# Oracle® Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.2.0): Concepts and Technologies Guide

Release 1 (11.1.1.2.0)

**Part No. E17363-01**

April 2010

ORACLE

Oracle Application Integration Architecture Foundation Pack 11*g* Release 1 (11.1.1.2.0): Concepts and Technologies Guide

Part No. E17363-01

# Contents

# Preface

Welcome to the *Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.2.0): Concepts and Technologies Guide.*

Oracle Application Integration Architecture (AIA) provides the following guides and resources for this release:

## Oracle AIA Guides

- *Oracle Application Integration Architecture Foundation Pack: Installation Guide*
- *Oracle Application Integration Architecture Foundation Pack: Getting Started with the Oracle AIA Foundation Pack and Demo*
- *Oracle Application Integration Architecture Foundation Pack: Concepts and Technologies Guide*
- *Oracle Application Integration Architecture Foundation Pack: Development Guide*
- *Oracle Application Integration Architecture Foundation Pack: Infrastructure Components and Utilities Guide*
- *Oracle Application Integration Architecture Foundation Pack: Reference Process Model Guide*
- *Oracle Application Integration Architecture Foundation Pack: Migration Guide for Foundation Pack 2.X to Foundation Pack 11gR1 (11.1.1.2.0)*

## Additional Resources

The following resources are also available:

| Resource | Location |
|---|---|
| Oracle Application Integration Architecture Foundation Pack: Product-to-Guide Index | My Oracle Support: https://support.oracle.com/ |
| Known Issues and Workarounds | My Oracle Support: https://support.oracle.com/ |
| Release Notes | Oracle Technology Network: http://www.oracle.com/technology/ |
| Documentation updates | My Oracle Support: https://support.oracle.com/ |

# 1. Understanding the Oracle AIA Reference Architecture

This chapter discusses the following topics:



## 1.1. Introduction

Oracle Application Integration Architecture (AIA) is a complete integration solution for orchestrating agile, user-centric, business processes across enterprise applications. AIA offers prebuilt solutions at the data, process, and user interface levels delivering a complete process solution to business end users. All of the AIA components are designed to work together in a mix-and-match fashion. They are built for configurability, ultimately helping to lower IT costs and the burden of building, extending, and maintaining integrations.

Powered by Oracle Fusion Middleware, AIA enables organizations to utilize the applications of their choice and create Composite Business Processes (CBPs) following these guiding principles that define the ground rules for development, maintenance, and usage of a service-oriented architecture (SOA):

- Reuse, granularity, modularity, compose ability, componentization, and interoperability.
- Standards-compliance (both common and industry-specific).
- Service identification and categorization, provisioning and delivery, and monitoring and tracking.

## 1.2. Describing the Goals of AIA

The goal of AIA is to provide the following features via these deliverables:

| Features | Deliverables |
|---|---|
| A robust architectural framework for engineering service-oriented business processes. | • Reference Process Models<br>• Reference Architecture<br>• Foundation Pack – infrastructure components<br>• Process Integration Packs (PIPs) – prebuilt solutions |
| Support for interaction styles to handle high transaction rates and volumes that are associated with mission-critical applications. | Reference Architecture for different Integration Styles with and without canonical abstractions. |
| Ability to leverage functionality provided by various Oracle and customer-owned software assets. | Programming Models for constructing and assembling different types of AIA service artifacts that leverage various Oracle tools. |
| Ability for customers to extend various AIA artifacts delivered as part of PIPs. | Programming Models for extending various AIA service artifacts. |
| Support for process model decomposition and analysis, service design, service construction, process definition, deployment plan generation, deployment, and upgrade. | Project Lifecycle Management |
| Support for runtime monitoring of Integration Flows resulting from deployed AIA services. | Monitoring |
| Governance of design-time and runtime AIA artifacts. | Governance |

## 1.3. Describing Possible Integration Types Using AIA

With AIA, business processes can be engineered according to the following types of integrations:

- User Interface Integration

  A User Interface Integration connects disparate systems to provide a unified view to the user. It is a single view to many heterogeneous systems that are integrated at the user-interface level. It significantly increases the productivity to the end user by eliminating the need for the user to toggle back and forth between these systems. For example, in the Siebel Order Capture application, the order configuration capability of the Oracle E-Business Suite application has been embedded. Although this approach provides a unified approach to the users, there is no aggregation of data at the applications level.

