standards template assigned to a project developer by the project architect or enterprise architect. The Compliance Checker

generates alerts if there are any violations or discrepancies. It might make use of SFID type of technologies to identify the libraries or software components used.

3.5.5 Asset Feedback Monitor

Asset Feedback Monitor collects the asset feedback from the consumers. Receiving and maintaining the feedback plays a key role in implementing closed loop governance. The feedback may be received manually through surveys and valuation forms or may be collected automatically through runtime metrics.

3.5.6 Dependency Manager

Dependency Manager maintains asset dependencies through the relationships defined by the Asset Relationship Manager. It creates dependency models or graphs to show the interdependencies between assets and assists with impact analysis.

3.5.7 Metadata Manager

This core component manages the metadata of assets. The asset metadata is defined through the Metadata Manager. The Metadata Manager also allows one to edit, organize, and update the asset description and associated metadata.

3.5.8 Asset Viewer

Asset Viewer is the interface to view the assets. It provides "Find" and "Search" functionalities to discover assets of interest using simple to advance search criteria. Asset Viewer should support a customizable view with varying levels of details.

3.5.9 Asset Usage Tracker

Asset Usage Tracker tracks the usage metrics of the assets and provides valuable input to the asset value evaluator for assessing the business value of the assets in quantifiable terms.

3.5.10 Asset Registrar

Assets are entered into the Metadata Repository through an asset registration process. Asset Registrar allows the creation, editing and removal of assets and associated metadata in the repository. Asset Registrars are the quality control gatekeepers for assets. As assets are harvested into the repository, Asset Registrars validate the quality of the assets prior to releasing them for use to the consumer community.

3.5.11 Asset Portfolio Manager

Planning the prioritization, development and delivery of assets is an important part of asset management and the Asset Portfolio Manager assists with managing the portfolio of assets.

3.5.12 Reports Manager

Reports related to asset inventory, asset usage and impact analysis are a value-added feature of metadata repositories. Reports Manager is responsible for everything related to the management of reports and report types. It creates reports based on out of the box templates as well as custom templates.

3.5.13 Asset Prescription Manager

Asset Prescription Manager prescribes the assets to projects or project developers. Prescriptive reuse is a more powerful concept than voluntary reuse. This component also keeps track of the prescription compliance status.

3.5.14 Asset Retriever

Asset Retriever is a component of the Metadata Repository that retrieves the assets and allows them to be downloaded by the consumer.

3.5.15 Asset Subscriber

Asset Subscriber subscribes the asset based on the request from the consumer. It notifies the consumer of any changes or updates to the asset.

3.5.16 Asset Value Evaluator

Asset Value Evaluator evaluates the value of the assets. It might use a number of different metrics to do it. For example, Predicted Net Hours Saved (PNHS) is a metric that quantifies the business value in terms of number of hours saved in development by reusing an asset. The Asset Value Evaluator provides input to the dashboard for display of the asset value.

3.5.17 Asset Version Manager

In a shared services environment, it is important to keep track of various versions of assets and how they are being used by consumers. Consumers may use different versions of the assets but are generally expected to move to the latest version of the assets at some point. The relationships between the asset versions and the details of how they differ should be maintained by the Metadata Repository. The Asset Version Manager manages the various asset versions which are usually the macro versions of the composite assets. The minor versions are generally not maintained by the Metadata Repository but are maintained by a version control system.

3.5.18 Metadata Store

The Metadata Repository uses the Metadata Store for storing all the data related to the management of the assets. The store needs adequate capacity to maintain all the information and should be tuned for optimal performance. The Metadata Store stores the metadata and may not store the actual payload. The Metadata Store also houses the parameters for optimizing the runtime performance of the assets. Metadata Repositories may include a content manager that may store the content in the Metadata Store.

3.6 Asset Repository

Asset repository refers to the tools or technologies that store the physical assets or payload, as opposed to the metadata that is stored in the Metadata Repository. A multitude of technologies may be employed in combination to manage the asset payloads. These may include

- Version control systems (VCS) or Software Configuration Management (SCM) may store versioned asset files.
- Configuration Management Databases (CMDB) may store configuration items.

- File Servers may store unversioned files and documents.
- Content Management Systems (CMS) may store structured or unstructured content files.

3.6.1 Version Manager

One of the primary uses of the asset repository is to manage the minor versions of the assets. Version control systems are specialized in managing the versioning of the assets. They provide a number of value added features like conflict resolution, file merge, branching and rollbacks. The Version Manager plays a key role in managing and controlling change in the projects.

3.6.2 Merge Manager

Quite often files need to be merged to resolve conflicts or to consolidate code in a federated development environment. Merge manager takes care of comparing the files and merging based on user specifications.

3.6.3 Authorization Manager

Assets may be restricted to certain users or groups. Authorization Manager provides authorized access to the assets including role based access control.

3.6.4 Transaction Manager

Transaction Manager supports a number of asset lifecycle transactions including the following.

- Check in: Allows assets to be added or updated in the repository
- Check out: Allows assets to be retrieved from the repository. It may use an optimistic strategy or pessimistic strategy. In a pessimistic strategy, the assets are locked when checked out, whereas in an optimistic strategy, the assets are not locked.
- Rollback: Allows the asset to be restored to a pervious version.
- Commit: Allows changes to be committed to the repository.

3.6.5 Asset Store

Asset Store is the database or file system where the asset payloads are stored and maintained.

3.7 Logical View Scenarios

This section presents a few sample scenarios that show how the logical components described in the generic logical model interact to accomplish the use case.

The following scenarios are discussed:

- Asset Lifecycle
- Service Consumption and subscription

3.7.1 Scenario 1: Asset Lifecycle

Asset lifecycle covers the creation, registration, usage, testing and retirement of assets.

