Oracle® Reference Architecture

Service-Oriented Integration

Release 3.0

E14485-03

September 2010

ORACLE®

ORA Service-Oriented Integration, Release 3.0

E14485-03

Primary Author: Bob Hensle

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

Contributor: Ravi Sankaran

Contents

3 Integration Reference Architecture

4 Product Mapping

List of Figures

Send Us Your Comments

ORA Service-Oriented Integration, Release 3.0

E14485-03

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

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

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

Preface

This document describes an architecture for a service-oriented approach to integration of applications in an IT environment. The architecture is described in a product agnostic way since the architecture does not require any specific products; rather, it is based on Oracle best practices for service-oriented integration. The document describes the concept of service-oriented integration and how this differs from more traditional integration approaches. The principles that are essential to a service-oriented approach to integration are listed and the principles are linked to the layers and capabilities included in the architecture. A separate section of the document maps Oracle products onto the architecture to illustrate how to realize the architecture using Oracle products.

It should be noted that Service-Oriented Architecture (SOA) is a topic broader than service-oriented integration. Service-oriented integration is only one (albeit important) aspect of SOA. SOA delivers many benefits beyond just the benefits associated with integration.

Audience

This document is intended for Enterprise Architects and Solution Architects who want to understand service-oriented integration and how to create an architecture that supports a shared service approach to exposing existing applications. Some level of understanding of SOA is required since this document does not provide any SOA background or primer material.

How to Use This Document

This document is intended to be read from start to finish. However, each section is relatively self contained and could be read independently from the other sections. This document can be used to understand service-oriented integration or as a blueprint for creating a service-oriented integration architecture.

Document Structure

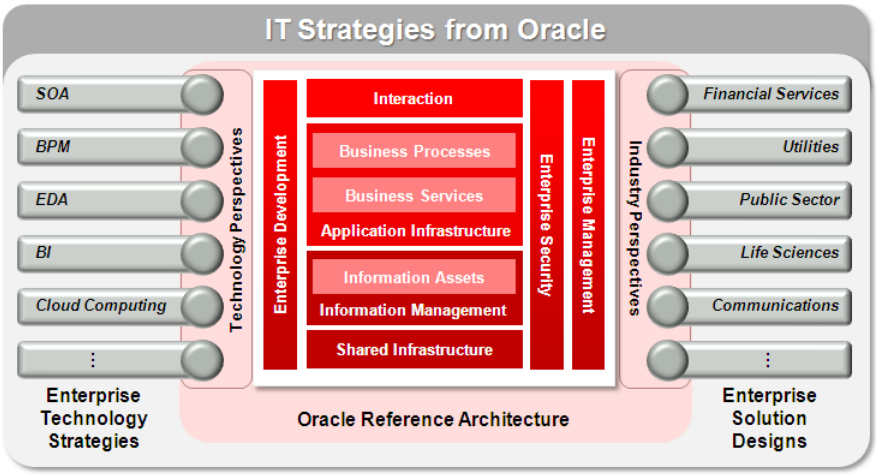
This document is organized in sections that first introduce and describe service-oriented integration and then describes an architecture that is based on and supports a service-oriented integration approach. Specifically,

- Chapter 1 provides a description of service-oriented integration and how this differs from more traditional integration approaches.
- Chapter 2 identifies the principles that should be met by any architecture that purports to support a service-oriented approach to integration.

- Chapter 3 describes the service-oriented integration architecture from various viewpoints.
- Chapter 4 maps Oracle products onto the architecture to illustrate how Oracle products can be used to realize the architecture.
- Chapter 5 describes various integration patterns and message exchange patterns that the architecture supports and will likely be used in integration scenarios.
- Chapter 6 is a brief summary of the document.
- Appendix A provides a list of additional resources that the reader may find informative and beneficial.

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 adoption and success. In addition, each ETS extends the Oracle Reference Architecture by adding the unique capabilities and components provided by that particular technology. It offers a horizontal technology-based perspective of ORA.
- **Enterprise Solution Designs (ESD)** are industry specific solution perspectives based on ORA. They define the high level business processes and functions, and

the software capabilities in an underlying technology infrastructure that are required to build enterprise-wide industry solutions. ESDs also map the relevant application and technology products against solutions to illustrate how capabilities in Oracle's complete integrated stack can best meet the business, technical and quality of service requirements within a particular industry.

This document is one of the series of documents that comprise Oracle Reference Architecture. ***ORAService-Oriented Integration*** examines the most popular and widely used forms of integration, putting them into perspective with current trends made possible by SOA standards and technologies. It offers guidance on how to integrate systems using service-oriented principles; thus bringing together modern techniques and legacy assets.

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.

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

1

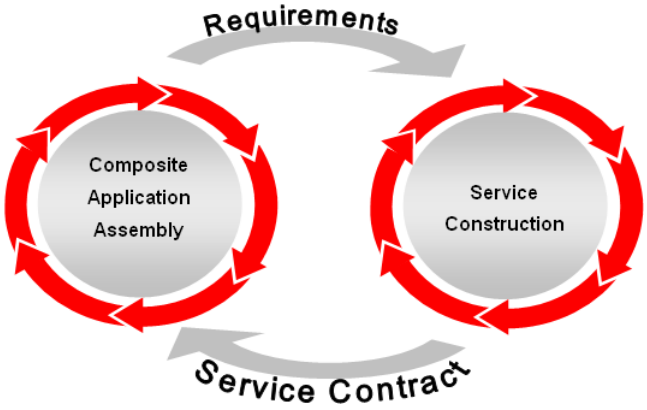
Service-Oriented Integration Overview

This section describes the core concepts of service-oriented integration and explains why service-oriented integration is different from how integration has traditionally been done.

1.1 Service-Oriented Integration Defined

The primary goal of service-oriented integration is to better leverage existing systems within the IT environment by applying service-oriented principles. Ultimately, the goal is to enable the assembly of composite applications, with little or no custom coding, that include capabilities sourced from existing systems. **Composite applications** are applications that pull together data, functionality, and process from multiple existing sources to solve a business problem or create new business value. Service-oriented integration is the mechanism to expose existing sources of data, functionality, and process so that those sources can be readily consumed by a composite application. These two related but distinct efforts are depicted in Figure 1–1.

Figure 1–1 Composite Applications and SOA Services



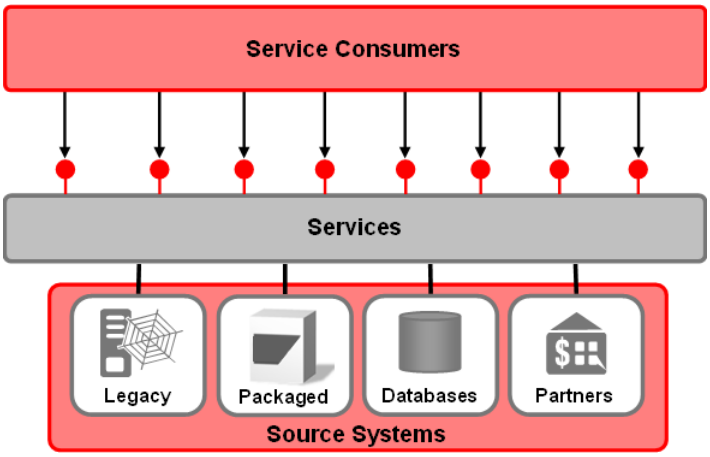
Service construction includes creating entirely new SOA Services and also exposing existing assets as SOA Services. This document focuses on the latter - exposing existing assets as SOA Services.

1.1.1 Services

It must be noted that creating a **SOA Service** from existing assets generally requires a good deal more than just adding a standards-based interface, i.e. simply service enabling existing assets is insufficient. The SOA Service needs to expose process, functionality, and data that is usable in a broader context than the source of the

capability was designed to meet. Therefore, creating a SOA Service usually entails some amount of aggregation, transformation, or expansion of existing capabilities provided by the source systems. This requires a SOA Services layer between the existing assets and the consumers as illustrated in Figure 1–2.

Figure 1–2 Services



The very essence of service-oriented integration is the building of a catalogue of SOA Services that expose, in a business-enabling way, the data, functionality, and process contained in existing systems. In order to reap the benefits of service-oriented integration, the SOA Services must decouple the consumers from the source systems without creating tight coupling between the consumer and the SOA Service itself. Additionally, the SOA Services must expose functionality that is aligned with the business needs of the service consumers, not just provide an API or data-level pass through to the source systems.

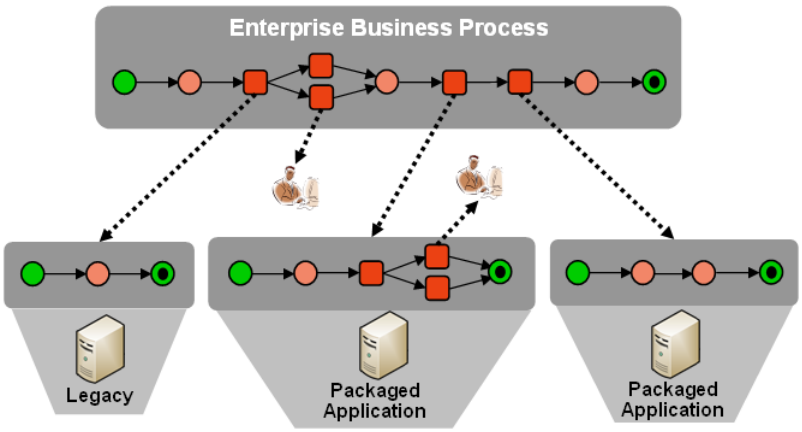
Successfully building a catalog of SOA Services requires a sound service engineering process that incorporates design and architectural rigor. This document focuses on the architecture required to realize such a catalogue of SOA Services, not on the service engineering process. However, there are aspects of the engineering process (e.g. **service contracts**, service versioning) that impart requirements on the architecture (see ***Software Engineering in a SOA Environment***).

There are three types of capabilities that existing source systems can provide to composite application construction: process, functionality, and data. Although the approach for creating SOA Services based on existing process, functionality, or data is similar, these different capabilities also provide unique challenges when creating SOA Services.

1.1.2 Process Integration

Frequently the existing source systems include business process or workflow built into the system. From the perspective of the source system, this is a complete business process. However, from the broader context of the organization, the built-in business process is actually only a portion of a larger business process that spans multiple back-end systems. Therefore, when creating the catalog of SOA Services these built-in business processes should be viewed as sub-processes that are part of the larger enterprise business process. This relationship is illustrated in Figure 1–3.

Figure 1–3 Enterprise Business Process



The enterprise business process as well as the built-in business processes may be fully automated or might include human interactions as illustrated in Figure 1–3.

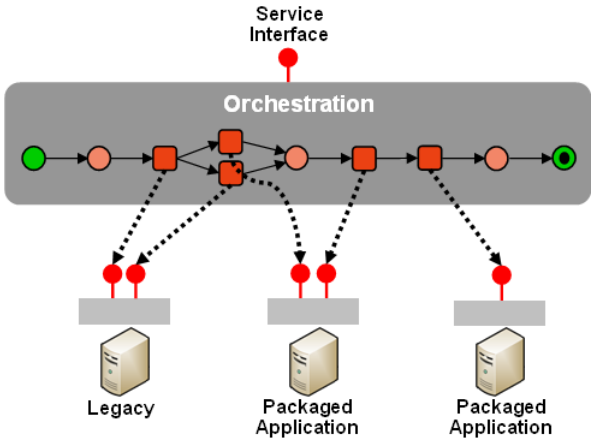
1.1.3 Functional Integration

The functional capabilities contained in existing source systems is usually too fine grained and domain specific to be useful, as is, for creating composite applications. The application hosting the functionality was usually constructed to provide users fine-grained control. Frequently the users of the system attend special training to become proficient with the application. Additionally, the application invariably contains far more functionality than is necessary in a broader context. Most of the functionality is specific to the domain (e.g. HR, Finance, Shipping) and has little relevance in an "enterprise" context.

Therefore, exposing existing functionality as SOA Services usually requires abstracting the functionality to a higher level so that it has meaning (and is more useful) in a broader context. Frequently this entails combining several fine-grained operations into a more course-grained operation.

Sometimes the fine-grained operations are associated with a built-in workflow exposed through the user interface, i.e. a set of prescribed operations that the end user performs in sequence. In this instance, when creating a SOA Service it is usually desirable to have a single operation on the SOA Service that encapsulates and hides the built-in workflow by performing the individual steps automatically. This type of orchestration is illustrated in Figure 1–4.

Figure 1–4 Orchestration



As illustrated in Figure 1–4, the orchestration may even span two or more existing applications. This orchestration of fine-grained operations to create a coarser-grained operation is a common technique to abstract functionality.

The input and output data specifications for operations on existing applications also tend to be application specific and not particularly amenable to a broader audience. Therefore, the SOA Service frequently exposes an interface that is significantly different than the underlying application operations expose. This data format issue is covered more in the next section.

1.1.4 Data Integration

Each existing application contains its own data model and data formats. This proliferation of data models and data formats is exacerbated by the fact that a single enterprise entity (e.g. customer, product, order) frequently has data elements stored in multiple existing applications. To be successful at exposing existing data via SOA Services, the integration approach must manage this complexity.

Master Data Management (MDM) is one approach to manage this complexity. MDM is a large topic unto itself, and is beyond the scope of this document. However, MDM is complementary to service-oriented integration, and if MDM has been deployed to manage data complexity, then it can be easily used by the SOA Services. MDM is beneficial to, but not required for, successful service-oriented integration.

The primary goal of a SOA Service that exposes an enterprise data entity is to make it easier to work with the entity and hide the complexity of how that entity is stored in existing source systems. To achieve this goal, the SOA Service needs to incorporate the following capabilities:

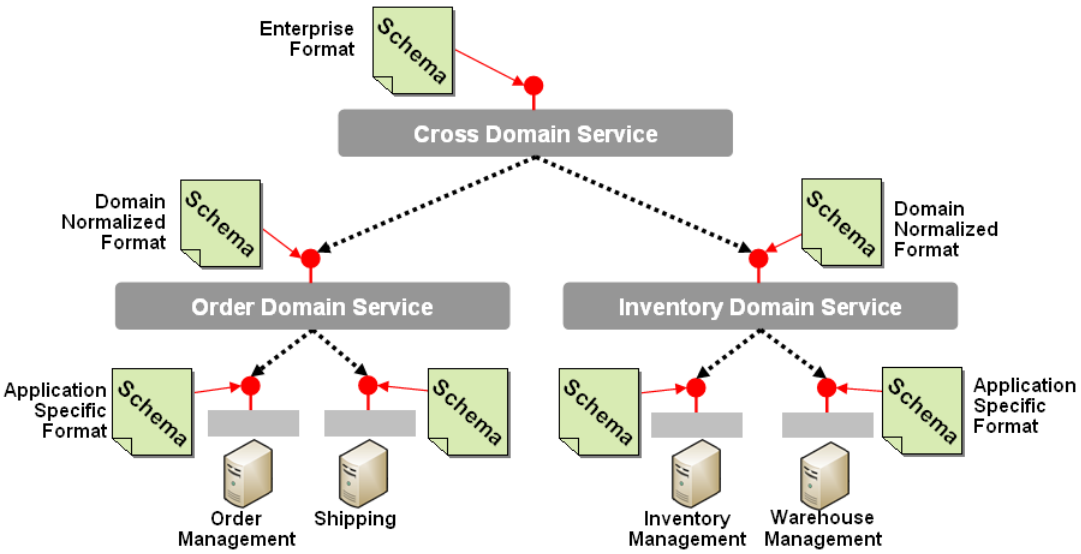
- Consumer Representation - The SOA Service should present an interface to the entity than is appropriate for the consumer of the data. A single SOA Service might provide multiple representations of a singe entity, each representation appropriate for a different type of consumer.
- Aggregation - The SOA Service may need to pull data elements from multiple source systems to construct a representation for a particular consumer.
- Synchronization - An update to an entity may require updating data elements in more than one source system. The SOA Service needs to ensure that all updates are done properly so the entity maintains consistency.