- Data Integration

  A Data Integration connects at the logical level of data, making the same data available to more than one application. This is accomplished by relying upon database technologies. This type of integration is a good candidate when there is a minimal amount of business logic to be reused across applications.

- Functional Integration

A Functional Integration connects applications at the business-logic layer. This type of integration is a good candidate when there is a need for reuse of functionality, such as business logic or processing.

- Process Integration

  A Process Integration is primarily accomplished by exposing object interfaces that can be consumed by other systems, by using message-oriented middleware (MOM) systems to send messages to the destinations, or by exposing web service interfaces that can be consumed by the clients.

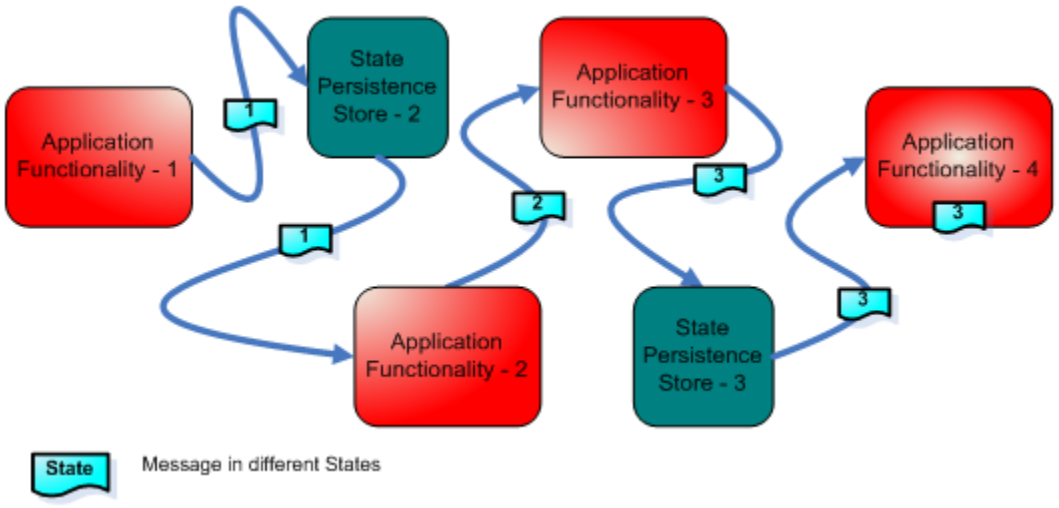
# 1.4. Describing Integration Styles Using AIA

This section discusses the following topics:

- Describing the Integration Flow Concept
- Describing Integration Through Native Application Interfaces Using the Oracle Applications Technology Infrastructure
- Describing Direct Integration Through Application Web Services Using Oracle SOA Suite
- Describing Integration Through Packaged Canonical and Standardized Interfaces Using Oracle Foundation Packs
- Describing Bulk Data Integration with an Extra, Transform, and Load Approach Using Oracle Data Integration Suite

## 1.4.1. Describing the Integration Flow Concept

An Integration Flow represents the journey of a message from a business event-triggering source, through possible intermediary milestones, to one or more target milestones. At each milestone, the message is stored in a different state.




An Integration Flow represents the runtime path of a message. It is not a design-time artifact.

AIA recommends a variety of integration styles and patterns to enable the flight of a message in an Integration Flow. The AIA artifacts that are required for the collaboration between applications or functions are dependent on the integration style adopted for an Integration Flow.