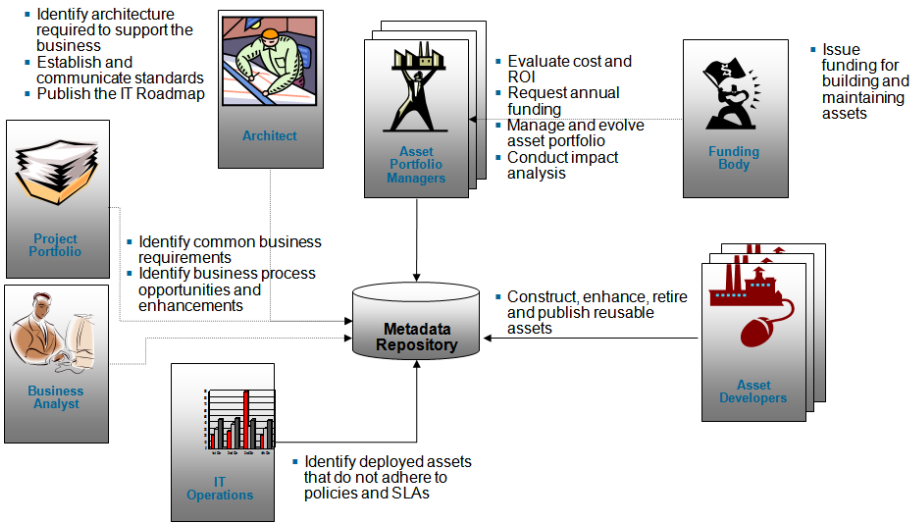
3.7.1.1 Use case

Following table summarizes the asset lifecycle use case.

Scenario Description	Asset Lifecycle
Overview	Assets go through several lifecycle stages from definition to retirement. Business drivers and business goals drive the definition and the implementation of the assets. The assets produced are either deployed directly or consumed by other higher level assets.
Actors	Architect, Registrar, Developer, Tester, Portfolio Manager
Pre-Conditions	■ Business requirements identified ■ Business process opportunities and enhancements identified ■ Funding has been approved and issued. ■ Architecture required to support the business has been identified ■ Project standards have been established and communicated ■ IT Roadmap has been published
Post-Conditions	Asset has successfully been produced and in operation
Basic Flow	The actors log into the Metadata Repository and they perform the necessary steps to accomplish the action of that lifecycle phase that include the following ■ Definition ■ Registration ■ Discovery ■ Consumption ■ Development ■ Testing ■ Production ■ Retirement The artifacts generated during the lifecycle phases can be made available through the Metadata Repository.

Figure 3–2 shows the asset lifecycle process and the role of the Metadata Repository.

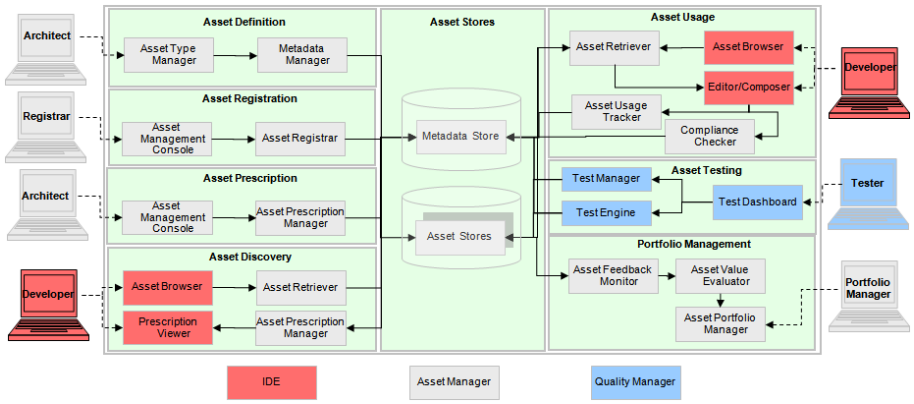
Figure 3–2 Asset Lifecycle Process



3.7.1.2 Logical View

Figure 3–3 shows the logical model for the asset lifecycle use case.

Figure 3–3 Logical View - Asset Lifecycle Use Case



- **Asset Definition**: An asset is defined by the architect using the Asset Type Manager. Asset Type Manager allows the asset types to be defined. The asset types may be standard or customized to suit the needs of the project. Defined assets are stored in the Metadata Store. Once defined, they can be reused for other assets and by other projects.
- **Asset Registration**: Once the asset type has been defined, assets of that type may be registered through the Asset Management Console. The Asset Registrar is the logical component that registers the assets. When assets are registered, the metadata associated with the assets are also defined and they become available to be reused by the consumer community.
- **Asset Prescription**: The enterprise architect or project architect determines the assets that must be used by the project teams. Those assets are prescribed to the developers based on the standards and templates defined for the project. This is done through the Asset Prescription Manager.

- **Asset Discovery**: Developers use the asset browser to find or search assets of interest and retrieve the details through the Asset Retriever. The assets that have been prescribed for the project are retrieved and displayed by the Prescription Viewer. The developers reuse components, SOA Services, and libraries when appropriate.
- **Asset Usage**: The developer downloads the asset through the Asset Retriever and creates composites reusing the downloaded assets. The Compliance Checker ensures that the compliance requirements are met. For example, it makes sure that the assets prescribed are used and no other violations are detected. Tools use technologies like Software File Identification (SFID) to track asset usage. Asset usage is tracked by the Asset Usage Tracker. The new assets created by the developers are registered through the Registrar through the asset registration process shown in the diagram.
- **Asset Testing**: Test scripts are managed and organized by the Test Manager and the Test Dashboard provides the interface to schedule and run the tests. The Test Engine runs the scheduled or manual tests and the results are fed back into the Metadata Repository.
- **Asset portfolio management**: Asset Portfolio Management plays a crucial part in planning asset production and consumption. The portfolio manager would also be interested in the determined value of the assets to manage and rationalize the asset portfolio. Asset Feedback Monitor receives the feedback from the users and metrics from operational systems. Asset Value Evaluator evaluates and quantifies the value of the asset that would be useful for the Asset Portfolio Manager.

3.7.2 Scenario 2: Service Consumption and Subscription

In a Service-Oriented Architecture, service assets are produced and published in the Metadata Repository for consumption by potential consumers. Consumers search the repository to find SOA Services that they can reuse and download them for use in their composite applications. They also subscribe to the SOA Service to be notified of any changes to them.

3.7.2.1 Service Consumption Use case

Following table summarizes the SOA Service consumption use case.