Providing consumer representations and reading from and writing to multiple source systems leads to the issue of data format transformations. For a very small number of source systems, point-to-point transformations can be used by the SOA Services. However, this approach becomes untenable as the number of source systems increases. Thus, a better approach is to create a normalized format for the data entities and then provide transformations to and from the normalized format for each source system.

> **Note:** The data normalization described in this document should not be confused with database normalization. Database normalization (e.g. third normal form) is a specific type of data normalization that is used in relational databases. Data normalization in this document is more generic in nature and is used to bring disparate data formats from multiple source system into a common (and hence more usable) format.

It should be noted that a single canonical data model for the entire enterprise is not required to successfully employ normalized data formats. Rather a federated approach to normalization can be used. For example, in a large enterprise each functional domain could create a normalized format. Transformations between the domain formats would then be created for SOA Services that span domains. This approach is illustrated in Figure 1–5.

Figure 1–5 Federated Data Normalization



Another benefit of creating SOA Services that expose enterprise data entities is that the storage of enterprise data can usually be significantly simplified once the SOA Services are deployed. Providing a single source of enterprise data simplifies access and eliminates the need for copying of data into operational data stores. Since access to the enterprise data is via the SOA Services and consumers are insulated from changes to the underlying source systems, a storage rationalization effort can be undertaken to reduce and simplify the number of source systems.

1.1.5 Bottom-Up Service Enablement

Bottom-up service identification is a process that looks at existing applications in an attempt to identify SOA Services. From the previous sections it should be apparent

why a strictly bottom-up approach to service enabling existing applications is unlikely to produce desirable results. In the absence of consumer requirements, it is highly unlikely that a SOA Service will be created that is in fact usable much less reusable. It is much more likely that the service enablement effort will be a waste of time and resources.

Top-down service identification analyzes business processes and employs business function decomposition to help identify SOA Services. Service enablement should always include a top-down approach to elicit consumer requirements and then a bottom-up approach can be used to identify existing sources that can be used to meet the requirements.

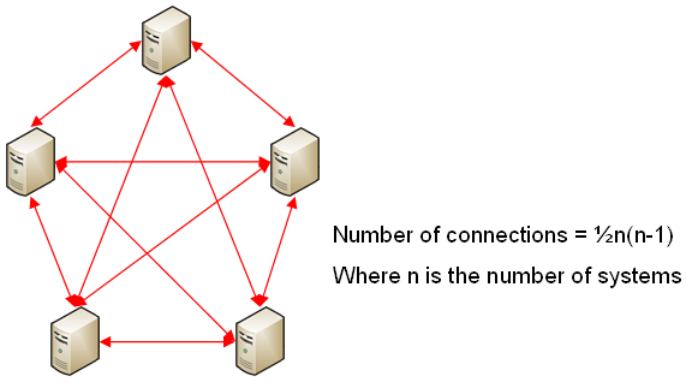
1.2 Difference from Traditional Integration

Service-oriented integration is a new approach to a problem that has existed in IT for decades. This section compares and contrasts service-oriented integration with integration approaches that have been, and in fact still are being, used to integrate disparate systems.

1.2.1 Point-to-Point

Point-to-point integration is when one application connects directly to another application. The technology to facilitate the connection may be a third-party technology or product, but the result is the same - the applications are tightly linked together. The consumer application must know all the details of the source application for the integration to be successful. This results in a very brittle and difficult to maintain architecture. And as the number of systems increases, the number of point-to-point integrations grows rapidly. Despite the significant downsides, point-to-point integration is very common primarily because it is the fastest way to integrate two systems. It is a classic "penny wise, pound foolish" approach to integration. Figure 1–6 illustrates point-to-point integrations.

Figure 1–6 Point-to-Point Integration



Service-oriented integration done correctly avoids the brittleness of a point-to-point integration architecture. In service-oriented integration the consumer is decoupled from the source system via the SOA Service that encapsulates and abstracts the source systems behind a service interface. Consumers of the SOA Service should need no details about the underlying source system. They should base the service usage solely on the contract provided by the SOA Service.

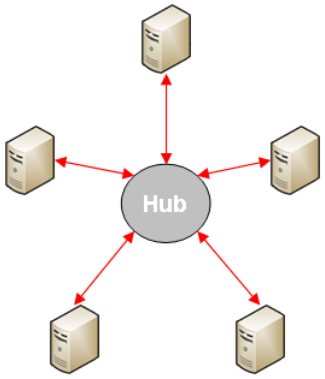
Unfortunately, simply adding a wrapper (e.g. **Web service**) to an existing system does not eliminate point-to-point integration. In fact point-to-point integration can be

accomplished quite nicely using Web service interfaces. To be truly service-oriented, the SOA Services created must be well designed and constructed so the source system details do not bleed through. Service infrastructure that provides capabilities such as service discovery and routing is also essential to decouple service providers and consumers.

1.2.2 Enterprise Application Integration

Enterprise Application Integration (EAI) is an approach for integrating multiple applications. EAI products are built around messaging products and are deployed in either a hub-and-spoke architecture or in a bus architecture. The hub-and-spoke form of an EAI architecture is illustrated in Figure 1–7.

Figure 1–7 Hub and Spoke EAI Architecture



Some argue that service-oriented integration is actually a form EAI. This is not correct. EAI is an application-oriented architecture. EAI provides the mechanism to have applications interact to share data and functionality. Service-oriented integration adds the concept (and concrete deployment) of SOA Services that are separate and distinct, with a lifecycle that is independent, from any application in the computing environment.

Another major difference between traditional EAI and service-oriented architecture is the need for a common (proprietary) message infrastructure when using EAI. This is not a required artifact of EAI, but rather an implementation detail that was true for all major EAI products. In order to participate in the integration, each system needed to incorporate the message infrastructure client.

This was required because the communication protocol used by the message infrastructure was unique to each EAI product. [For this discussion, the communication protocol includes the connection protocol (e.g. TCP/IP) as well as the message formats and message exchange patterns.] For example, there was no way for a TIBCO client to send a message on a SeeBeyond message bus, or vice versa. Even when the message programming interface was standardized (e.g. **JMS**) the communication protocol was still unique and proprietary. A JMS client could only enqueue or dequeue a message from a queue provided by the same vendor as provided the client. This led to vendor lock-in.

Service-oriented integration should be based on standardized communication protocols such as Web services or **REST**. Service-oriented integration can also use proprietary communication protocols (e.g. MQSeries, TIBCO, JMS) whenever appropriate. In fact, traditional EAI products can be easily incorporated into a service-oriented integration approach. The EAI product can provide connectivity to source applications which is especially helpful for legacy systems.

1.2.3 Batch

Batch processing of data is another approach that is frequently used to integrate applications. There are two main types of batch integration. The first is where changes to a system are collected over a period of time by writing the change to some persistent storage (e.g. file). The changes are then transferred to another system and those changes are loaded in the other system. The primary disadvantage of this type of integration is the time lag for the data between the two systems.

The second type of batch integration is when large amounts of data are extracted from one system and then loaded into another system. This type of batch integration is frequently used for data warehouses. This type of batch integration is not easily replaced by a real-time connection between the systems. The amount of data being transported and the compute resources used to extract, transform, and then load the data is significant.

Service-oriented integration can be used to eliminate the first type of batch integration. The primary advantage is that there is no longer a time lag between the two systems with respect to the data changes. The second type of batch integration is best left as a batch process. From an integration perspective, it is desirable to minimize the number of batch integrations that are being performed, but when the business requirements dictate the movement of large amounts of data, batch processing is the best way to meet those requirements.

1.3 Complementary Technologies

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

1.3.1 Web Interface

This category includes a variety of web-based interface technologies including portals, mash-ups, JSON, Atom Feeds, etc. The Web interface provides "integration at the glass" capabilities for the end user. While very important from an end user perspective, this type of integration is not a core topic for this document. In the context of this document, these technologies are consumers of the SOA Services created as part of a service-oriented integration effort.

It is important that the service-oriented integration architecture support construction and deployment of SOA Services that expose data and functionality in a manner amenable to end user manipulation. Thus, these Web technologies levy requirements on the architecture to ensure that these types of service consumers are fully supported.

1.3.2 BPM

Business Process Management (BPM) technologies that can be used within, or independent of, a service-oriented architecture. Many BPM projects have been successfully deployed without including the use of any SOA Services. Likewise, SOA has also been successfully applied to solutions where no BPM exists.

However, combining BPM with SOA produces an architecture that delivers more than the sum of the parts. Using BPM as a business process execution layer over a layer of SOA Services delivers easily measurable business value and demonstrates the value of SOA Services. The SOA Services make it much easier to automate business processes and facilitate business process improvements and streamlining.

Thus, although BMP technologies are not dependent on SOA and SOA is not dependant on BPM, the synergies are so great that it is vital to describe how the two interact in a service-oriented integration architecture. Therefore, this document includes BPM in the architecture described below.

1.3.3 Event Processing

Far too often integration is viewed as one-way i.e. the only interaction is client systems making requests to backend systems that then respond to the requests. Most modern applications that will serve as source systems in an integration scenario are far more complex and capable than this one-way scenario encompasses.

Sure, the source systems will respond to requests, but it is also likely that the source systems can (and should) initiate actions by raising events. Thus, a complete service-oriented integration architecture must include the ability for source systems to raise events, and the architecture must provide a mechanism to properly handle the events i.e. event processing. Therefore, standard event processing is covered in this document.

Complex Event Processing (CEP) is an extreme case of event processing. True CEP requires specialized infrastructure to handle complicated event correlation and high throughput and low latency requirements. CEP is not covered in this document.

1.3.4 Master Data Management

Master Data Management (MDM) is complementary to service-oriented integration. If an MDM solution has been deployed to manage data complexity, then the master data can be easily leveraged by SOA Services developed as part of an integration effort. Thus, MDM does facilitate data integration aspects of a service-oriented integration effort.

MDM has little if any impact on the functional integration aspects of a service-oriented integration approach. Therefore, MDM is certainly not a replacement for service-oriented integration. In fact, MDM is not required for successful service-oriented integration. And, an MDM program can progress in parallel with a service-oriented integration program.

In the context of this document, any MDM solution is considered a source system that could be leveraged as part of the integration effort.

1.3.5 B2B

B2B (Business to Business) refers to a set of technologies that have been used to automate interactions between companies that are business partners. Using B2B requires that business partners agree to specific messages formats and to specific message exchange patterns. The message formats and message exchange patterns are usually defined by an industry group (e.g. RossettaNet, HR-XML Org, IATA).

B2B specific technologies are rarely used for integrations within an organization. Since the primary focus for service-oriented integration is internal to a company, B2B technologies usually do not apply. However, there are some potential touch points with B2B and internal integration.

- Sometimes a driving force for internal integration is the desire to establish B2B integration. In this case, the B2B effort levies requirements on the internal integration effort. Thus, the SOA Services created as part of the integration effort must be designed to meet the needs of a B2B client application.

- Since the B2B message formats are defined by industry standards groups, these message format standards might be used as the basis for internal message formats as well. Using industry standards for message formats will also facilitate any future B2B efforts that might arise.
- An exiting partner system may be a source system for the integration effort. This scenario is becoming more common as Software-as-a-Service and Cloud Computing become more widespread.
- A robust service-oriented integration architecture successfully deployed within an enterprise readily facilitates external integration as well. A thin façade can be added to expose selected SOA Services to external partners.
- Following B2B standards for internal SOA Service definitions can facilitate a mergers and acquisition strategy. When a merger or acquisition take place, a B2B approach can be used to rapidly integrate the two companies.

1.3.6 Identity Management

Identity Management (IdM) can be viewed as the integration of multiple identities and identity stores within an enterprise. Thus, IdM is a kind of integration. However, IdM is focused on identities, not business data and business functionality. Business data and business functionality integration is the focus for this document, hence IdM is not in scope.

Service-oriented integration can leverage IdM solutions where they exist. IdM provides a single source for authentication and authorization that can significantly simplify security concerns in the integration effort. In the context of this document, IdM solutions are considered source systems for security capabilities.

1.3.7 Service Infrastructure

Service infrastructure provides many of the key capabilities within a service-oriented integration architecture. These key capabilities are specifically called out in the architecture described below. This document does not attempt to describe service infrastructure in general nor the capabilities that service infrastructure could or should provide beyond those capabilities directly related and key to integration. See the ***ORA SOA Infrastructure*** document for more details on service infrastructure.

2

Service-Oriented Integration Principles

This section defines the principles of service-oriented integration. These principles provide the guidance for creating a sound integration architecture that will enable service-oriented integration. The principles are:

2.1 Logical Data Representations

Principle	**Logical Data Representations**
Statement	Message and data formats should be based on logical representations of business objects rather than native application data structures.
Rationale	The primary purpose of service-oriented integration is to facilitate the creation of composite applications. Propagating application specific data formats makes composition more difficult. For further discussion see Section 1.1.4.
Implications	▪ Logical representations of the business entities which are based on the actual usage of the entities need to be created. ▪ The architecture must provide the capability to construct and use a logical data model. See Section 3.1.4

2.2 Normalized Data Formats

Principle	**Normalized Data Formats**
Statement	Data transformations are to and from normalized formats.
Rationale	Normalized data formats facilitate composition and reduce the number of transformations that must be created and maintained. A canonical data representation that spans the enterprise can be used but is not required. A federated approach to data normalization is also possible. For further discussion see Section 1.1.4.
Implications	▪ If they do not currently exist, normalized data formats must be created. ▪ The architecture must provide the capability to transform data from one format to another. See Section 3.1.4. ▪ The architecture must support data federation specifically with respect to deployment of data services. See Section 3.4.2.