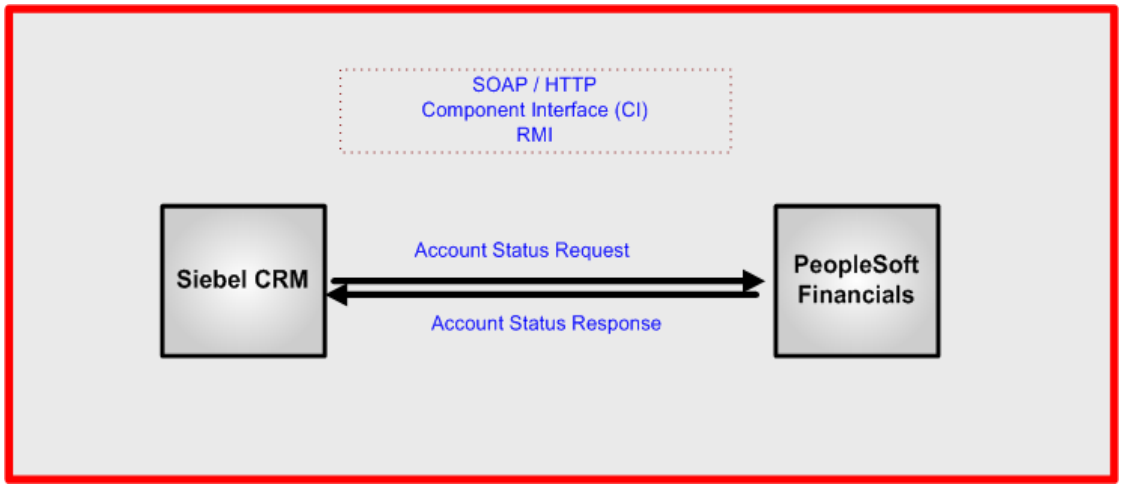
## 1.4.2. Describing Integration Through Native Application Interfaces Using the Oracle Applications Technology Infrastructure

In this style, messages flow from the requester application to the provider application. The mode of connectivity could be SOAP/HTTP, queues, topics, or native adapters. No middleware is involved in this integration.

The requester application must establish the connectivity with the provider applications. It is the responsibility of the requester application to send the request in the format mandated by provider's API, as well as to interpret the response sent by the provider. In addition, the requester and provider applications are responsible for the authentication and authorization of requests.

The Integration Flow consists of individual application functions interacting directly. All capabilities required to make this interaction possible need to be available in the individual applications.

The following diagram illustrates how a requester application interacts directly with the provider application.




Example of a requester application interacting directly with a provider application

In the case of more complex situations in which the Integration Flow consists of multiple steps involving interactions with multiple applications, it is imperative that workflow-like capability be leveraged in one or more applications.

There are no AIA artifacts that need to be built in this case. Direct connectivity must be established between applications.

### 1.4.3. Describing Direct Integration Through Application Web Services Using Oracle SOA Suite

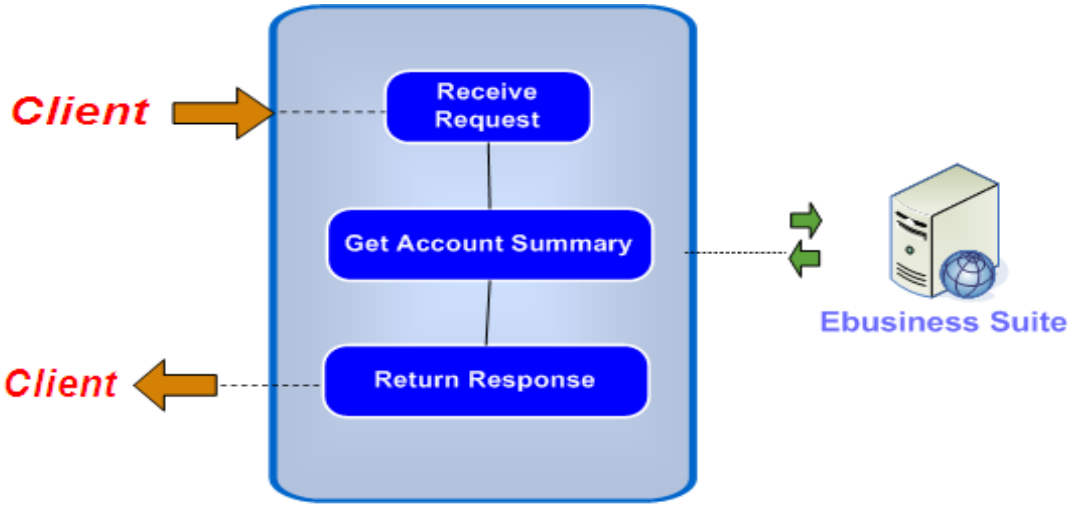
A provider application-specific AIA service, exposing a coarse-grained functionality of the provider application leveraging one or more APIs, is created with a suitable provider application-specific interface.