Scenario Description	Service Consumption

Overview	Service consumers need to be able to quickly and effectively search, evaluate and extract services. Before a consumer begins to develop new project artifacts/services or modify existing artifacts/services that may have developed in the past, the consumer searches the Repository for existing services that meets his/her needs. To discover a service, the consumer may: ■ Navigate through the tree of the desired view to locate the desired service within the Repository ■ Search to quickly find a service, using any combination of keywords, service category, or area of the business or technical architecture ■ Use search facilities from within the development environment (IDE) to retrieve services from Repository ■ Utilize a UDDI browser to discover services within Repository during design time The consumer then evaluates the service's applicability to his/her project and decides to use the service as well as be notified of changes.
Actors	Service Consumer
Pre-Conditions	■ Valid user of the system ■ Required services exist within the registry
Post-Conditions	■ Consumer consumes the asset in a minimal number of steps ■ Consumer downloads any payload (files) associated with the assets
Basic Flow	■ Consumer logs into the Repository. ■ The consumer enters a keyword into the basic search field. ■ The consumer selects an asset from the list and begins evaluating the asset's details. ■ The consumer reviews extraction history, star ratings, forum responses, requirements, documentation, relationships and all other metadata. ■ The consumer selects the Extract button from the asset detail screen. ■ The consumer selects a project and possibly other related assets. ■ The consumer receives the asset's payload.
Alternate Flow #1 (Advanced Search)	■ The consumer selects the Advanced Search option. ■ The consumer enters two or more search criteria to narrow his/her search. ■ (Resume Step 3. in Basic Flow)
Alternate Flow #2 (Browse Taxonomy)	■ The consumer selects the Browse Tree option. ■ The consumer selects a type of taxonomy ■ The consumer selects a specific category ■ (Resume Step 3. in Basic Flow)
Alternate Flow #3 (IDE)	■ The consumer chooses the Search Repository option from the IDE ■ The consumer enters search criteria in the resulting panel ■ (Complete steps 3-4 in Basic Flow) ■ The consumer follows the wizard in the IDE to extract the asset into the local environment

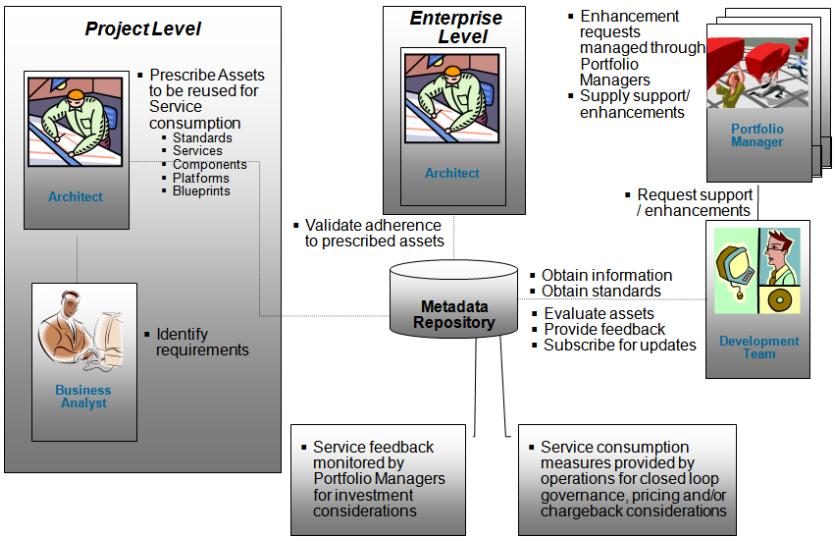
3.7.2.2 Service Subscription Use case

Following table summarizes the SOA Service subscription use case.

Scenario Description	Service Subscription
Overview	Service consumers often need to be notified of changes to a service, or may wish to register their interest in a service. Once a service of interest is discovered, the service consumer chooses to subscribe to the service. The consumer may then receive notifications of changes to the service (new versions, deprecation of a service, etc.), as well as notifications and messages from the service producer. The consumer may also choose to unsubscribe to the service if desired.
Actors	Service Consumer
Pre-Conditions	▪ Valid user of the system ▪ Required assets exist within the registry
Post-Conditions	▪ Consumer is subscribed to the asset ▪ Consumer is automatically notified (via email) of any major changes to the asset
Basic Flow	▪ Service consumer logs into the Repository ▪ Consumer finds an asset using any of a variety of search methods ▪ Consumer selects the Subscribe option for the asset

Figure 3–4 shows the SOA Service consumption process.

Figure 3–4 SOA Service Consumption Process



At the project level, a couple of major activities happen. The project architect prescribes assets to be used by the development team. These assets may include standards, service recommendations, components, platforms, and blueprints. Independently the requirements are gathered and documented. The Business Analyst registers the requirements in the metadata repository for visibility, discovery, and sharing.

The development team obtains information and standards from the Metadata Repository and evaluates SOA Services and other assets. Services that meet the criteria

are downloaded and consumed in the composite application. The development team also provides feedback on the usage and the realized or expected value of them. They can also subscribe to receive any updates to the asset to ensure that the latest and greatest version of the service is used.

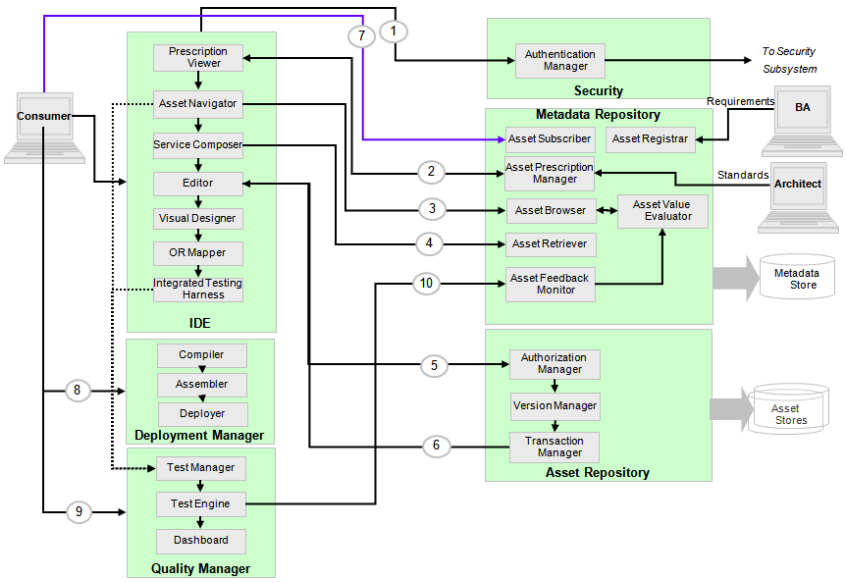
As a governance activity, the enterprise architects validate adherence to the prescribed assets. Any exceptions are flagged and investigated for the cause and remedial actions are taken. The asset prescribed may be unsuitable or unusable for the project for several reasons. The reasons are identified and analyzed. Depending on the analysis, a number of actions are possible including the creation of a new version of a service or the use of a local component.

If a modification is required for the project, then Portfolio Management is informed of the enhancement requests. Portfolio management plans the release of the future versions and keeps track of the usage of the enterprise SOA Services and other assets.