2.3 Avoid Point-to-Point Integrations

Principle	**Avoid Point-to-Point Integrations**
Statement	Point-to-point integrations are to be avoided.
Rationale	Point-to-point integrations are brittle, inflexible, and expensive to maintain. There are cases where point-to-point integrations are required but these should be handled as exception cases. Example exceptions include performance requirements that can only be met using point-to-point connections and when large amounts of data must be moved. For further discussion see Section 1.2.1.
Implications	■ Existing point-to-point integrations should be decommissioned. ■ The architecture needs to provide a means to de-couple participants in the integration. See Section 3.1.7 and Section 3.3.1. ■ The architecture must provide an mechanism to move large amounts of data where necessary. See Section 3.1.1.

2.4 Technical Orchestrations

Principle	**Technical Orchestration**
Statement	Technical orchestration is separated from business processes.
Rationale	Making a clear distinction between technical aspects and business aspects facilitates maintenance of both. Technical aspects change when the underlying systems change whereas business aspects change when the business changes. For further discussion see Section 1.1.2 and Section 1.1.3
Implications	■ Guidelines need to be established to ensure that the necessary separation is created and maintained. See Section 3.2.2. ■ The architecture must support both technical orchestrations and business processes. See Section 3.1.5 and Section 3.1.6.

2.5 Business-Level Services

Principle	**Business-Level Services**
Statement	SOA Services expose business-level functionality and data.
Rationale	Business-level functionality and data are required to create business valuable composite applications. There may also be lower-level SOA Services that are reused by the business-level SOA Services. For further discussion see Section 1.1.5.
Implications	■ A service engineering process that identifies appropriate SOA Services needs to be established. ■ The SOA Services must be constructed respecting the layers of the architecture. See Section 3.2.1. ■ The architecture must support business-level SOA Services. See Section 3.1.5.

2.6 Service Contract

Principle	**Service Contract**
Statement	SOA Services include a contract that specifies the functional and non-functional capabilities provided.
Rationale	In order to support business-level composition, the SOA Service must have a contract that is understandable to a business person. For further discussion see Section 1.1.
Implications	■ A service engineering process that enforces creation of the service contract must be established. ■ The architecture must support discovery of existing SOA Services based on the service contracts. See Section 3.1.7. ■ Some of the capabilities provided by the SOA Service and documented in the service contract may be fulfilled by configuration of infrastructure. See Section 3.3.1.

2.7 Minimal Application Modifications

Principle	**Minimal Application Modifications**
Statement	Extensive, intrusive modifications to existing applications should be avoided.
Rationale	Some minor modifications may be required to support connectivity, but requiring extensive modifications defeats a major reason for integration. For further discussion see Section 1.1.1.
Implications	■ SOA Services provide a layer over existing applications where new functionality can be hosted; thereby reducing the need to modify existing applications. ■ The architecture must provide a mechanism to connect to existing applications without requiring extensive modifications to the applications. See Section 3.1.3.

2.8 Access Via Services

Principle	**Access Via Services**
Statement	Enterprise data and functions should be accessed through SOA Services.
Rationale	SOA Services create the foundation for agile composite application development. Failing to leverage SOA Services defeats the primary goal of service-oriented integration. For further discussion see Section 1.1.1
Implications	■ Governance that enforces the usage of SOA Services to access enterprise data and functionality must be established. ■ The architecture must provide a mechanism to access enterprise data via SOA Services. See Section 3.1.4. ■ The architecture must provide a mechanism to access enterprise functions via SOA Services. See Section 3.1.5.

2.9 Opaque Service Implementations

Principle	**Opaque Service Implementations**
Statement	The implementation of a SOA Service is opaque to all service consumers.
Rationale	Service consumers must be able to successfully call a SOA Service without needing to understand the internal workings of the SOA Services. For further discussion see Section 1.1.1
Implications	■ The service interface must be constructed such that implementation details do not propagated through the interface. ■ The architecture must support opaque service implementations. See Section 3.1.7.

2.10 Platform Independence

Principle	**Platform Independence**
Statement	The service consumer platform is independent from the service provider platform.
Rationale	SOA Services need to support any kind of service consumer regardless of the particular platform that the consumer is built upon. For further discussion see Section 1.1.1
Implications	■ SOA Services must be constructed so that platform dependencies are not introduced. ■ The architecture must support service consumers hosted on platforms different from the service providers. See Section 3.1.7.

2.11 Location Transparency

Principle	**Location Transparency**
Statement	The location of a service provider is unimportant to the service consumer and vise versa.
Rationale	Location transparency helps provide the de-coupling of service consumers and providers necessary for extensive SOA Service reuse. For further discussion see Section 1.1.1
Implications	■ Service providers should make no assumption about the location of the service consumer. ■ Service consumers should make no assumption about the location of the service provider. ■ The architecture must support the ability for a service consumer to call a service provider regardless of the location of either. See Section 3.1.7.

2.12 Concurrent Service Versions

Principle	**Concurrent Service Versions**
Statement	There may be multiple versions of a SOA Service in production concurrently.

Rationale	Invariably a SOA Service will require modifications to support new consumers or to expand functionality. Supporting concurrent versions of a SOA Service is essential for a sound service versioning approach. For further discussion see Section 1.1.1
Implications	■ A service versioning strategy needs to be established. ■ The architecture must support multiple, concurrent versions of a SOA Service. See Section 3.1.7.

2.13 Graceful Service Migration

Principle	**Graceful Service Migration**
Statement	Service consumers are able to migrate to a newer version of a SOA Service gracefully.
Rationale	Service consumers should migrate to a new version of a SOA Service as part of a normal maintenance process. The coordinated deployment of service consumers and service providers should not be necessary. For further discussion see Section 1.1.1
Implications	■ A service migration strategy needs to be established. ■ The architecture must support graceful service migration. See Section 3.1.7.

2.14 Event Processing

Principle	**Event Processing**
Statement	**Applications included in the integration can raise events which must be processed consistently**.
Rationale	Events allow the applications that are included in the integration to notify other applications that something important has occurred. The events that are raised by applications must not be ignored and the architecture should handle the events in a consistent manner. Each event should trigger the appropriate response. For further discussion see Section 1.3.3
Implications	■ The architecture must provide a mechanism for applications to raise events. See Section 3.1.3. ■ The architecture must provide a mechanism to process events consistently regardless of the type of event. See Section 3.3.3.

The above principles provide sound guidance for creating a service-oriented architecture for integration. Each organization embracing service-oriented integration should evaluate the principles listed above and derive their own set of principles that match the specific environment and goals of their organization.

3

Integration Reference Architecture

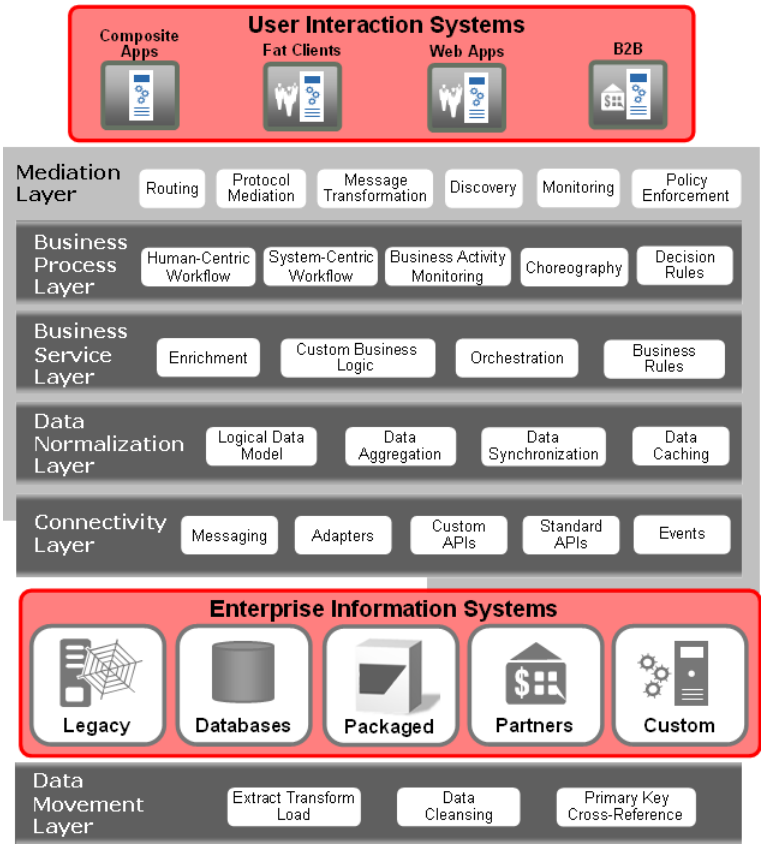
This section describes a high-level architecture for service-oriented integration. Each section provides a view of the architecture from a particular perspective.

3.1 Logical View

The Logical View of the architecture describes the various layers in the architecture. Each layer encapsulates specific capabilities for the overall architecture. Upper layers in the architecture leverage the capabilities provided by the lower layers. Generally, upper layers call lower layers in the architecture and the reverse (i.e. lower levels calling upper layers) is prohibited. There may be some special cases that are exceptions to this rule (e.g. Section 3.3.3). Upper layers are allowed to call capabilities provided by any lower layer and, therefore, may skip any intermediate layers.

The layers in the architecture are shown in Figure 3–1.

Figure 3–1 Integration Architecture Logical View



Each layer in the architecture beginning at the bottom and progressing to the top is described in subsequent sections. The description of each layer does not attempt to enumerate every possible capability provided by the layer; rather, only the key (essential) capabilities provided by the layer are listed and described.

3.1.1 Data Movement Layer

The Data Movement Layer provides the batch and bulk data handling for the architecture. This layer exists primarily to offload bulk data movement from the upper layers in the architecture. Bulk data movement is a necessary evil in many enterprises, and therefore, the architecture must provide a mechanism to provide this capability in an efficient, controlled manner. Without this layer, the other layers in the architecture might be misused to move large blocks of data, a task for which the other layers are ill-suited.

The key capabilities encapsulated by this layer include the following:

- **Extract-Transform-Load** - The ETL/ELT capability provides the bulk data movement from one persistent store to another.
- **Data Cleansing** - Data cleansing ensures the quality of the data that is being moved in bulk. Data cleansing can also be applied to ensure quality and consistency on individual data updates.
- **Primary-Key Cross-Reference** - Each persistent data store is likely to have its own unique primary key scheme. This layer provides the cross-reference between these primary key schemes so that an identity can be maintained across disparate data stores.

The Data Movement Layer is shown below the Enterprise Information Systems since the bulk data movement is done "behind the scenes" and this processing is primarily invisible to the upper layers of the architecture. However, the results achieved by the Data Movement Layer are clearly valuable to the upper layers and both the quality data and the primary-key cross-reference are leveraged by the upper layers in the architecture wherever appropriate.

In some IT departments bulk data movement is used to create online aggregate data views by pulling data from multiple systems together into a single data store. Although this architecture does not prohibit this, preferred approach is to create enterprise data entities in the Data Normalization Layer (Section 3.1.4) that aggregate data across source systems without the need to persistently store copies of data.

3.1.2 Enterprise Information Systems

An **Enterprise Information System** (EIS) is any system that provides data and/or functionality that is exposed by the integration architecture. Examples of EIS include packaged applications, legacy systems, databases, partner systems, etc. An EIS may also be a service consumer. As SOA becomes more prevelent, it will become more common for EISs to participate nativily in service-oriented integration.

This is not formally a layer in the integration architecture. Rather the entire purpose of the architecture is to expose the data and functionality that is contained in the various Enterprise Information Systems to provide business value beyond what is already provided by the systems themselves.

3.1.3 Connectivity Layer

The Connectivity Layer provides access to the Enterprise Information Systems. There are a variety of technologies and products that can be leveraged to connect to existing EISs. Generally, both synchronous and asynchronous techniques are employed by this layer to provide connectivity. Both proprietary and standards-based technologies and products are used.

The key capabilities provided by this layer include:

- **Messaging** - Messaging provides asynchronous connectivity to EISs. Many traditional integration products (e.g. MQSeries, TIBCO) are based on messaging systems. Enterprises have successfully used these products to connect to EISs and this existing connectivity can be leveraged by this architecture.
- **Adapters** - Adapters (e.g. JCA) provide a standardized approach for connecting to EISs.
- **Standard APIs** - The EIS may provide access via one or more standard application programming interface (e.g. JDBC, RMI, WS*). The connectivity layer includes these standard interfaces natively; thereby allowing easy access to any EIS that exposes a standard API.
- **Custom APIs** - Some EISs (especially legacy) only provide access via a custom application programming interface. The connectivity layer provides the capability to call these custom APIs and wrap them with a standardized interface.
- **Events** - Events allow the EISs to initiate actions. The connectivity layer provides the mechanism so that an EIS can raise an event that is handled by the Mediation Layer just like any other message.

The primary purpose of this layer in the architecture is to encapsulate and hide the complexity of connecting to back-end systems. This allows the upper layers in the

architecture to treat the EISs in a more uniform and generic fashion; thus providing greater reusability and portability across back-end systems.

3.1.4 Data Normalization Layer

The Data Normalization Layer provides a standardized format for data entities. Each EIS stores data in its own (usually proprietary) format. This layer transforms the data to a form that is readily consumable by the upper layers in the architecture.

The key capabilities provided by this layer include:

- **Logical Data Model** - The logical data model provides an "enterprise" view of data entities. The logical data model for an enterprise frequently incorporates industry standard data formats (e.g. HR-XML, HL7, IATA).
- **Data Aggregation** - In order to construct a complete view of an enterprise data entity, it is frequently necessary to aggregate data from several sources. This layer includes this ability to aggregate data from multiple sources to provide a complete enterprise data entity.
- **Data Synchronization** - Data synchronization ensures that all data entities remain consistent regardless of where an update or change is made. Changes are propagated to all copies and views in an orderly manner.
- **Data Caching** - Data caching allows the computational expense of aggregation and transformation to be shared across more than one service request.

The primary purpose of this layer in the architecture is to encapsulate and hide the complexity of the data models and formats used by the back-end systems. This allows the upper layers in the architecture to operate on data entities that match the needs of the business rather than operating on data that match the storage approach of the back-end systems. The data normalization provided by this layer should not be confused with database normalization (see Section 1.1.4).

3.1.5 Business Service Layer

The Business Service Layer exposes data and functionality that is understandable and has meaning in a business context. This layer uses the lower layers to create SOA Services that a business person would understand.

The key capabilities provided by this layer include:

- **Enrichment** - Enrichment is the process of adding data elements to a data entity to give the entity increased information (and hence value) in a business context.
- **Orchestration** - Orchestration is used to combine multiple lower-level operations into a Business Service that hides the complexity of the lower-level operations.
- **Custom Business Logic** - Custom business logic is hosted in this layer of the architecture. Custom business logic frequently leverages lower-level layers to provide some of the exposed data and functionality.
- **Business Rules** - Business rules describe the operations, definitions, and constraints that apply to an organization. Essentially they are if-then statements that define or constrain some aspect of the business. Although business rules are frequently captured in custom code, it is far better to extract these business rules into explicit rule sets so that they can be more easily understood, modified, and maintained.

The primary purpose of this layer in the architecture is to provide an interface that matches the business. This layer also provides the layer of innovation over the existing

back-end systems. Custom business logic and rules can be implemented that use existing systems yet provide capabilities that extend beyond the capabilities provided by the back-end systems.