Several business initiators can invoke this AIA service. If the business initiators cannot present the request in the format understood by the provider application-specific AIA service, a requester application specific-AIA service is used to transform the business initiator request into the provider-application format.

The requester application-specific AIA service is responsible for the authentication and authorization of the requests. The provider application-specific AIA service propagates the authentication and authorization information of the requests to the provider application.

The Integration Flow consists of a requester application-specific AIA service artifact deployed on the middleware that manages interactions with all provider application-specific AIA services.

The following diagram illustrates how a service deployed on the middleware enables the integration between the requester and the provider applications.




Example of Integration Flow leveraging provider services

In the case of more complex situations in which the Integration Flow involves interactions with multiple applications, the requester application-specific AIA service implements a workflow-like capability and manages interactions with all provider application-specific AIA services.

The AIA service artifacts that need to be developed are determined by the complexity of the data exchange and the various message exchange patterns that are involved.

## 1.4.4. Describing Integration Through Packaged Canonical and Standardized Interfaces Using Oracle Foundation Packs

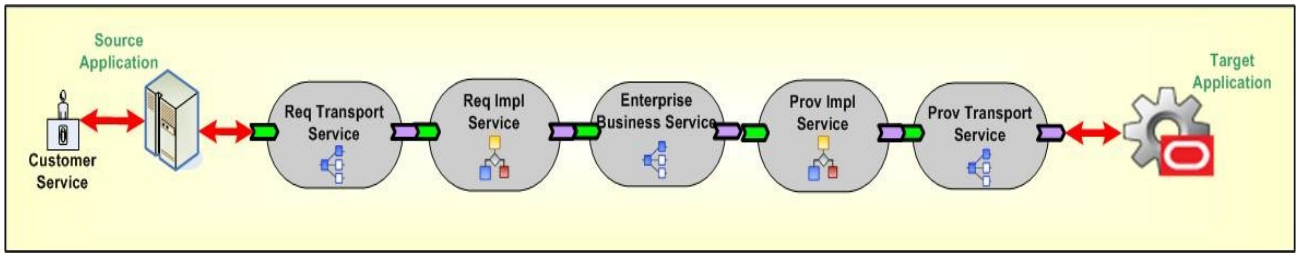
Loose-coupling through a canonical (application-independent) model is a characteristic of a true SOA. Participating applications in loosely coupled integrations communicate through a virtualization layer. Instead of direct mappings between data models, transformations are used to map to the canonical data model. While this allows for greater reusability, the transformations both increase the message size and consume more computing resources. For functional integrations, this is the ideal integration pattern because the reusability gained is worth the slight overhead cost.

In this case, an Enterprise Business Service (EBS) based on relevant Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) is created as a mediator service.

A provider service, exposing a coarse-grained functionality of the provider application leveraging one or more APIs, is created with the same EBM interface as the EBS operation interface.

If the business initiators cannot present the request in the format understood by the EBS operation interface, a requester service is used to transform the business initiator request to the provider service format.

The following diagram illustrates how the request sent by the source application is processed by target application with the help of the EBS and a set of intermediary services. Note that the request as well as provider transport services are optional. They are needed only in the case of non-SOAP-based transports.




### Example using canonical model-based virtualization

In the case of more complex situations in which the Integration Flow involves interactions with multiple applications, the requester application-specific AIA service presents its request to the mediator AIA service. The mediator AIA service triggers an AIA service that implements a workflow-like capability and manages all interactions with the provider application-specific AIA services through mediator AIA services.

In this case, it is assumed that the mediator AIA service interface chosen is accepted as a common interface. Thus, all requester application-specific AIA services invoke this mediator AIA service and all provider application-specific AIA services implement this common interface. The AIA service artifacts that need to be developed are determined by the complexity of the data exchange and the various message exchange patterns that are involved.