3.7.2.3 Logical View

Figure 3–5 shows various logical components that are involved in this use case. As a precondition, the business analyst identifies the requirements and registers them through the asset registrar component. The standards and templates are prescribed by the architect through the asset prescription manager.

Figure 3–5 Logical View - SOA Service Consumption Use Case



The following list describes the logical flow for the SOA Service consumption.

1. The service consumer first logs into the system and is authenticated using the security plug-in. The security system may be pointing to an enterprise security store for users and roles.
2. If any assets have been prescribed to the user, the Asset Prescription Manager pushes them to the Prescription Viewer so that the user can view and act upon them.
3. The consumer uses the Asset Navigator to browse the asset directory to discover the asset of interest. The Asset Value Evaluator provides the determined value of

the assets so that the consumer can make educated decisions on if the assets should be (re)used or not.

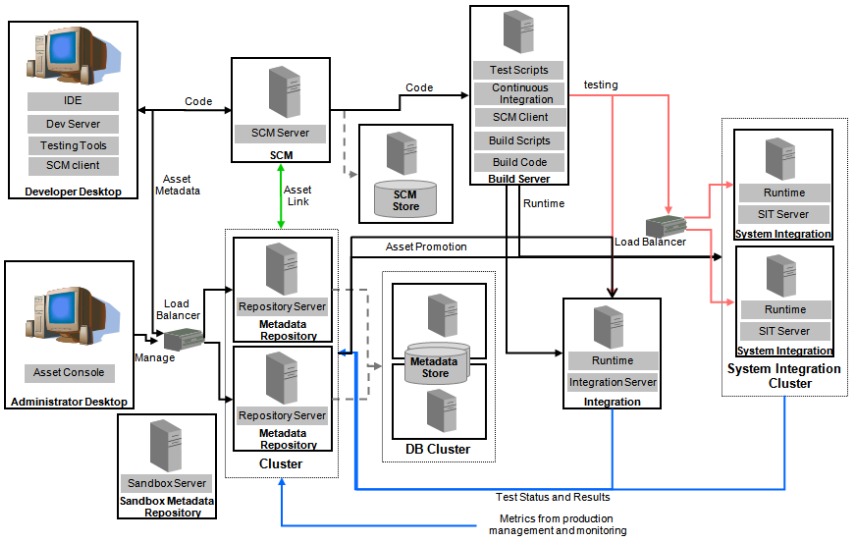
4. Once the consumer determines that the asset should be (re)used, the intention is recorded and the Asset Retriever allows the asset payload to be downloaded. In some cases, the consumer can select the desired endpoint and that endpoint is automatically embedded into their code base.
5. The actual payload may not be stored in the Metadata Repository and it may need to be pulled in from other systems like Version Control System or a file server. The consumer logs into the system that maintains the payload of the asset.
6. The asset is copied using the asset transaction manager. The consumer uses the asset to create the higher level composite asset that is further developed in the editor which may require the development of additional data access components that are developed using the O/R Mapper. The composite service is unit tested using the integrated test harness to ensure that it works according to specifications.
7. Consumer subscribes to the asset through the asset subscriber. Subscribers receive notifications when the asset is updated or new versions are created.
8. Once the code is completed, the service is compiled, assembled and deployed in one of the testing environments. It is also harvested into the Metadata Repository to get visibility into the test endpoint. The test endpoint can go through a governance workflow, and then the Metadata Repository can promote the test endpoint to the Test Registry.
9. The service is tested using the Quality Manager. The Test Manager manages the test scripts and the test engine runs the test scripts. The results are shown in the dashboard. The dotted line shows that the discovered service is tested for functionality before the reuse decision.
10. The test results are fed back into the asset feedback monitor and the asset performance is recorded.

4

Deployment View

The deployment view describes packaging and deployment related aspects of ORA Engineering. Figure 4–1 shows a typical deployment view of the ORA Engineering components.

Figure 4–1 Deployment View



4.1 Developer Desktop

Developer desktop runs the components that are needed for basic modeling, development, and unit testing of applications. It runs the IDE and the integrated modeler.

Development server is usually run locally for flexibility and isolation. It is run in single server mode with no clustering due to developer desktop limitations. Unit testing tools and integration testing tools are installed for quality management. Integrated SCM client is used for version control and code synchronization. Developers may also use browser based tools for the repository to publish and consume assets.

4.2 Administrator Desktop

Administration of the development infrastructure is done through browser-based and native asset management console and server management consoles. The administrator

is responsible for managing the development infrastructure including the Metadata Repository and the SCM.

4.3 SCM Server

The SCM server may be deployed in a clustered configuration or non-clustered configuration depending on the service level agreements. SCM server manages the code base and configuration. It uses file store or database for maintaining the asset payload and to manage the versioning of the assets.

4.4 Metadata Repository

The Metadata Repository is typically deployed as a single enterprise-scoped instance to promote a single view of the enterprise. In order to support the whole enterprise, it is clustered with multiple servers supporting the load requirements. Incoming requests are load balanced using a hardware or software load balancer. Metadata store is typically a clustered (e.g. RAC) medium performance database that can fulfill the asset management and search requests satisfactorily.

A sandbox Metadata Repository may be deployed for training and testing purposes. It is usually a standalone instance with no integration with other systems.

The repository receives the production operational metrics from the production systems for the registered assets. It also connects to the production systems for rogue asset detection.

4.5 Build Server

Build server is used for building the runtime packages and for testing before deployment. It houses and runs the build scripts and test scripts. Continuous integration tools (like Cruise Control) are run on the build server for scheduled and automated build and testing. The build server usually pulls the latest or specified version of the source code from the SCM before each build and deploys the runtime built to the appropriate environment (integration in this example). Any testing against the integration server can also be run from the build server. Assets are generally harvested from the Build into the Metadata Repository to provide visibility throughout the lifecycle.

4.6 Integration Environments

An integration environment is used to run the integration tests. There could be multiple levels of integration testing environments. Figure 4–1 shows two integration environments, one primarily for developers to test their components with the rest of the application and a system integration testing environment to perform a full integration testing.

The integration testing environment is typically a single server to perform integrated functional testing of individual components.

The system integration environment is typically clustered and it is the first environment for testing cluster-related features in the application.

Test results are fed back into the Metadata Repository for tracking and valuation purposes. Some assets are promoted from the Metadata Repository to the integration servers (e.g. service registry) directly through approval processes, which are shown by the "asset promotion" line in Figure 4–1.

4.7 Best practice deployment environments

When it comes to setting up the development, testing and production environments, a number of different configuration choices are available. The best configuration for any organization depends on its specific requirements and capabilities. Figure 4–2 describes an ideal best practice deployment configuration that addresses the primary concerns of the stakeholders.