3.1.6 Business Process Layer

The Business Process Layer automates business processes. This layer uses the lower layers, especially the Business Service Layer, to create and automate business processes. The business processes generally span multiple Enterprise Information Systems.

The key capabilities in this layer include:

- **Human-Centric Workflow** - Human-centric workflow is a business process that is primarily or entirely composed of human tasks. There may be some relatively small amount of system interaction to support the human tasks.
- **System-Centric Workflow** - System-centric workflow is a business process that is primarily composed of system-to-system activities. There may be some human tasks in the business process or humans may be required to handle the exception cases in an otherwise entirely system-to-system process.
- **Choreography** - Choreography defines the messages that flow back and forth between systems that are participating in a business processes. An example of choreography would be a RossettaNet Partner Interface Process (PIP).
- **Business Activity Monitoring** - Business activity monitoring (BAM) provides visibility into business processes. The BAM information is exposed in dashboards that are used to measure and improve business processes.
- **Decision Rules** - Most business processes include decision points where branching is done based on some criteria. Decision rules are the encapsulation of these decision criteria into a rule that is then more easily modified and maintained.

The primary purpose of this layer in the architecture is to define and automate the business processes external to, and independent of, the specific backend systems used in the organization. This isolates the business process from backend system changes, and conversely, isolates the backend systems from business process changes. De-coupling the business processes from the backend systems simplifies changes and maintenance for business processes and backend systems.

This layer generally provides the greatest and most measurable business value.

3.1.7 Mediation Layer

The Mediation Layer provides loose coupling for the entire architecture. It decouples the layers of the architecture as well as decoupling external users of the layers from the specific layers in the architecture.

The key capabilities in this layer include:

- **Routing** - Routing provides the ability to send the client request to the appropriate provider based on some criteria. The routing may even include sending the client request to multiple providers. This capability facilitates location transparency, versioning, scalability, partitioning, request pipelining, SLA management, etc.
- **Protocol Mediation** - Protocol mediation is the ability to handle a client request using one protocol (e.g. WS*, JMS, REST) with a provider using a different protocol. This provides protocol decoupling between the provider and the consumer.

- **Message Transformation** - Message transformation allows a client request using one message format to be handled by a provider that expects a different message format. This provides message format decoupling between the provider and the consumer.
- **Discovery** - Discovery is the mechanism by which a client finds a provider of a particular SOA Service. Discovery can occur at design time or runtime.
- **Monitoring** - Monitoring captures runtime information about the messages flowing through the mediation layer. Since the mediation layer is an intermediary for message traffic, it provides a centralized monitoring capability.
- **Policy Enforcement** - Policy enforcement provides consistent application of policies (e.g. WS-SecurityPolicy) across all messages flowing through the mediation layer. Since the mediation layer is an intermediary for message traffic, it provides a centralized policy enforcement capability.

The primary purpose of this layer in the architecture is to facilitate communication between layers in the architecture and between this architecture and the systems that connect to this architecture. This layer is infrastructure in the truest sense and therefore rarely maps directly to business requirements. However, this layer provides key capabilities that make the architecture service oriented and is the primary focus for meeting non-functional requirements such as scalability, reliability, availability, maintainability, etc.

3.1.8 User Interaction Systems

A User Interaction System is any system that accesses any layer within the integration architecture. There are a wide variety of systems that can access the integration layers. Some examples include web-based applications, fat client applications, partner applications, etc.

This is not formally a layer in the integration architecture. Rather these are the systems that gain value from the data and functionality exposed by the layers of the architecture.

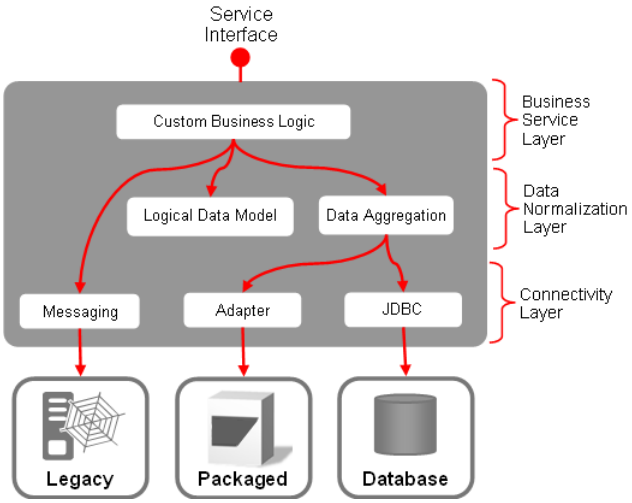
3.2 Development View

The Development View of the architecture describes aspects of the architecture that are of interest to developers building assets that conform to and leverage the architecture. In a service-oriented integration architecture the primary developer artifacts are the SOA Services that are created to expose data and functionality contained in source systems.

3.2.1 Services

A SOA Service may be exposed at any layer in the logical architecture. A SOA Service implementation may also span layers in the integration architecture as depicted in Figure 3–2.

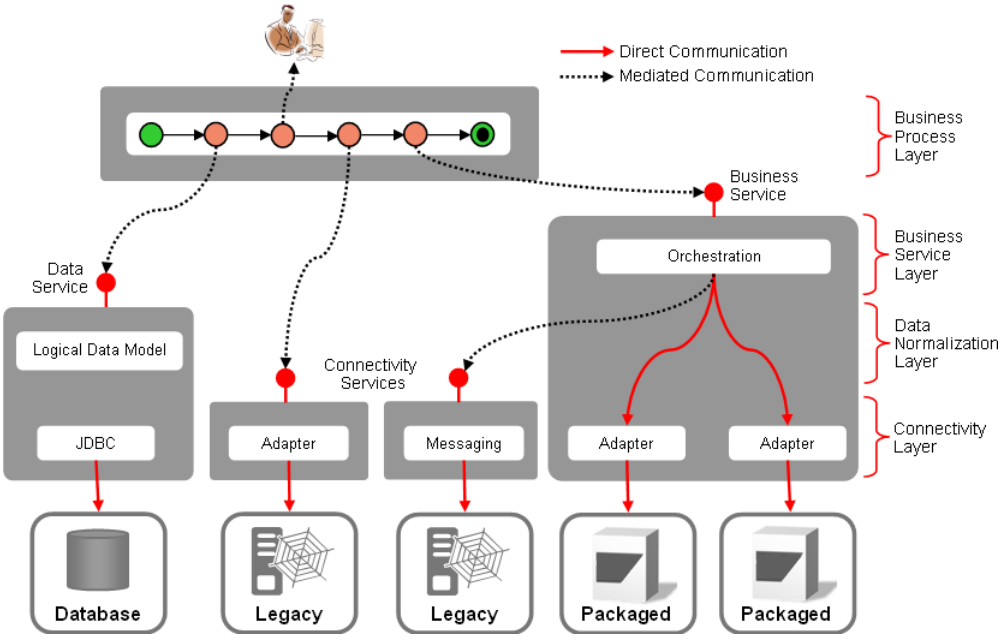
Figure 3–2 Business Service Spanning Layers



When constructing a SOA Service that spans layers, it is important that the developer still understand and apply the layers so the implementation uses the same separation of concerns. This not only makes the SOA Service easier to maintain, it also makes it possible to easily refactor the SOA Service to expose lower level functionality as a separate and distinct SOA Service.

The SOA Services can be classified by the uppermost layer that is included in the service functionality. Figure 3–3 illustrates several types of SOA Services and also illustrates how upper layers call services from lower layers in the architecture.

Figure 3–3 Architecture Layers and Services



In Figure 3–3, the dotted lines indicate communication paths that are going through the mediation layer, whereas the solid lines indicate direct communication that is not mediated. Communication internal to a SOA Service implementation does not use the mediation layer. All communication using the interface to a SOA Service should pass through the mediation layer.

Figure 3–3 shows the business process directly accessing a Connectivity Service. While the architecture allows this direct interaction, in general this should be avoided since it tends to make the business process directly dependent on the actual backend system being called. A much better approach is to keep the business process isolated from the backend systems by having the business process call only Business Services or Data Services. An example where the business process might call a Connectivity Service directly is when the backend system is built leveraging an existing standard and the Connectivity Service exposes a standard interface (e.g. OSS/J).

3.2.2 Business Process versus Orchestration

Figure 3–3 illustrates a business process and an orchestration. The business process in this example is depicted as a system-centric workflow with one step that requires human interaction. The orchestration is strictly system-to-system interactions. This separation illustrates a key concept for developers, namely that business processes should not contain any lower-level technical details. The technical details should be encapsulated by Business Services that expose an interface that is meaningful to a business user.

For example, the business process may have a step that requires updating a customer address. However, the customer address may be stored in several backend systems. This complexity is not of interest to a business user, nor is it a core part of the business process. Therefore, this technical complexity should be encapsulated by a shared Business Service that exposes an 'update customer address' operation. The operation then updates all necessary systems. Notice that this encapsulation also makes it possible to change the backend systems or even remove backend systems without any impact to business processes that update customer addresses. Only the shared Business Service needs to be modified.

Conversely, the orchestration that is included in the Business Services should not include steps that are likely to change due to changes in business strategy or execution. Those steps belong in the Business Process Layer so that they can be easily changed whenever the business changes.

3.2.3 Service Composition

Service composition is the ability to leverage lower-level services to create a higher-level service. The orchestration shown in Figure 3–3 is an example of service composition since the Business Service leverages the Connectivity Service to supply some of the necessary capability. This architecture supports and encourages service composition as a primary approach for developers to apply.

When doing composition, the developer should respect the layering of the architecture. Thus, a Business Service could leverage existing Connectivity Services or Data Services and a Data Service could leverage existing Connectivity Services. But, a Data Service should not call a Business Service and a Connectivity Service should not call a Data Service or a Business Service. Composition by calling services within the same layer of the architecture (e.g. Business Service calling another Business Service) is also fully supported. However, care must be taken when using composition within a layer since this can lead to complex dependencies that hinder maintenance and versioning.

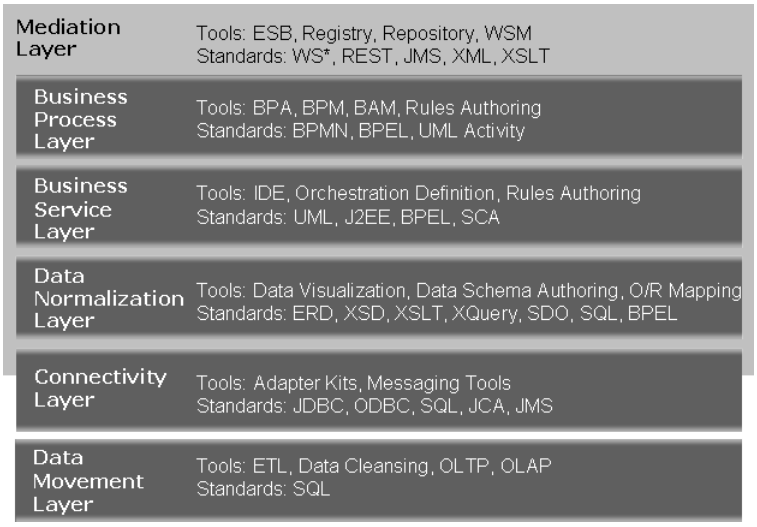
Depending on the products and technology used to implement the Mediation Layer, it may also be possible to construct service compositions in the Mediation Layer. This is commonly referred to as a service pipeline. In a service pipeline, the Mediation Layer handles a service request by routing the request to one or more services potentially doing message transformations and protocol translations as well. While powerful, this

capability should be used judiciously since the Mediation Layer is not a general purpose programming environment and is usually a shared resource. Excessive computing or configuration errors within the Mediation Layer could negatively impact all service traffic passing through the Mediation Layer.

3.2.4 Tools and Technologies

This service-oriented integration architecture does not require the usage of any specific tools or technologies. Rather the architecture is based on the principles that were presented in Chapter 2 and can be realized using a wide variety of tools and technologies. Nonetheless, there are types of tools and technologies that are well suited to service-oriented integration, fit nicely with the architecture, and therefore are likely to be used by developers applying this architecture. These tools and technologies are shown in Figure 3–4 and described for each layer in the architecture.

Figure 3–4 Tools and Technologies



3.2.4.1 Mediation Layer

The primary tools and technologies for this layer are Enterprise Service Bus (ESB), Registry, Repository, and Web Service Management (WSM). There are also many existing and emerging protocol standards that come into play including WS*, REST, JMS, XML, XSLT, etc.

3.2.4.2 Business Process Layer

The primary tools and technologies for this layer are for capturing, analyzing, executing, and monitoring business processes. These include Business Process Analysis (BPA), Business Process Management (BPM), and Business Activity Monitoring (BAM). Rules engines may also be used to capture and execute decision rules for business processes. There are several associated standards including BPMN, BPEL, and UML Activity Diagrams.

3.2.4.3 Business Service Layer

The primary tools and technologies for this layer include traditional software development tools (e.g. IDE). Additionally, the orchestration capability requires a developer focused process definition tool. Rules engines may also be used to capture

and execute business rules. Associated standards include UML, J2EE, BPEL, SCA, WSCDL, etc.

3.2.4.4 Data Normalization Layer

The primary tools and technologies for this layer are for modeling data and for defining and manipulating data formats. Thus, the primary tools are visualization and authoring tools for ERDs, O/R mapping, XML, XSD, XQuery, XSLT, etc. Industry standard data formats are also important for this layer. Because this layer also performs data aggregation and data synchronization, orchestration tools (e.g. BPEL) are also essential for this layer.

3.2.4.5 Connectivity Layer

The primary tools and technologies for this layer are adapter development kits and messaging technologies. Also, there is a plethora of possible technologies that may be needed to connect to the wide variety of systems that may be included as source systems. Associated standards include JDBC, ODBC, and JCA.

3.2.4.6 Data Movement Layer

The primary tools and technologies for this layer are ETL tools and data cleansing tools. Persistent storage tools and technologies for both OLTP and OLAP are vitally important for this layer.

3.3 Process View

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

3.3.1 Mediation

Mediation is a key component in the overall architecture providing the decoupling between consumers and providers. Figure 3–5 illustrates a typical flow of a message through the mediation layer.