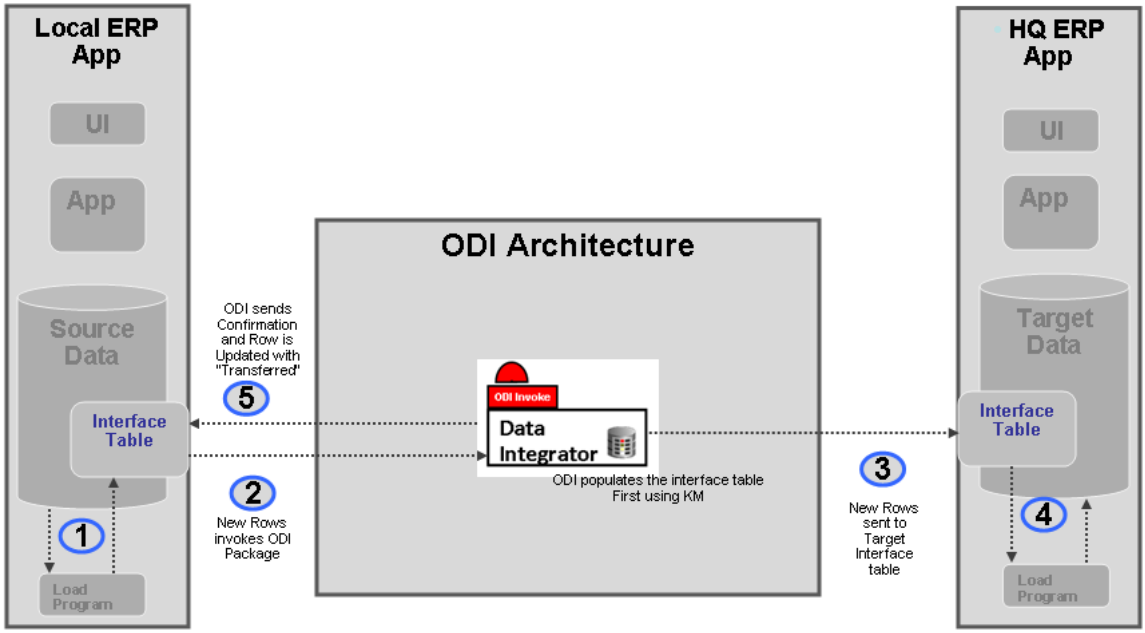
## 1.4.5. Describing Bulk Data Integration with an Extra, Transform, and Load Approach Using Oracle Data Integration Suite

Bulk data processing involves a large batch of discrete records or records with very large data sets. The methodology used for this integration is a point-to-point integration specializing in the movement of data with high performance and a reusability trade-off.

### 1.4.5.1. Integrations for High-Volume Transactions without an Xref Table

For use cases in which storing cross-reference (Xref) data for high-volume transactions does not make sense, AIA recommends using a point-to-point integration using Oracle Data Integrator.

An example of such a use case is a retail store chain headquarters that receives business transaction data from individual retail stores at the end of each day after each store closes. Depending on the time zone where each store is located, they start sending their day's worth of business transactions to headquarters. In this scenario, there is no need to store Xref data between each individual local store and headquarters because there will not be any data manipulation language (DML) operations on such data sets.




High Volume Transactions Without Xref

Example of a high-volume transactions integration without the use of an XREF table

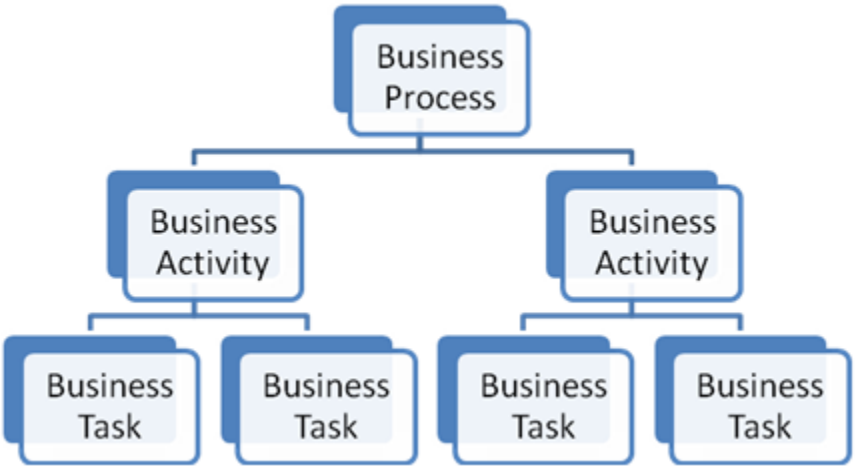
The steps to load data are the same as those for using an Oracle Data Integrator package. There is no AIA component in this architecture. Local Enterprise Resource Planning (ERP) applications load their interface table and invoke an Oracle Data Integrator package to send data to the headquarters interface table. Once the data is processed, the Oracle Data Integrator updates the local ERP application's interface table with a status of Transferred or Processed for each row.