Figure 4–2 Best Practice Deployment Environments

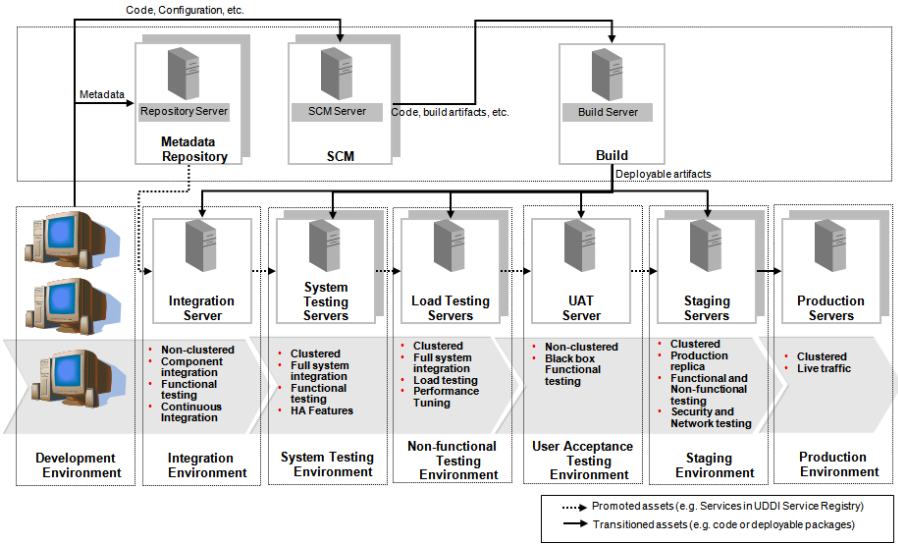


Figure 4–2 shows the movement of code and configuration across various environments. Ideally the following environments would be created in an enterprise.

- Development
- Integration
- System Testing
- Non-functional testing
- User Acceptance Testing
- Staging
- Production

It is not expected to have all the environments but most medium to large enterprises would require similar environments. Some large organizations have additional environments for parallel testing and integration.

It should be noted that the Build server is linked to all of the environments except development and production. In development, builds are done locally and in production, builds are transferred from staging as a best practice. This ensures that the code deployed in production has been tested thoroughly and has not been touched or rebuilt since then.

The diagram also shows the characteristics of each of the environments. The characteristics of the environments are summarized in the table below.

Environment	Characteristics

Development	· Single instance
	· Integrated with the IDE and development tools
	· Debug enabled for troubleshooting
Integration	· Non-clustered environment for testing the integrated code base
	· Component integration - components developed by different developers are consolidated and testing in this environment
	· Functional testing - This is primarily a functional testing environment.
	· Continuous Integration - This is an ideal environment for continuous integration and helps detect integration issues early.
System testing	· Clustered environment where clustering related high availability features are tested
	· Full system integration testing happens in this environment.
	· Functional testing is still the primary objective in this environment
Non-functional testing	· Clustered to handle the high load requirements
	· Full system integration environment
	· Load testing is performed in this environment
	· Performance Tuning of code and servers are done based on the test results
User acceptance testing	· Non-clustered environment as the primary objective is end user black box functional testing
Staging	· Clustered environment
	· Staging is a production replica that mirrors production environment
	· Both functional and non-functional testing are performed in this environment
	· Security and production network testing happens in this environment
	· May use data from or similar to production environment
Production	· Clustered environment
	· Supports live traffic from the users
	· May include performance monitoring tools for monitoring performance or gathering metrics.

5

Product Mapping View

This section describes how Oracle products fit together to realize the engineering infrastructure defined in the previous sections.

5.1 Products Included

There are a number of products from Oracle that can be used individually to satisfy specific engineering needs, or used in combination to establish a complete engineering infrastructure.

- **Oracle JDeveloper and ADF**: Oracle JDeveloper is an integrated development environment (IDE) that simplifies the development of Java-based applications and user interfaces with support for the full development life cycle. The Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java Platform, Enterprise Edition (JEE) standards and open-source technologies to simplify and accelerate implementing solutions. Oracle ADF simplifies the development of enterprise solutions that search, display, create, modify, and validate data using web, wireless, desktop, or web services interfaces. Used in tandem, Oracle JDeveloper 11g and Oracle ADF provide an environment that covers the full development lifecycle from design to deployment, with drag-and-drop data binding, visual UI design, and team development features built in.
- **Oracle SQL Developer Data Modeler**: Oracle SQL Developer Data Modeler is an independent, standalone product with a full spectrum of data and database modeling tools and utilities, including modeling for Entity Relationship Diagrams (ERD), Relational (database design), Data Type and Multi-dimensional modeling, full forward and reverse engineering and DDL code generation. The Data Modeler imports from and exports to a variety of sources and targets, provides a variety of formatting options and validates the models through a predefined set of design rules.
- **Oracle Business Process Analysis (BPA) Suite**: Oracle Business Process Analysis (BPA) Suite speeds process innovation by rapidly modeling business processes and converting them into IT executables. Oracle BPA Suite delivers a comprehensive set of integrated products that allows business users to design, model, simulate, and optimize business processes to achieve maximum operational efficiency.
- **Oracle BPM Studio**: Oracle BPM Studio is a desktop application that allows modeling and implementation of business processes. It creates a common interface for business analysts and developers by providing common views into the same process model. Oracle BPM Studio allows integration and design of business

activities using a process driven method to coordinate and manage internal and external business services.