Figure 3–5 Mediation Processing

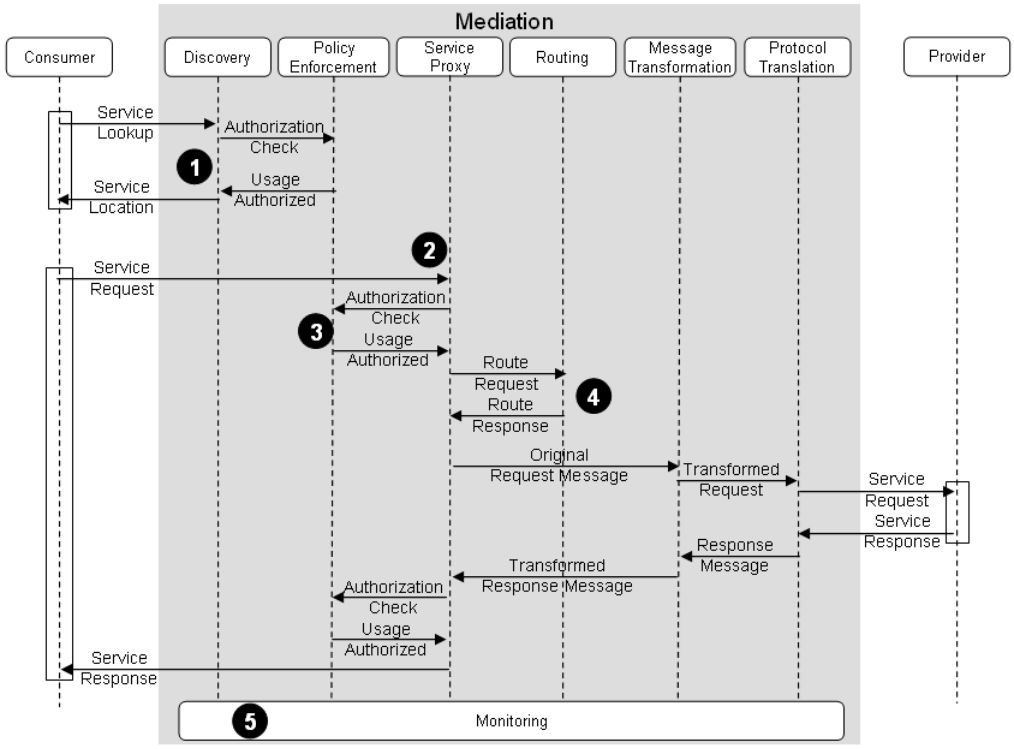


Figure 3–5 is only an example of the activities within the Mediation Layer and does not necessarily correspond to any particular product. Descriptions for the numbers shown in Figure 3–5 are as follows:

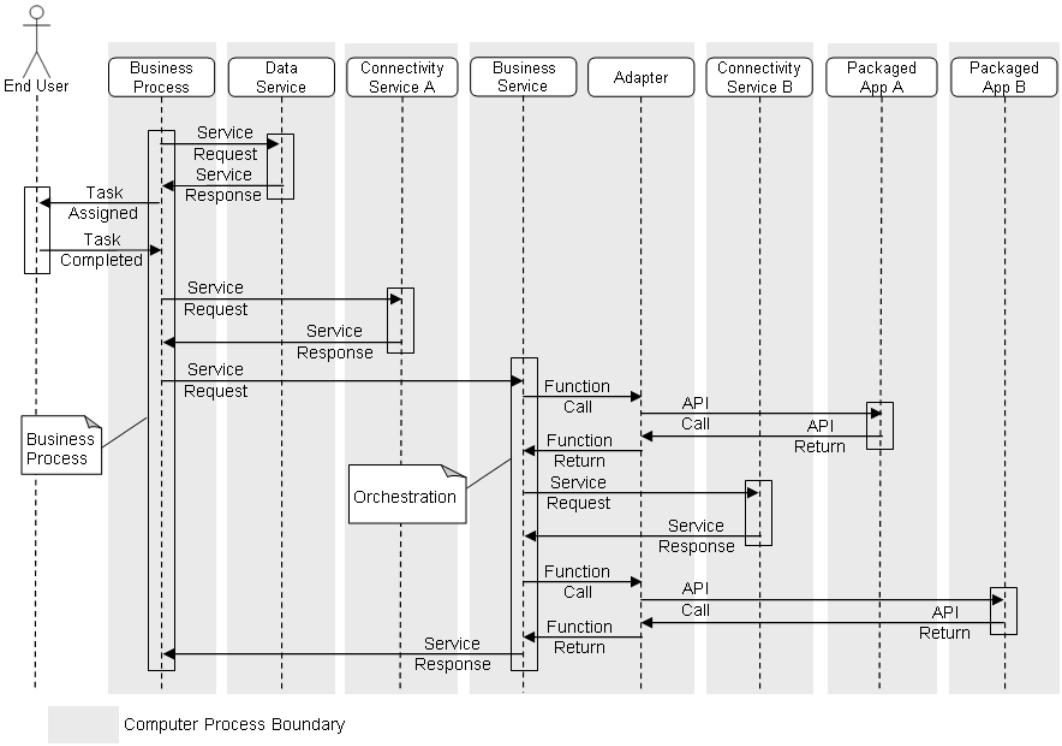
1. Before calling a SOA Service, the service consumer must first find the appropriate SOA Services. This is called service discovery and may be performed at development time or at runtime. The architecture supports both. There is also an authorization check done to see which SOA Services the consumer is allowed to see.
2. The service request is made to a service proxy, not directly to service provider. The service proxy allows the Mediation Layer to apply all the capabilities that have been configured for that service request.
3. An authorization check is performed on the service request. Although not always required, leveraging the authorization capability within the Mediation Layer provides a centralized approach to securing SOA Services. An authorization check may also be performed before forwarding the response message to ensure that the consumer is allowed to see the contents within the response message.
4. Once the service request has been authorized, the request is routed to the appropriate service provider. The request message may be transformed before being sent to the provider and the protocol may be translated as well.
5. All of the activities within the Mediation Layer are monitored. This provides a centralized monitoring capability across all SOA Services.

Figure 3–5 shows a simple one-to-one call between a service request and a service provider. However, as described in Section 3.2.3, the Mediation Layer might implement a service pipeline that results in multiple service providers being called i.e. step 4 above might be repeated.

3.3.2 Process Boundaries

The service-oriented integration architecture spans multiple systems and the computer processes within those systems. Figure 3–6 is another view of the SOA Services and integration shown in Figure 3–3. It illustrates the message flow and the process boundaries that are crossed by the messages.

Figure 3–6 Process Boundaries



As this figure illustrates, a single business process may result in messages that span many computer processes. In general, every service invocation results in a process boundary being crossed. In fact, since each service invocation also passes through the Mediation Layer (as shown in Figure 3–5), each service invocation crosses two process boundaries, mediation and service provider. Additionally, the service implementation may also cross service boundaries. For example, the Business Service shown above makes API calls to packaged applications (via adapters) and each packaged application is running in its own process.

Although seven different processes are shown, Figure 3–6 actually simplifies the number of processes included in the integration. Both of the Connectivity Services (A and B) would likely include out-of-process calls to the source system for which they are providing connectivity.

Each time a process boundary is crossed there are performance impacts from the network and message marshalling and de-marshalling. This is a primary reason why SOA Services should expose relatively course-grained interfaces. This is also a reason why a service implementation might span multiple layers in the architecture as illustrated in Figure 3–2. This also explains why the architecture includes a separate Data Movement Layer, since moving large quantities of data across service boundaries is computationally very expensive.

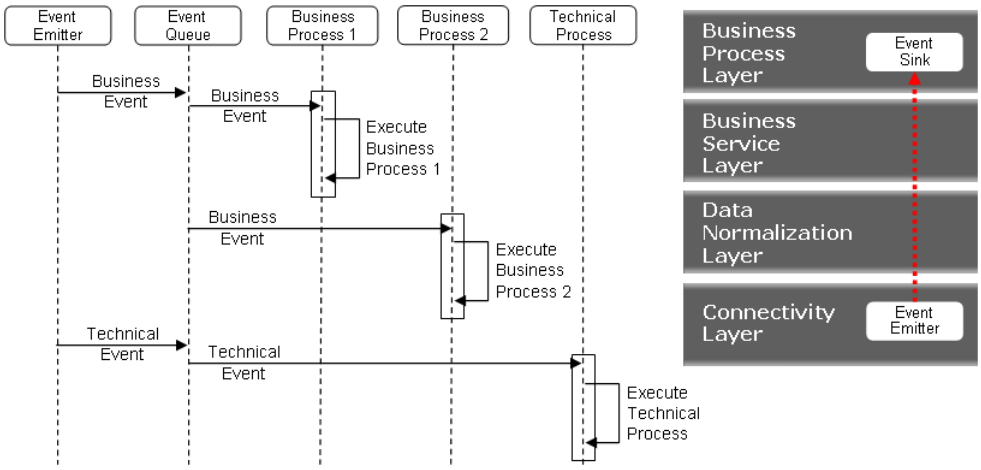
3.3.3 Event Handling

Events allow one system (event emitter) to notify other systems (event sink) that something of interest has changed. There are two broad categories of event types:

- **Business Event** - A business event is an event that is of business relevance and would be readily understood by a business person. An example of a business event is an 'inventory low' event.
- **Technical Event** - A technical event is an event that is relevant to IT but not directly relevant to the business. However, the cause of the event may impact the business especially if it is not handled expeditiously. An example of a technical event is an 'SLA violation' event. Another example of a technical event is an event that is used to synchronize data across backend systems.

These two types of events and how they are handled are illustrated in Figure 3–7.

Figure 3–7 Event Handling



As illustrated by Figure 3–7, in this architecture all events are routed to the Business Process Layer and the appropriate business processes are executed for that event. Essentially this is a mechanism for a lower level in the architecture stack, the Connectivity Layer, to initiate actions that might include interactions with all other levels in the architecture. This is essential since the generated event will likely be backend system specific; therefore it is likely that the data must be normalized and some amount of custom logic may be required to convert the event into an event that is backend system agnostic.

The types of business process that the event triggers will depend on whether the event is a business event or a technical event.

- **Business Process** - A business event will trigger a business process i.e. a process that will involve business activities and frequently requires actions from a business person. For example, an 'inventory low' event might trigger a 'restock inventory' business process.
- **Technical Process** - A technical process is a business process for IT i.e. it is a process that synchronizes backend systems or rectifies some problem in the computing environment and may require actions from an IT resource. For example, a 'SLA violation' event might trigger a 'provision resources' technical process.

Figure 3–7 shows the execution of two business processes for the business event and one technical process for the technical event. The event handling is based on the

publish-and-subscribe messaging pattern (see Section 5.4). This allows any number of subscribers to register for events from the event queue; thus any number of business processes can be initiated for a particular event.

While many events are independent and the order in which they are processed is unimportant, there are also cases where the order of the events is vitally important. Therefore, the event handling in the architecture must provide the capability for the events to be strictly ordered, and thereby ensure that events are processed in the same order as they were generated.

This event handling approach is not suitable for very high volume events (e.g. stock ticker) since each event triggers a business process. As noted in Section 1.3.3, complex event processing is not included in this architecture. Additional capabilities not including in this architecture are required to handle CEP and high volume events.

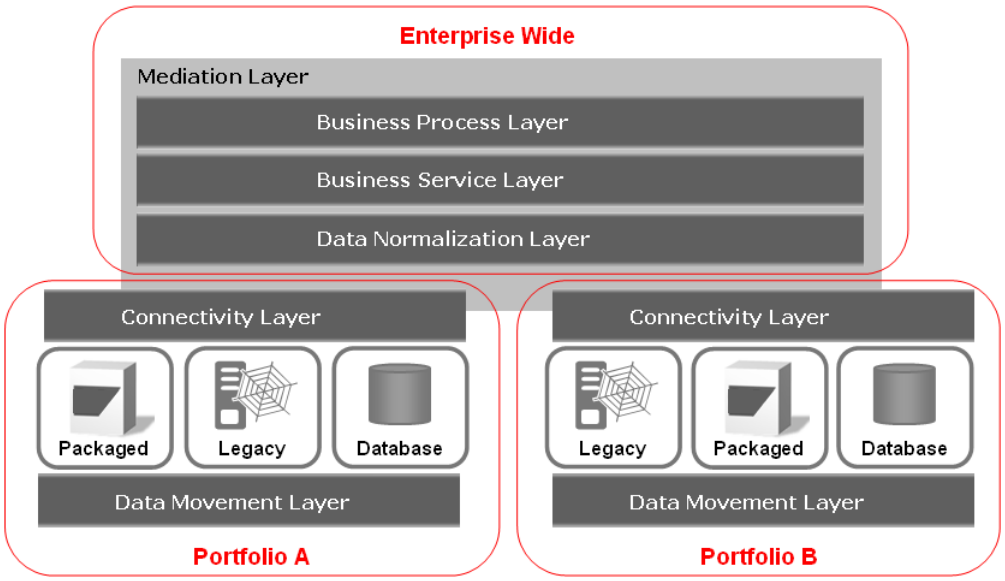
3.4 Deployment View

There are a myriad of ways that the architecture can be deployed within an enterprise. Organizational size, structure, and span of management control play a significant role in determining many aspects of deployment, especially how much is shared across the enterprise and how much is specific to a portfolio or division.

3.4.1 Mostly-Shared Deployment

In a mostly shared deployment, only the lower levels of the architecture are deployed independently across portfolios or divisions. This type of deployment is illustrated in Figure 3–8.

Figure 3–8 Mostly-Shared Deployment

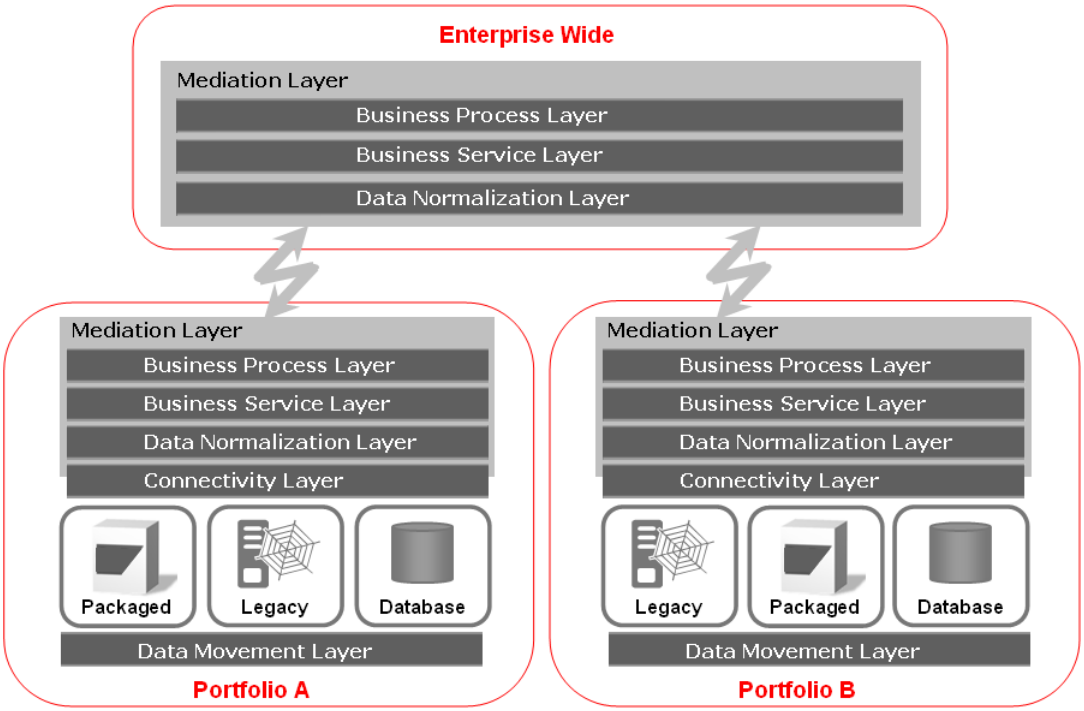


This type of deployment has most of the layers deployed such that they are shared across the enterprise. Only the lowest levels of the architecture are specific to the portfolios. It should also be clear that for a smaller enterprise it is possible to have all of the integration layers shared enterprise wide.