- **Oracle Enterprise Pack for Eclipse (OEPE)**: Oracle Enterprise Pack for Eclipse is a set of certified plug-ins supporting JEE and Web Service standards where Eclipse is the preferred Integrated Development Environment (IDE) within an organization. As part of Oracle Fusion Middleware, the Oracle Enterprise Pack for Eclipse supports development with technologies including Database, Java SE, Java EE, Web Services, XML, and the Spring Framework.
- **Oracle Enterprise Repository (OER)**: Oracle Enterprise Repository serves as the core element to the Oracle SOA Governance and metadata management solution. Oracle Enterprise Repository provides a solid foundation for delivering governance throughout the development lifecycle by acting as the single source of truth for information surrounding assets and their dependencies. Oracle Enterprise Repository provides a common communication channel for the automated exchange of metadata and asset information between consumers, providers, policy decision points, and additional governance tooling. It provides the visibility, feedback, controls, and analytics to deliver business value.
- **Oracle Service Registry (OSR)**: Oracle Service Registry is a fully V3-compliant implementation of the UDDI (Universal Description, Discovery and Integration) specification. A UDDI registry provides a standards-based foundation for locating services, invoking services and managing metadata about services (security, transport or quality of service). A UDDI registry can store and provide these metadata using arbitrary categorizations, called taxonomies.
- **Oracle Application Express (APEX)**: Oracle Application Express (Oracle APEX) is a rapid web application development tool for the Oracle database. Using only a web browser and limited programming experience, it allows the development and deployment of professional applications that are both fast and secure.
- **Oracle Application Testing Suite (ATS)**: Oracle Application Testing Suite is an integrated, comprehensive application testing solution that provides the tools needed to ensure the scalability and reliability of business-critical applications.
- **Oracle Real Application Testing**: Oracle Real Application Testing offers an easy-to-use change assurance solution that enables businesses to fully assess the outcome of a system change in a test environment, take any corrective action if necessary, and then to introduce the change safely to production systems, minimizing the undesirable impact on them. Real Application Testing offers two unique features, Database Replay and SQL Performance Analyzer (SPA) that together provide a comprehensive and flexible solution for assessing impact of changes on test systems using real production workload thereby enabling businesses to adopt new technology changes with minimal risk, and at a significantly reduced cost while at the same time improving their SLAs.
- **Oracle Enterprise Manager (OEM)** - OEM is a family of management products, to manage Oracle environments. OEM enables centralized management functionality for the complete Oracle IT infrastructure, including systems running Oracle and non-Oracle technologies. OEM is a single, integrated solution for managing all aspects of the Oracle Grid and the applications running on it. It delivers a top-down monitoring approach to delivering the highest quality of service for applications with a cost-effective, automated configuration management, provisioning, and administration solution.
- **Oracle Database and Real Application Clusters (RAC)** - Persisting and managing the data is a fundamental requirement of any infrastructure. Oracle database is a proven and market leading database that offers sophisticated clustering and

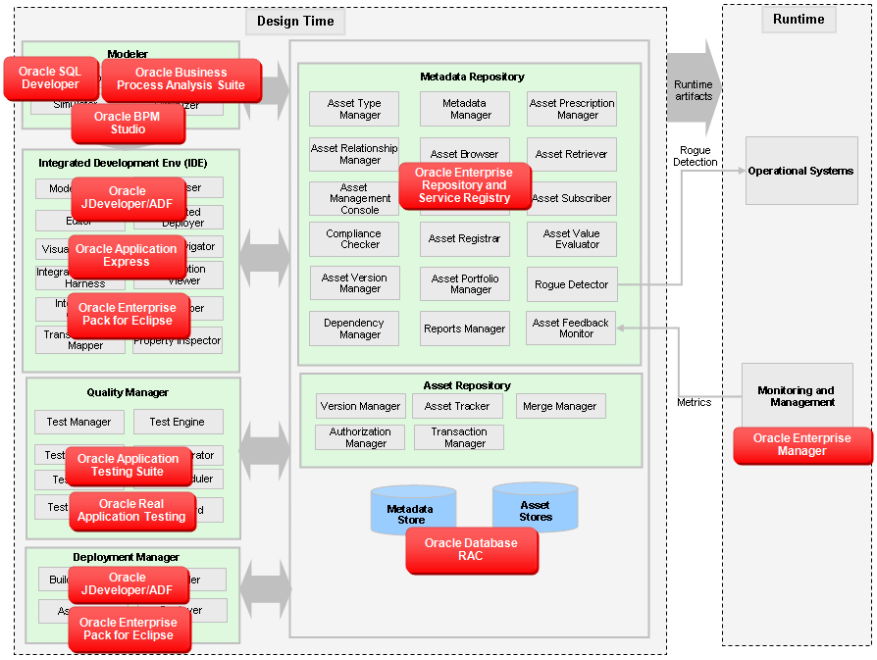
management features to implement the asset stores. Oracle Real Application Cluster (RAC) provides scalability and high availability in the database tier. The database tier can be horizontally scaled by adding database server instances that access the storage grid. The RAC instances can be configured for load balancing or failover based on the specific needs of the asset stores.

5.2 Product Mapping

The following section illustrates the mapping of Oracle products onto the Logical View.

The mapping diagram in Figure 5–1 positions each product with respect to their primary role. However, some products that have some high-level functionality that overlaps with other products, which is not shown on the diagram. To understand the differences between this overlapping functionality, for a complete list of product features, architecture documentation, and product usage please consult the Oracle Product documentation.

Figure 5–1 Product Mapping View



5.2.1 Modeler

Oracle has a number of modeling tools that cover a wide spectrum of modeling from business process modeling to database modeling.

Oracle BPA Suite and Oracle BPM suite provide business process modeling, simulation and optimization capabilities. Oracle SQL developer data modeler provides all the necessary capabilities for data modeling and data design.

5.2.2 Integrated Development Environment (IDE)

Oracle JDeveloper is a complete IDE that provides all the capabilities listed in the logical model. Oracle JDeveloper provides a number of modeling capabilities

including business process (BPEL) modeling, data modeling and UML modeling. JDeveloper and ADF provide declarative and visual development environment to accelerate solution development. The JDeveloper editor provides an advanced coding environment with code assist, code navigation and interactive code templates features.

JDeveloper integrates with the Oracle Enterprise Repository (OER), so developers can browse, search, evaluate reusable assets based on the metadata and consume them when composing solutions. JDeveloper Property Inspector provides improved access to help and actions that can be launched for a given property.

Oracle Enterprise Pack for Eclipse extends the IDE capabilities to the Eclipse platform. OEPE Object Relational Mapping (ORM) Workbench provides Java Persistence API (JPA) entity generation, annotation, visualization tools, data editing tools, schema viewer, and DDL export. OEPE also supports visual editing of code and deployment descriptors. Eclipse also has integration with Oracle Enterprise Repository and supports asset prescription and asset usage tracking through Software Fingerprint ID (SFID).

Although not a full-fledged IDE, Oracle Application Express (APEX) provides some of the basic capabilities to model, design and develop web based applications driven by the application database using web based development environment. APEX supports visual and declarative programming that is generated through wizards and property sheets.

5.2.3 Quality Manager

Most of the Oracle products include quality management capabilities for unit testing and troubleshooting. Oracle JDeveloper provides test harness to test and troubleshoot application components.

Oracle Application Testing Suite is a comprehensive testing suite that provides much of the functionality needed for the Quality Manager logical component. It includes a test manager to manage test plan, test documents and scripts. It provides several capabilities including functional testing, load testing, job scheduling and report generation.

Oracle Real Application Testing is a quality management solution focused on the database. It offers a solution for capturing real production workload and replaying it on a test system with identical characteristics. SQL Performance Analyzer (SPA) analyzes the performance before and after changes to the SQL execution plan and produces reports to assist the troubleshooting process.

5.2.4 Deployment Manager

Both Oracle JDeveloper and OEPE provide packaging and deployment capabilities. Integrated deployment from the IDE allows the applications in development to be targeted to the development server. These tools also provide a script based interface to compile, build and deploy applications. They also provide graphical interface to edit the deployment descriptors and assemble the applications.

5.2.5 Metadata Repository

The combination of Oracle Enterprise Repository (OER) and Oracle Service Registry (OSR) complete the capabilities of the Metadata Repository. OER provides a rich set of asset management functionalities including the following:

- Asset type management
- Asset relationship management

- Asset registration
- Dependency analysis
- Compliance checking
- Asset prescription
- Asset versioning
- Asset usage tracking
- Asset reporting
- Asset subscription
- Rogue detection
- Asset value evaluation
- Asset feedback management for closed loop governance

OSR provides a UDDI compliant metadata interface for runtime components. OER integrates with OSR and exports service assets automatically to OER. OSR collects the service metrics and feeds back into OER for process improvement.

OER leverages the workflow engine in the Oracle BPM to streamline and automate governance activities.

Oracle Service Bus (OSB) 11g leverages Oracle Enterprise Pack for Eclipse (OEPE) for asset harvesting. There are plug-ins that allow harvesting from OSB eclipse, as well as, search, review and download of assets.

5.2.6 Monitoring and Management

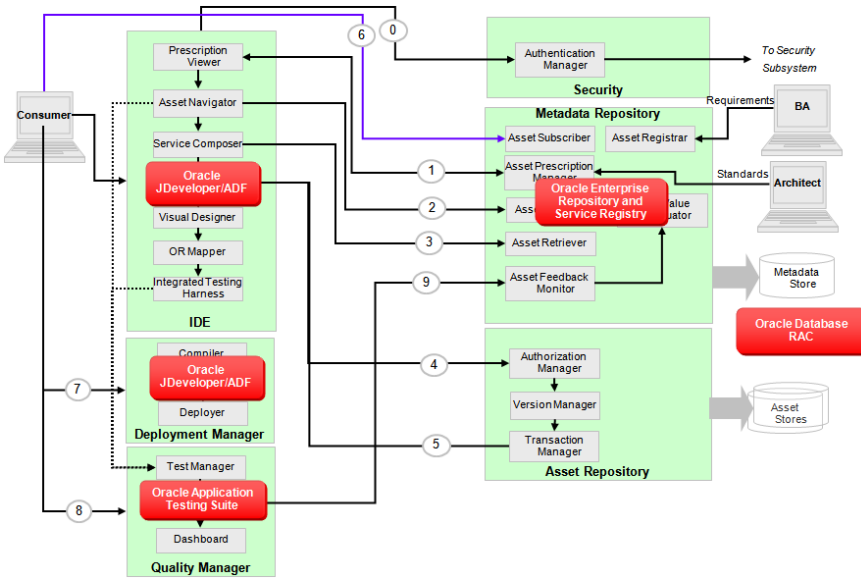
Oracle Enterprise Manager is the primary monitoring and management product. Monitoring and Management is covered by the *ORA Monitoring and Management* document in detail.

5.3 Product Mapping View - Service Consumption

Figure 5–2 shows the product mapping view for the SOA Service consumption use case discussed in the logical view section.

- Oracle JDeveloper provides the IDE related capabilities such as asset navigation, service composition, visual design, OR mapping and integrated testing. It also provides the deployment capabilities.
- Oracle Application Testing Suite provides the quality management capabilities such as test management and test engine.
- Oracle Enterprise Repository (OER) provides the asset management capabilities including asset registration, asset subscription, asset prescription and asset browsing. Oracle Service Registry provides the conduit to gather metrics into OER from the production systems. OER provides the prescription viewing capabilities through Eclipse and VS.Net plug-ins.

Figure 5–2 Product Mapping - SOA Service Consumption Use Case



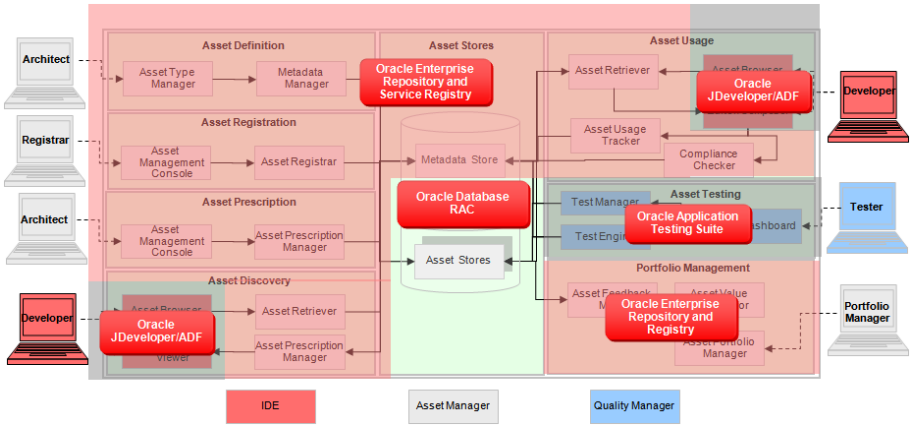
5.4 Product Mapping View - Asset Lifecycle

Figure 5–3 shows the Oracle products mapped to the asset lifecycle use case discussed in the logical view section.

Most components in the asset lifecycle use case are provided by Oracle Enterprise Repository. These components are highlighted in Figure 5–3.

Oracle JDeveloper/ADF provides the developer integration capabilities and Oracle Application Testing Suite provides the testing capabilities for the tester.

Figure 5–3 Product Mapping - Asset Lifecycle Use Case



6

ORA Engineering Best Practices

This section lists best practices related to ORA Engineering.

6.1 Metadata Repository Vs. Version Control System

One of the most frequently asked questions is "What is the difference between version control system and a Metadata Repository? Is it necessary to have both of these components?"

Version control systems are tools that track the history of changes that occur during the development of software, and allow users to access any version of the software. They manage problems that occur from multiple developers working simultaneously on the same code. They allow developers to "check out/check in" the code that they are working on and they enable consistent and repeatable builds. A version control system manages software "work in process".