3.4.2 Hierarchical Deployment

As the enterprise size increases it becomes more difficult to deploy a single shared environment for integration. It becomes more likely that a distributed or hierarchical deployment would be a better fit as illustrated in Figure 3–9.

Figure 3–9 Hierarchical Deployment



This type of hierarchical deployment has several advantages including:

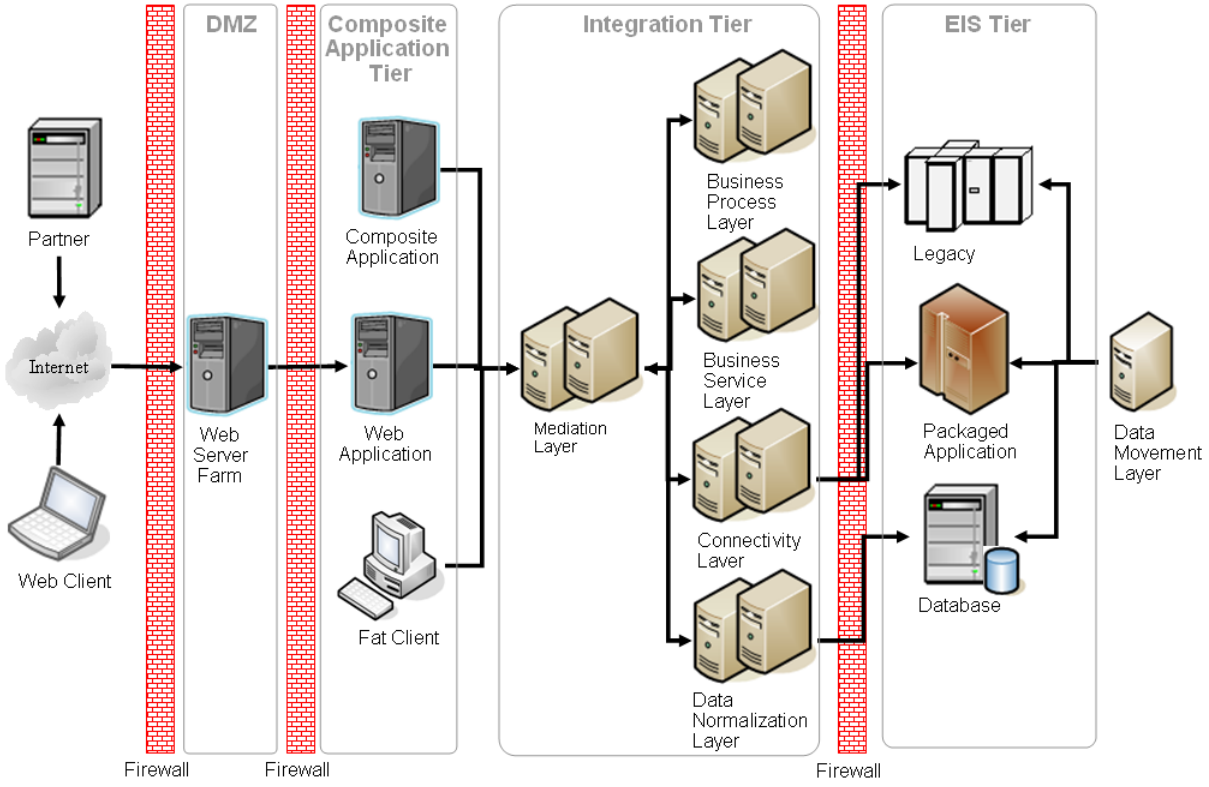
- Each portfolio can define its own business processes, SOA Services, and data formats independent from other portfolios.
- This deployment supports the federated data normalization approach described in Section 1.1.4.
- Business processes or SOA Services that span portfolios exist at the enterprise level and can leverage the portfolio business processes and SOA Services.
- The mediation layer provides location transparency between the portfolio and enterprise domains.
- This type of deployment easily supports acquisitions since a new acquisition can be treated as simply another portfolio.

The primary disadvantage of a hierarchical deployment is the increased cost and complexity of supporting more hardware and software. Adherence to standards to support interoperability is also more important in a hierarchical deployment since the various portfolios may select different products.

3.4.3 Physical Deployment

There are a myriad of ways that the architecture can be deployed to hardware. Figure 3–10 illustrates a high-level view of how the architecture might be deployed to hardware.

Figure 3–10 Physical Deployment of Layers



The illustration shows each layer deployed to separate hardware. While this deployment is possible, it may also be desirable to deploy layers of the architecture to the same hardware. Showing the layers deployed to separate hardware makes the communication paths more explicit. This also shows the layers deployed to multiple hardware boxes to provide scalability and reliability. In a modern data center where server virtualization is widespread, the layers would be deployed to virtual servers. The number of virtual servers hosting each layer would be determined by the load on the layer and could be dynamically provisioned as necessary.

Many of the detailed hardware deployment decisions are driven by the specific products and technologies used to realize the architecture. A more detailed, product specific hardware deployment illustration is contained in Section 4.3.

3.5 Security

Security is a very large topic unto itself and a general discussion of security is beyond the scope of this document. However, there are some security topics that are of particular relevance to service-oriented integration. These topics are discussed in this section. The ***ORA Security*** document contains much more information on security in a service-oriented architecture.

3.5.1 Transport-Level Security

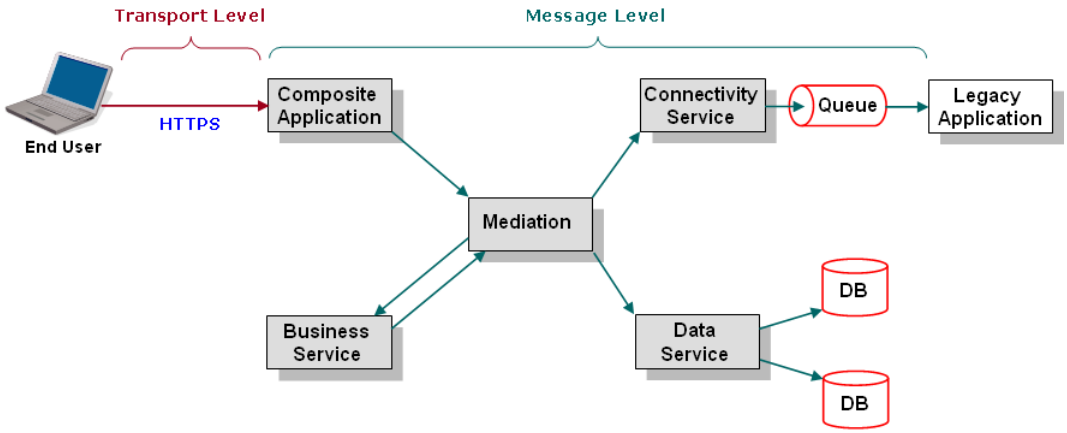
Transport-level security establishes a secure connection between two network endpoints. Transport-level security is widely implemented as Secure Socket Layer (SSL) and Secure Hype-Text Transfer Protocol (HTTPS). Service-oriented integration leverages these standards to provide security between network endpoints in the

architecture. SOA Services that are implemented using the WS* standards can easily apply HTTPS wherever needed.

3.5.2 Message-Level Security

Message-level security protects the contents of a message across multiple network endpoints and across multiple layers of the integration architecture as illustrated in Figure 3–11.

Figure 3–11 Message-Level Security



SOA Services that are implemented using the WS* standards can apply the WS-Security standards to provide message-level security. Message-level security should be used whenever the contents of the message contain information that should only be available to parts of the integration architecture.

For example, in a loan origination business process, the user's social security number will be required to obtain their credit score. However, this information should not be available for all the other steps in the business process.

3.5.3 Security Propagation

Security propagation allows the service consumer's security credentials to be used to make security decisions farther downstream in the processing of a service request. Providing security propagation end-to-end in service-oriented integration is very difficult because the processing of a service request may cross multiple process boundaries and involve multiple different products and technologies. Thus, end-to-end security propagation is usually not a capability provided by a service-oriented integration architecture.

Nonetheless, security propagation is valuable and is usually supported across some layers in the integration architecture. For example, the end user security credential should always be available to the Business Process Layer in the architecture.

3.5.4 Credential Mapping

Credential mapping provides the ability to swap one security credential for another security credential. Credential mapping is frequently required when connecting to source systems, especially legacy systems, since the source systems frequently implement their own security based on users that are configured inside the system. Credential mapping allows the data and functionality encapsulated by the source

system to be exposed to service requests that originate from users that are not configured in the source system.

4

Product Mapping

This section describes how Oracle products fit together to realize the service-oriented architecture defined in the previous section.

4.1 Products Included

There are two categories of Oracle products that map into the service-oriented integration architecture, Fusion Middleware products and the Application Integration Architecture (AIA) products.

4.1.1 Fusion Middleware Products

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

- **Oracle BPEL Process Manager (BPEL-PM)** - BPEL-PM provides the ability to quickly build and deploy orchestrations and business processes in a standards-based manner.
- **Oracle Business Activity Monitoring (OBAM)** - OBAM is a complete solution for building interactive, real-time dashboards and proactive alerts for monitoring business processes and services.
- **Oracle Business Process Management (OBPM)** - OBPM is a complete set of tools enabling collaboration between business and IT to create, automate, execute, and optimize business processes.
- **Oracle Business Rules (OBR)** - OBR evaluate rules rapidly using a light-weight, high performance rules engine.
- **Oracle Business-to-Business Integration (OB2BI)** - OB2BI provides a single integrated solution for rapidly establishing online collaborations and automated processes with your business partners.
- **Oracle Data Integrator (ODI)** - ODI is a next-generation Extract Load and Transform (E-LT) technology that offers the productivity of a declarative design approach, as well as the benefits of a platform for seamless batch and real-time integration.
- **Oracle Data Profiling and Data Quality (ODPDQ)** - ODPDQ is an investigation and quality monitoring tool and a powerful rule-based data cleansing engine.
- **Oracle Data Service Integrator (ODSI)** - ODSI provides the ability to quickly develop and manage federated, bidirectional (read and write) data services for accessing single views of disparate information in a standards based, declarative manner.

- **Oracle Enterprise Repository (OER)** - OER is an industry-leading metadata repository providing a solid foundation for delivering governance throughout the service lifecycle by acting as the single source of truth for information surrounding SOA assets and their dependencies.
- **Oracle Integration Adapters (OIA)** - OIA uses standards based on the J2EE platform to create a service orientated approach to unlocking the assets that have evolved in IT environments.
- **Oracle Service Bus (OSB)** - OSB is designed to connect, mediate, and manage interactions between heterogeneous services, legacy applications, packaged applications and multiple ESB instances. It includes built-in management and monitoring capabilities.
- **Oracle Service Registry (OSR)** - OSR is a fully compliant UDDI v3 registry providing a standards-based interface for runtime infrastructure to dynamically discover and bind to deployed service end points.
- **Oracle Web Service Manager (OWSM)** - OWSM provides a solution for governing the interactions with services through security and operational policy management and enforcement.
- **Oracle WebLogic Suite (OWLS)** - OWLS fully implements the latest Java EE standards, provides industry leading reliability and performance, and includes comprehensive management capabilities.

There are other Fusion Middleware products that could be listed (e.g. Web Center, Identity Management) but these products do not provide the key capabilities required in service-oriented integration. Rather they are complementary products that can be added where appropriate to enhance the solution.

Each of the listed products provides key capabilities to a service-oriented integration architecture. However, not every organization will require all the capabilities these products provide. The integration requirements for each organization must be evaluated and the proper products included in the architecture.

4.1.2 Application Integration Architecture

The Application Integration Architecture (AIA) products are built using the products listed in Section 4.1.1.

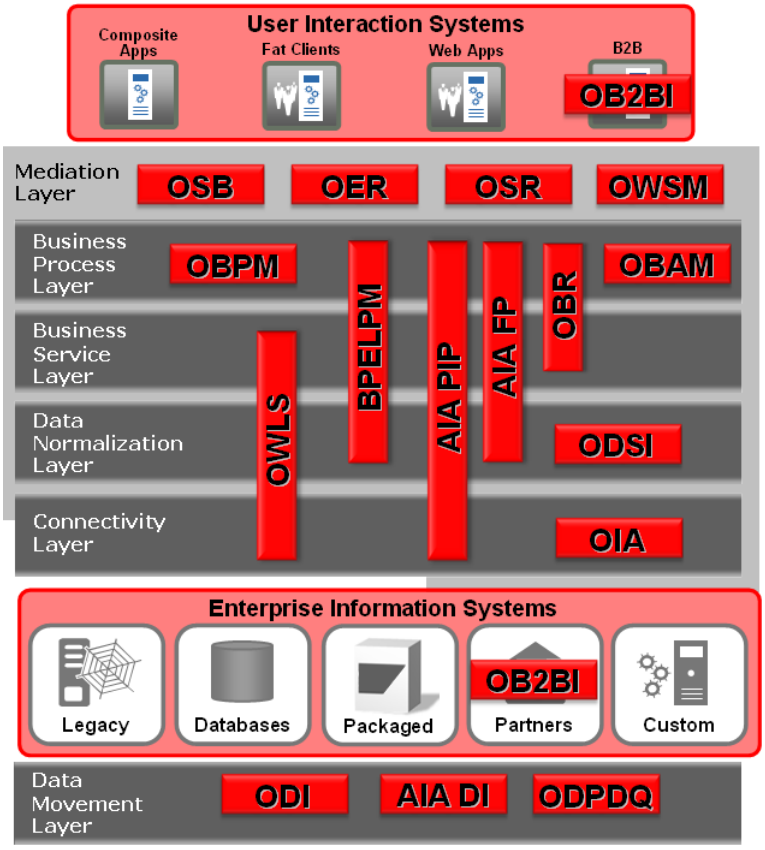
- **Foundation Pack (AIA FP)** - AIA FP includes application independent business process models that act as a blueprints for business processes and enterprise business objects and services that provide reusable, standards-based building blocks.
- **Direct Integration (AIA DI)** - AIA DIs are pre-built integrations that manage data flows and data synchronizations between applications.
- **Process Integration Pack (AIA PIP)** - AIA PIPs are pre-built composite business processes across enterprise applications.

It is worth noting that AIA provides solutions to rapidly integrate Oracle Applications into a service-oriented integration solution. In this architecture, Enterprise Information Systems would include any Oracle Applications deployed within the organization.