The role of a Metadata Repository is two-fold. A Metadata Repository is an inventory of completed software or "finished goods inventory", and should serve as the channel of distribution. The inventory in the Metadata Repository should have passed the QA process, and should be packaged for final distribution. This means that the compiled and/or source code, documentation, test results, etc., are complete and can be consumed by other developers and stakeholders. The Metadata Repository manages the macro-version of these composite assets.

The Metadata Repository should work in conjunction with the organization's version control systems. Organizations should be able to store their reusable assets in a version control system, on a file server, ftp server, web server, or any other "warehouse" or storage system of their choice but still be able to link to them from the Metadata Repository.

In order to effectively consume the organization's "finished goods" software inventory, developers must be able to:

- Easily find what they are looking for
- Quickly determine the relevance to their needs
- Effectively evaluate technical compliance
- Obtain a copy of the asset or the relevant endpoint information

The Metadata Repository serves this purpose.

As with any channel of distribution, there should be a customer relationship management mechanism that tracks information about the consumer. The Metadata Repository should track information such as the projects that are using the company's software assets. The system should automatically notify consumers of new versions.

Finally, the system should be able to track the "revenues" or savings realized through the consumption of the organization's software assets.

Version control systems are robust tools, but alone, they provide an ineffective means of tracking and distributing finished goods software inventory for the following reasons:

- Version control systems are not easily searchable
- Version control systems lack robust metadata that allows developers to determine relevance
- Each development group typically has their own version control system(s), which makes enterprise-level reuse virtually impossible.
- Version control systems don't provide the level of consumer tracking and reporting necessary to support software reuse.

Version control systems and Metadata Repositories are robust tools that serve very different purposes. While they may work in conjunction with each other, each manages different portions of the organization's overall software inventory.

6.2 Packaging Best Practices

The Reusable Asset Specification (RAS) provides the following guidelines with respect to asset packaging.

- Every reusable asset must contain at a minimum one manifest file and at least one artifact, to be considered a valid reusable asset.
- An asset package is the collection of artifact files plus a manifest.

There are several asset packaging scenarios including:

6.2.1 Bundled As Single Archive File

This approach to packaging may be used in a team development environment. However, it also works well with less formal asset-based development processes as well as single user environments.

For this packaging approach, using a compression algorithm, all the files including the manifest file can be combined into a single archive file, making distribution of the asset easier.

6.2.2 Unbundled with artifacts in original location

Developing artifacts in a team environment generally includes the use of version control systems. One approach to defining assets is to add the asset manifest file to the SCM and point to the artifacts in their original location.

6.2.3 Unbundled with artifacts moved to new location

In many cases when artifacts are to be packaged within an asset, the artifacts require some modification to make them reusable. At this point the artifact may take on a new identity and be moved to a new location wherein it may be modified as necessary.

6.3 Backup and recovery

In order to protect investment, all software and configuration must be backed up in a form suitable for fast recovery. Backup and recovery processes must be established

and the required infrastructure must be planned and implemented. Assets must be backed up periodically for emergency recovery purposes and they should be recoverable to the last known best state.

7

Summary

Engineering infrastructure plays a key role in improving developer productivity and building robust, bullet-proof solutions. An effective engineering infrastructure must support an integrated asset-centric approach to engineering where assets are harvested and organized efficiently, yielding the following benefits

- Improve the agility of the business.
- Increase realized the business value
- Promote sharing and reuse
- Assist in implementing design-time governance
- Improve developer productivity
- Accelerate time-to-market

This document covers the capabilities and logical components that constitute an Oracle solution engineering environment and describes the deployment architecture. It also outlines the principles and best practices for engineering.

An integrated engineering environment is an essential part of the Oracle infrastructure and due care should be taken in planning and implementing an infrastructure that promotes rapid solution design and development aligned to the business objectives.

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 may be of interest:

A.1 Related Documents

ORA Monitoring and Management - This document provides a reference architecture for designing a management and monitoring framework to address the needs for the modern IT environment.

A.2 Other Resources and References

In addition, the following materials and sources of information relevant to SOA Infrastructure may be useful:

- Object Management Group Reusable Asset Specification - http://www.omg.org/technology/documents/formal/ras.htm
- OMG Systems Modeling Language (SysML) - http://syseng.omg.org/SysML.htm
- Wikipedia - http://en.wikipedia.org
- OMG XML Metadata Interchange (XMI) - http://www.omg.org/spec/XMI/2.1.1/
- Unified Modeling Language (UML) - http://www.uml.org/
- SOA Modeling Language (soaML) - http://www.omg.org/spec/SoaML/
- OMG MetaObject Facility (MOF) - http://www.omg.org/spec/MOF/2.0/
- Java Community Process (JCP) Java Specification Requests (JSR) - http://jcp.org/en/jsr/all
- Service Component Architecture (SCA) - http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications
- *Objects, Components, and Frameworks with UML: The Catalysis(SM) Approach* - Desmond Francis D'Souza, Alan Cameron Wills
- Best Practices for Building Production Quality EAR files - http://www.oracle.com/technology/pub/articles/dev2arch/2008/01/packaging-best-practices.html
- Java Platform Enterprise Edition - http://java.sun.com/javaee/

- Oracle Enterprise Repository documentation - http://download.oracle.com/docs/cd/E13164_01/oer/docs10134/index.html
- Oracle Application Testing Suite - http://www.oracle.com/technology/software/products/app-testing/index.html
- Oracle JDeveloper and ADF - http://www.oracle.com/technology/products/jdev/index.html

Glossary

Please refer to the *ORA Master Glossary* for a comprehensive list of glossary terms.

Extreme Programming (XP)

Extreme Programming (XP) is a software development methodology which is intended to improve software quality and responsiveness to changing customer requirements. As a type of agile software development, it advocates frequent releases in short development cycles (timeboxing), which is intended to improve productivity and introduce checkpoints where new customer requirements can be adopted.Scrum is an iterative incremental framework for managing complex work (such as new product development) commonly used with agile software development.

Scrum

Scrum is an iterative incremental framework for managing complex work (such as new product development) commonly used with agile software development.

Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovery and Integration (UDDI) is a platform-independent, XML based, open industry initiative, sponsored by the Organization for the Advancement of Structured Information Standards (OASIS), enabling businesses to publish service listings and discover each other and define how the services or software applications interact over the Internet.

Version Control System (VCS)

VCS manages changes to documents, programs, and other information stored as computer files. It is most commonly used in software development, where a team of people may change the same files.