4.2 Product Mapping

The mapping of the products listed in Section 4.1 to the layers of the service-oriented integration architecture (Figure 3–1) is shown in Figure 4–1.

Figure 4–1 Oracle Product Mapping



Some products map to more than one layer in the architecture. When this is the case, it is important that service development still properly practices the separation of concerns that the layers in the architecture provide.

For example, BPELPM can be used to create either business processes or technical orchestrations. These two types of usage of BPELPM should remain separated (as described in Section 3.2.2) even if both types are deployed to a single instance of BPELPM.

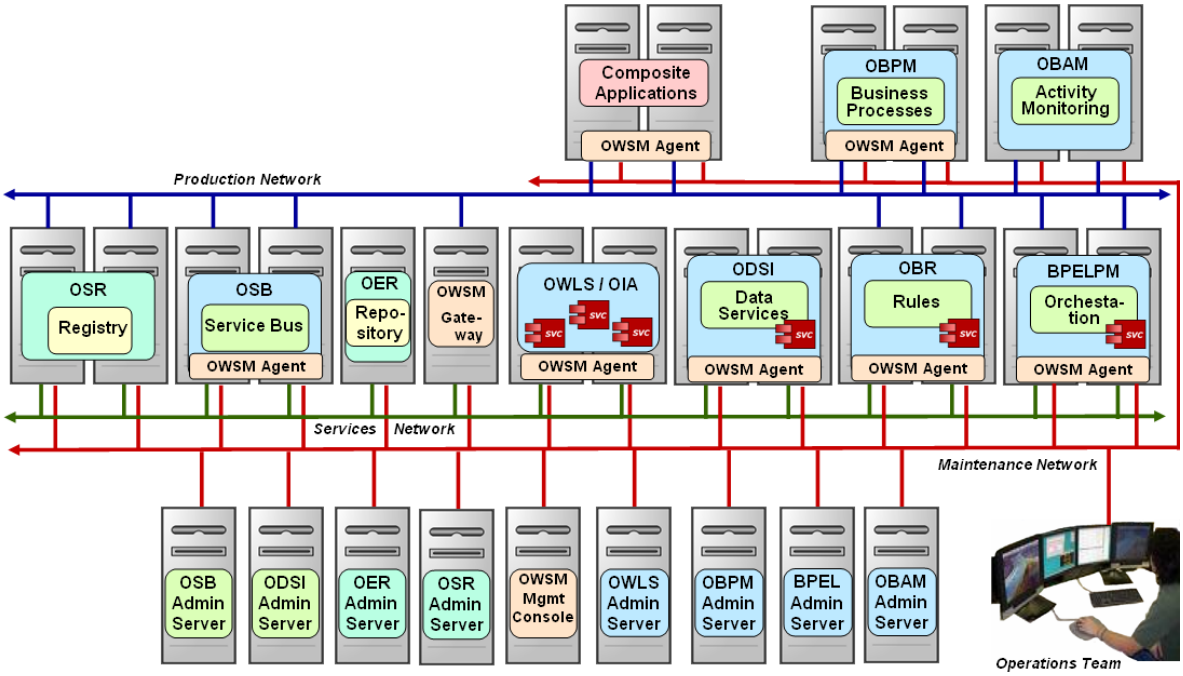
OWLS spans three layers of the architecture because J2EE is a standards-based general purpose programming environment; thus it can be used to host connectivity, data normalization, business service, or any combination of the three. This is commonly the approach used to create a SOA Service that spans multiple layers of the architecture as shown in Figure 3–2.

The AIA products are special in that they are pre-packaged, full-blown integrations; thus they span multiple layers in the architecture and are actually built on the other products.

4.3 Deployment View

This section provides an example of how the products listed in Section 4.1 might be deployed to physical hardware. A number of factors influence the way products are deployed in an enterprise; therefore it is not possible to provide a definitive deployment for the products. An example deployment is illustrated in Figure 4–2.

Figure 4–2 Deployment of Products



The sample deployment illustrates the positioning of infrastructure elements across three separate networks. The production network is a protected network that connects applications running in the production environment. This is an existing network that may actually be made up of several physical networks linked together. Existing applications can access the Registry for service discovery (both design-time and run-time). They can also access the Service Bus, which hosts proxy services that provide mediated access to the SOA Services.

A Services Network is shown connecting elements of the service-oriented integration architecture. This network provides connectivity between the Service Bus and SOA Services. The reason it is separated from the production network is to eliminate the possibility to access SOA Services without going through the Service Bus; thereby preventing service consumers from bypassing the mediation layer.

The third network is a maintenance network which is used by the operations team to administer various infrastructure components. It allows administration servers and operations terminals access to control production applications, services, and infrastructure.

All of the essential run-time components in the architecture are deployed to two physical machines to provide reliability, availability, and scalability. Naturally in a real deployment these components would be deployed to as many servers as necessary to provide sufficient capacity.

The administration and management console components are deployed to single hardware instances since these components are not essential to runtime. They are only necessary when an administrative change is being done to one or more of the essential run-time components.

The AIA products are not shown in the deployment diagram because these products are layered on top of the products shown in the deployment diagram.

4.4 Pre-Built Integration

AIA is much more that a suite of products that can be used to create a service-oriented integration. AIA actually includes pre-built business processes, services, entities, and data transformations that together equal a concrete realization of the product-agnostic, service-oriented integration architecture described in Chapter 3. Whereas Chapter 3 uses generic terms to describe the architecture, AIA defines specific terms with concrete realizations. This section defines these AIA concepts and puts them into the context of the generic architecture.[1]

4.4.1 AIA Concepts

This section describes each of the concepts built into AIA and how the concept relates to the product-agnostic architecture described in Chapter 3.

4.4.1.1 Industry Reference Models

Industry Reference Models include both the business process flows as well as the underlying canonical data models that identify the critical data to be exchanged during the business process.

Industry Reference Models provide the top-down analysis required to successfully build a catalog of SOA Services. This requirement to perform top-down analysis was described in Section 1.1.5.

4.4.1.2 Enterprise Business Object

Each Enterprise Business Object (EBO) provides an application-independent representation of business entities (e.g. customer, product, order) and is based on the canonical data models provided by the Industry Reference Models. EBOs are implemented as XML schemas via XSDs.

Enterprise Business Objects provide a realization of the Logical Data Model in the Data Normalization Layer described in Section 3.1.4.

4.4.1.3 Enterprise Business Message

An Enterprise Business Message (EBM) is an application-independent message that is used as input to or output from an Enterprise Business Service. EBMs are based on the EBOs but apply to specific operations on the Enterprise Business Services; therefore and EBM may contain only part of an EBO or may contain multiple instances of EBOs. EBMs also include a standard message header. EBMs are implemented as XML schemas via XSDs.

Enterprise Business Messages provide a realization of the request and response messages passed between SOA Services.

4.4.1.4 Enterprise Business Service

An Enterprise Business Service (EBS) provides the specific actions that occur against Enterprise Business Objects. For instance, add customer, delete customer, and modify customer are example operations on an EBS. All inputs and outputs to an EBS are EBMs. The EBS interface is described by WSDL.

There are two distinct types of EBSs provided by AIA: entity EBSs and process EBSs. An entity EBS provides read/write access (create, update, delete, query, sync, etc.) to

[1] This description of AIA is based on the 11gR1 release of AIA.

an EBO. A process EBS contains more complicated orchestration logic across multiple EBSs which might also include multiple EBOs.

Enterprise Business Services provide a realization of the Business Service Layer described in Section 3.1.5. An EBS might also encapsulate a business process (or sub-process) in which case the EBS would be a realization of the Business Process Layer described in Section 3.1.6.

4.4.1.5 Enterprise Business Flow

An Enterprise Business Flow (EBF) defines an orchestration of one or more EBSs and is based on the Industry Reference Models. Each EBF is contained within a Process EBS. The EBF is implemented as a BPEL flow.

The EBFs capture business processes and therefore provide a realization of System-Centric Workflows in the Business Process Layer described in Section 3.1.6. Some EBFs capture lower-level orchestrations across entity EBSs, and therefore would be classified as an orchestration in the Business Service Layer described in Section 3.1.5.

4.4.1.6 Application Business Message

An Application Business Message (ABM) is an application-specific message that is used as input to or output from an Application Business Connector Service. An ABM is based on the native data formats of the application. ABMs are implemented as XML schemas via XSDs.

Application Business Messages provide a realization of the request and response messages to/from Enterprise Information Systems described in Section 3.1.2.

4.4.1.7 Application Business Connector Service

An Application Business Connector (ABC) Service encapsulates the particulars of an application and exposes the functionality of the application in a representation that is aligned with the EBSs. This type of ABC Services is known as a provider ABCS. Conversely, another type of ABC Service, known as a requester ABC Service, provides the interface that allows the application to invoke operations on EBSs. In both types the ABC Service does the necessary transformations, enrichments, orchestrations, etc. as required to match the specifics of the application to the interfaces provided by the EBSs.

The interface to both types of ABC Services is described by WSDL. The inputs and outputs to a provider-specific ABC Serives are EBMs whereas the inputs and outputs to a requester-specific ABC Services are ABMs.

Provider-specific ABC Services span the Data Normalization Layer (Section 3.1.4) and the Connectivity Layer (Section 3.1.3) . Requestor-specific ABC Servicess fall within the Data Normalization Layer.

4.4.1.8 Enterprise Repository

The AIA uses OER to provide a unified repository for design, development, deployment, and governance.

4.4.1.9 Service Bus

AIA is certified on OSB which ensures that the SOA Services provided by AIA can be readily deployed to OSB.

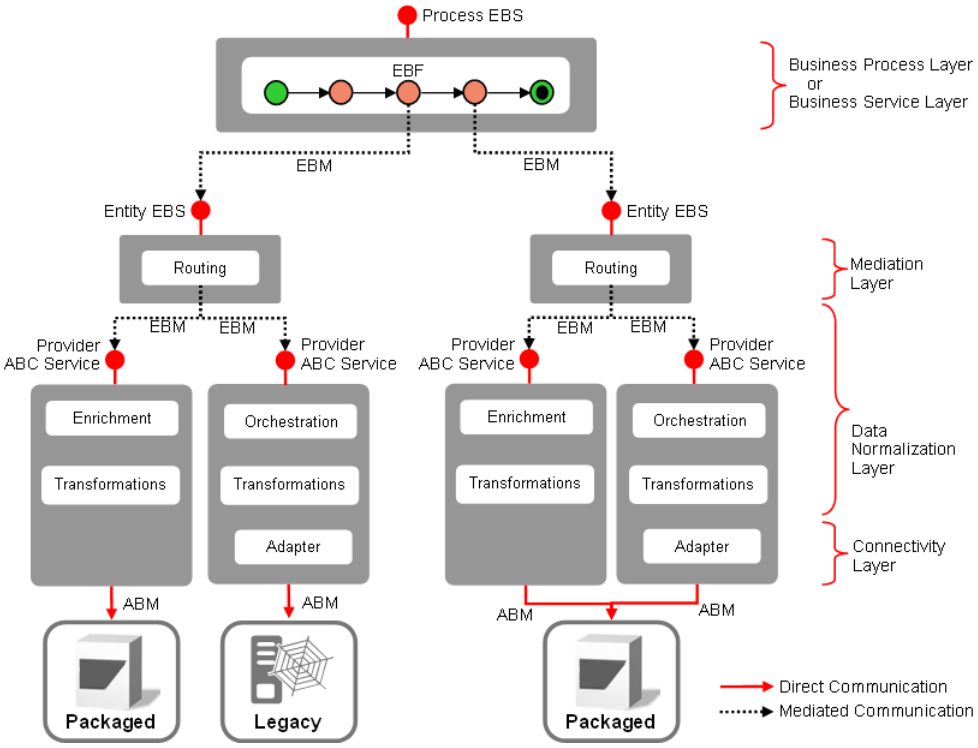
4.4.2 Illustrating AIA Concepts

This section illustrates the AIA concepts using the same graphics as were used to illustrate similar concepts in Chapter 3

4.4.2.1 Service Composition

AIA heavily leverages service composition. In fact, each EBS is constructed via composition since the EBS uses one or more ABC Services to provide its functionality. An example process EBS that uses entity EBSs is illustrated in Figure 4–3.

Figure 4–3 AIA Enterprise Business Service



This example illustrates one process EBS. If the EBF contained in the process EBS is implementing a lower-level orchestration (see Section 3.2.2) then the process EBS would fall into the Business Service Layer (Section 3.1.5). Conversely, if the EBF is implementing a business process then the process EBS would fall into the Business Process Layer (Section 3.1.6).

The process EBS in this example contains an EBF that defines a business flow that includes calls to two supporting entity EBSs. Each entity EBS routes the request from the process EBS to the appropriate provider ABC Service. Each ABC Service is responsible for servicing a request expressed in an application-agnostic EBM by transforming the data and interfacing with source systems. In this example, there are three source systems: two packaged applications and a legacy application. This example also shows that an ABC Service not only performs transformations but may also include an orchestration of lower-level interface calls to the source system to successfully respond to the more granular request from the EBS.

This example also illustrates that multiple ABC Services may connect to a single source system. This design prevents an ABC Service from becoming too large and unmanageable. Instead, the logic to encapsulate the source system can be partitioned into a number of ABC Services; thereby easing maintenance and versioning.

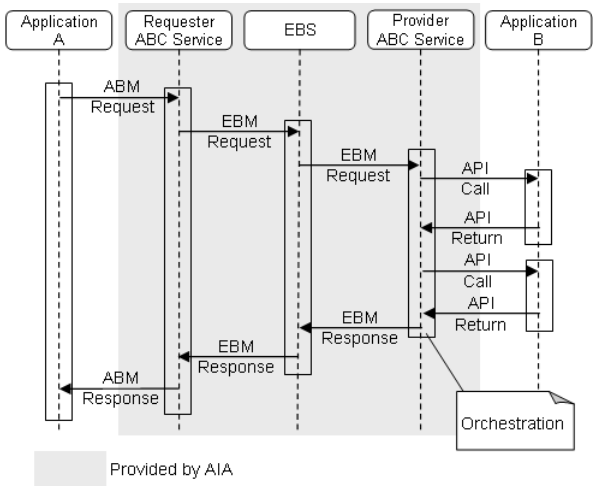
Figure 4–3 shows that the primary role of the entity EBS is to route the request to the appropriate ABC Service. Thus, the entity EBS is actually implemented in the Mediation Layer. This is an important concept: the service implementation includes the configuration of the service infrastructure. In AIA the infrastructure provides the routing capability for the entity EBSs. The infrastructure can provide other capabilities as well including security enforcement, protocol mediation, throttling, etc.

Obviously Figure 4–3 is just one example of many different possible scenarios. The key concept is that the EBS exposes application-agnostic functionality and the EBS delegates to one or more ABC Services. Each ABC Service encapsulates the complexity and application specifics of providing the application-agnostic functionality.

4.4.2.2 Service Request Processing

AIA supports a variety of application interaction patterns including synchronous request-response, asynchronous request-response, publish-and-subscribe, fire-and-forget (aka one-way) messaging, etc. An example request-response interaction is shown in Figure 4–4.

Figure 4–4 AIA Request-Response Interaction

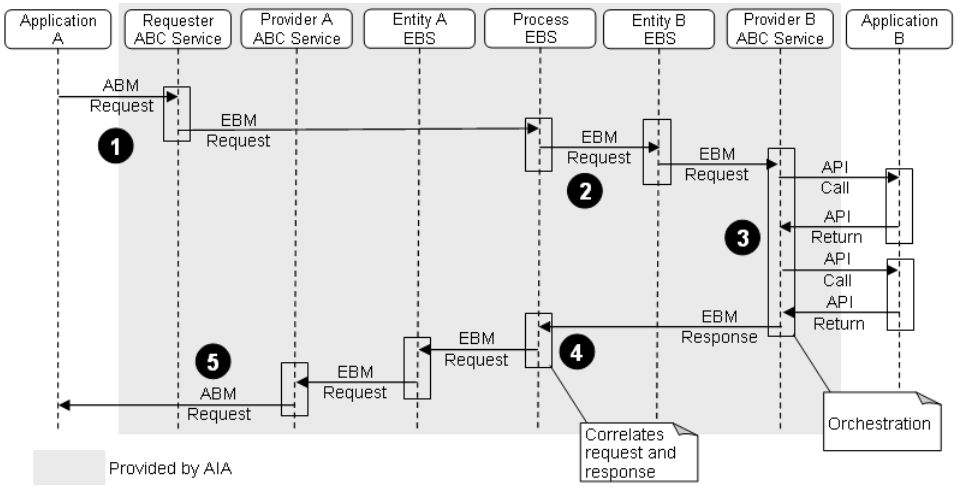


This example application interaction illustrates how the ABC Services encapsulate the application specifics for both the requesting application and the responding application. The requester ABC Service accepts an application specific message (ABM), converts that to an application-agnostic message (EBM), and calls the appropriate EBS to service the request. The EBS handles the request by delegating to the appropriate provider ABC Service. In this example the provider ABC Service includes a simple orchestration to make multiple API calls on the application. This illustrates that the application-agnostic request may not directly match the interface provided by the source application.

By encapsulating the application specifics in the ABC Services, the EBS remains application agnostic. This allows the EBS to capture and provide industry standard capabilities and best practices while also allowing plug-and-play with a variety of source applications. To support a different application only the ABC Service needs to change.

Not all application integration scenarios can be handled by simple synchronous request-response interactions. A more complex asynchronous request response is illustrated in Figure 4–5.

Figure 4–5 AIA Asynchronous Request-Response Interaction



Descriptions for the numbers shown in the figure are as follows:

1. Application A initiates a process flow by sending an ABM to a request ABC Service. The request ABC Service translates the message into an application-agnostic EBM and invokes a process EBS. Application A then continues processing without waiting for any response.
2. The process EBS forwards the EBM to the appropriate entity EBS and then enters a quiescent state until a response is returned at some later time. The entity EBS routes the EBM to the correct provider ABC Service.
3. The provider ABC Service translates the EBM into an application specific ABM and invokes operations on Application B. The ABC Service then takes the responses from Application B and constructs a new EBM that is the asynchronous response and invokes the process EBS with the EBM.
4. The process EBS is responsible for correlating the response with the original request so that the response can be properly handled. The process EBS invokes the appropriate entity EBS to process the response.
5. The entity EBS routes the EBM to the correct provider ABC Service. The provider ABC Service translates the EBM into an ABM and invokes the appropriate operation on Application A.

The diagram does not explicitly show the message queuing that is being done to decouple the various participants. The message queuing is done via JMS and is provided by the infrastructure upon which AIA is built.

4.4.3 Summary

This section provided a brief overview of AIA concepts and how those concepts provide a realization of the product-agnostic architecture presented in Chapter 3. AIA is not just an architecture. It is a service-oriented application integration architecture, design, and implementation.

5 Integration Patterns

Over the years there have been many different approaches used to integrate applications. Some of the approaches that proved to be particularly prevalent and successful have become known as integration patterns. There are entire books written on the different types of integration patterns. This document will not summarize or try to replace such works.

There are some patterns that are so common and so foundational that any integration architecture must support them. It is these foundational patterns that are documented here to ensure that any realization of the integration architecture described in Chapter 3 includes support for these patterns.

5.1 Message Exchange Patterns

Message exchange patterns describe the order and sequence of messages passing between a service consumer and a service provider. The foundational message exchange patterns that the architecture must support include:

5.1.1 One-Way

This pattern is a standard one-way messaging exchange where the consumer sends a message to the provider with no error (fault) path provided. This is equivalent to the 'In-Only' message exchange pattern defined in WSDL 2.0.

5.1.2 Reliable One-Way

This pattern is for reliable one-way message exchanges. The consumer initiates with a message to which the provider responds with status. The status response can be either a success or a fault. This is equivalent to the 'Robust In-Only' message exchange pattern defined in WSDL 2.0.

5.1.3 Request-Response

This pattern is a standard two-way message exchange where the consumer initiates with a message, the provider responds with a message or the provider may respond with a fault if it fails to process the request. This is equivalent to the 'In-Out' message exchange pattern defined in WSDL 2.0.

5.1.4 Request Optional-Response

This pattern is a standard two-way message exchange where the provider's response is optional. Consumer and provider both have the option of generating a fault in

response to a message received during the interaction. This is equivalent to the 'In Optional-Out' message exchange pattern defined in WSDL 2.0.

5.2 Synchronous Communications

When using synchronous communication the service consumer blocks until the service provider responds. This is usually the easiest type of communication to program in the consumer application. Thus, synchronous communication must be supported by the architecture.

5.3 Asynchronous Communications

Many applications included in integration scenarios do not provide a synchronous interface. Asynchronous communication is also used when the response time for the source system is too slow to support the timelines of the calling systems. Thus the architecture must support asynchronous communications. An example asynchronous communication is illustrated in Figure 5–1.

Figure 5–1 Synchronous and Asynchronous Messaging

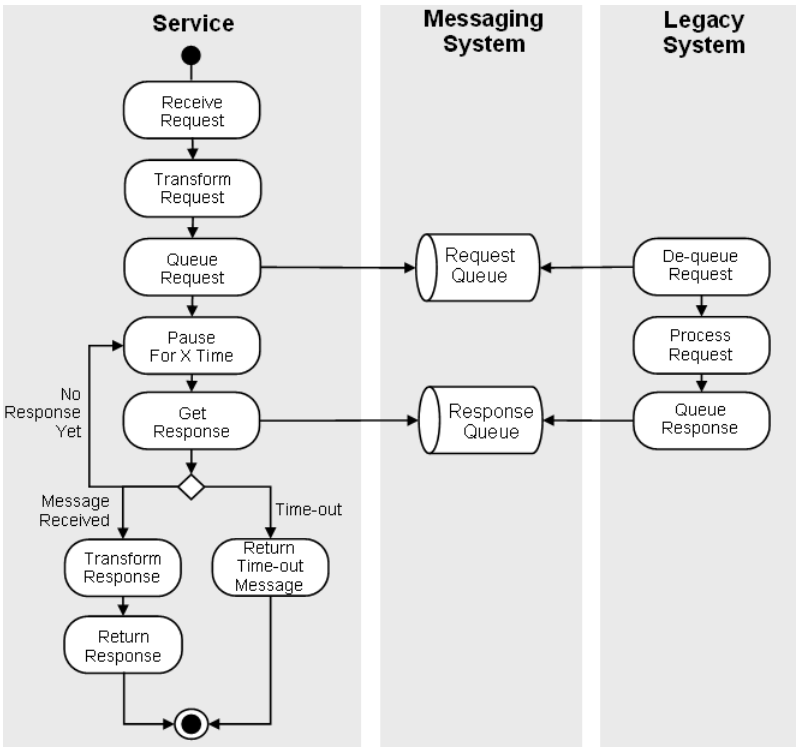


Figure 5–1 also illustrates the bridging of synchronous and asynchronous communications since the service waits for the asynchronous response from the legacy system before responding (synchronously) to the original request.

5.3.1 Bridging Synchronous and Asynchronous

In some cases the client requires a synchronous communication paradigm but the source system does not support synchronous communication. In this case the architecture must support bridging between synchronous and asynchronous. This

bridging capability is frequently provided as part of the protocol mediation capability within the Mediation Layer.

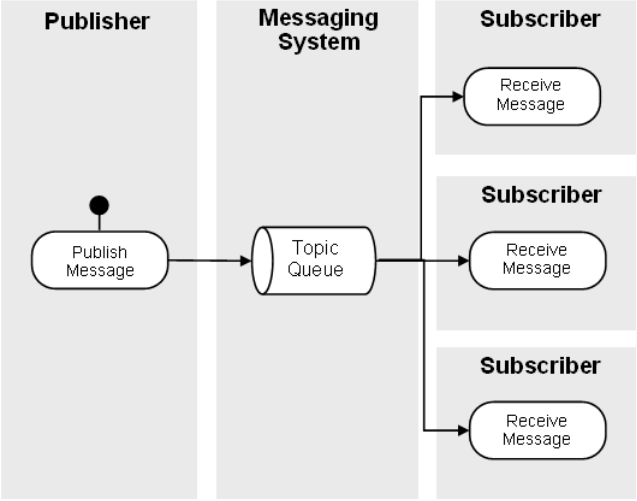
5.3.2 Store and Forward

Store-and-forward is a special case of asynchronous communication. In the store-and-forward pattern, the request message is put onto a queue for later retrieval by the target of the request message. Similarly, the response message is put onto a response queue for later retrieval. This is a very common approach used by messaging systems (e.g. MQ Series) to integrate with legacy systems. The architecture must support this integration pattern to facilitate integration with legacy systems and existing messaging systems.

5.4 Publish and Subscribe

The publish-and-subscribe communication pattern is used when a single message is of interest to multiple consumers. This pattern is illustrated in Figure 5–2.

Figure 5–2 Publish and Subscribe

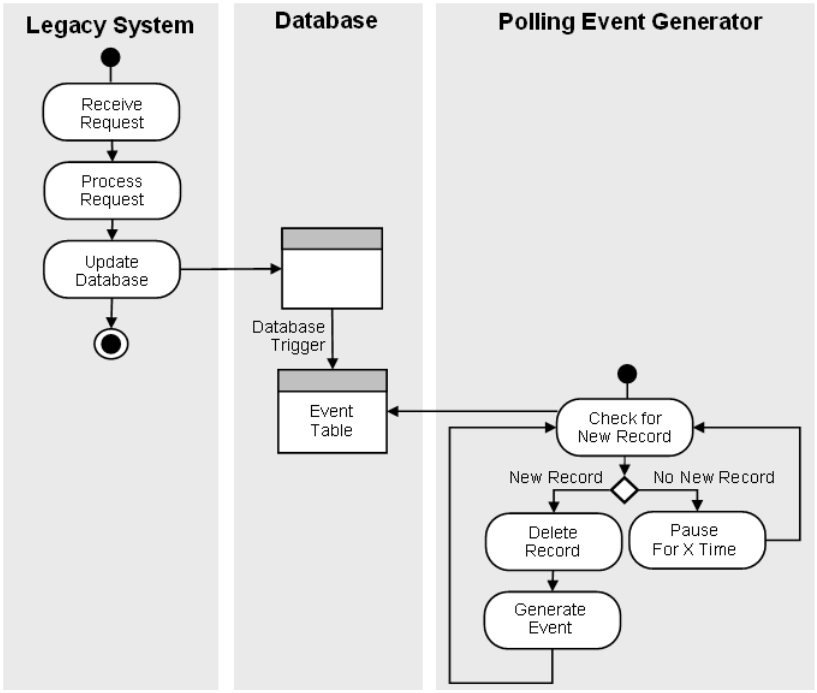


The publish-and-subscribe communication pattern is frequently used for event handling since an event may be of interest to a variety of consumers. The publish-and-subscribe communication pattern also frequently includes the ability for subscribers to set a filter for the messages that are of interest. This reduces the amount of computation for the subscriber by eliminating messages that are not of interest at the topic queue.

5.5 Polling

Polling is a common mechanism used to find messages or changes in applications that require action. The interaction shown in Figure 5–1 includes polling of the response queue to find a response message. This is one application for polling. Polling is also commonly used as an event generation approach for systems that do not natively support eventing. A common approach is to poll a database table for new records as illustrated in Figure 5–3.

Figure 5–3 Polling Event Generator



6

Summary

Integration is a continuing problem within enterprise IT. Although not a silver bullet, employing a service-oriented approach to integration provides tangible benefits and is superior to traditional integration techniques. Successful service-oriented integration is achieved by applying a layered architecture that provides all necessary capabilities and includes a consistent separation of concerns. Proper layering in the architecture eases maintenance by enforcing encapsulation of the various capabilities provided by the architecture.

Oracle offers a complete set of products that provide the foundation for service-oriented integration. This document includes various views of the Oracle products deployed in the layered architecture providing all the foundational capabilities required in service-oriented integration.

This document covered only the core concepts and components of service-oriented integration. Complementary technologies and products are covered separately and in greater depth in the appropriate Enterprise Technology Strategy (ETS) documents.

A

Further Reading

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

Software Engineering in an SOA Environment - This document describes the major aspects of a service-oriented engineering approach.

ORA SOA Foundation - This document presents important basic concepts of SOA that are instrumental to building applications for a SOA environment. It covers topics including the components of a SOA Service, service layering, service types, the service model, composite applications, invocation patterns, and standards that apply to SOA.

ORA SOA Infrastructure - This document describes the role of infrastructure and the capabilities it provides to a services environment. It offers an array of views to define infrastructure for SOA, including logical and physical views, as well as technology and product mapping.

ORA BPM Foundation - This document presents important basic concepts of BPM that are instrumental to modeling and automating business processes. It covers topics including a brief history of BPM, the business process lifecycle, the different types of modelling notations, applicable standards and technology, and how BPM relates to SOA.

ORA BPM Infrastructure - This document describes the role of infrastructure and the capabilities it provides to a BPM environment. It offers an array of views to define infrastructure for BPM, including logical and physical views, as well as technology, and product mapping.

In addition, the following materials and sources of information relevant to ORA Service-Oriented Integration may be useful:

Application Integration Architecture: Pre-Built SOA - This Oracle whitepaper describes Application Integration Architecture (AIA) and how AIA is used to quickly and easily integrate Oracle and non-Oracle applications.

Enterprise Composite Applications, Application Integration Architecture - This Oracle whitepaper presents specific business integration problems and describes how AIA was used to solve these problems.

Getting Started with Oracle Application Integration Architecture Foundation Pack and Demo - This Oracle whitepaper describes the Application Integration Architecture approach and demonstrates the concepts and components via the Application Integration Architecture Foundation Pack demo.

Integration Patterns - This web site describes numerous Enterprise Application Integration approaches and patterns.

SOA Patterns - This web site describes numerous design patterns used in Service-Oriented Architectures.