# 1.5. Describing AIA Reference Process Models

AIA Reference Process Models are collections of best practices from various Oracle application portfolios. These models are delivered by AIA as Oracle Business Process Analyzer content.

Reference Process Models serve the following purposes:

- Help categorize Business Processes into Business Activities and Business Tasks.
- Build a repository of reusable Business Activities and Business Tasks
- Establish key performance indicators
- Aid in identifying equivalent AIA service artifacts described as part of the AIA service inventory




Relationships between Business Processes, Business Activities, and Business Tasks

## 1.5.1. What is a Business Process?

A set of activities and tasks accomplished in a particular business area are treated as a business process. Examples include Sales Management, Inventory Management, and so on. In themselves, they do not deliver business value. Business value is realized when business processes from different business areas are coordinated.

## 1.5.2. What is a Business Activity?

A Business Activity is a coordinated set of Business Tasks. A Business Activity can be considered to be equivalent to a subprocess. Examples include Process Payment, Ship Goods, and so on. Business activity functionality is provided by one application or a combination of applications.

### 1.5.3. What is a Business Task?

A Business Task is an elementary activity or atomic process capable of handling a unit of work. It cannot be split further without a loss of business meaning. It can be implemented by one or more providers. Examples include Create Customer, Query Payment, and so on.

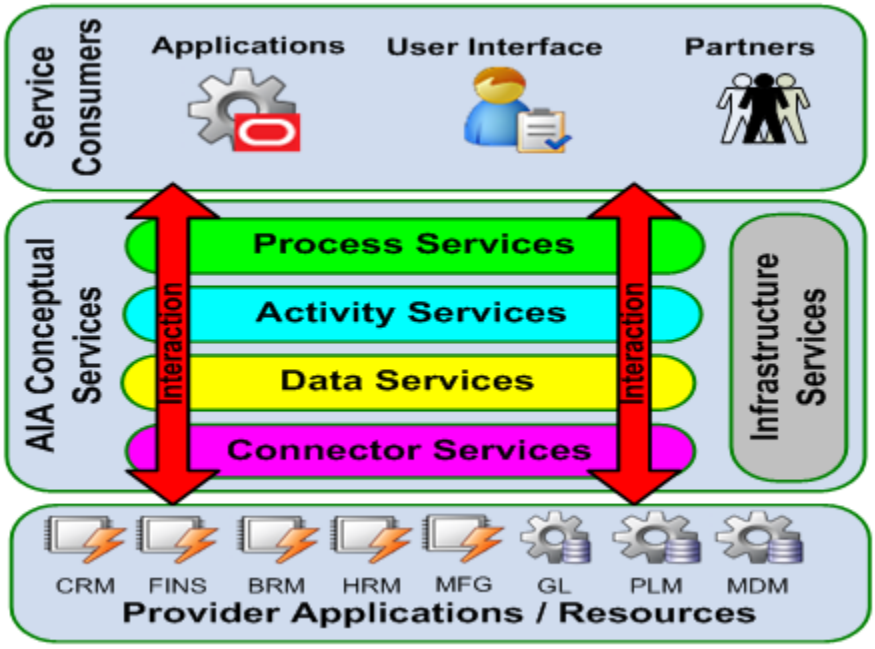
### 1.5.4. What is a Composite Business Flow?

A Composite Business Flow is a set of coordinated tasks and activities, involving both human and system interactions across one or more business areas that will lead to accomplishing a set of specific organizational goals. Characteristics of Composite Business Flows include the following:

- Large, complex, and long running
- Widely distributed and customized
- Dynamic
- Automated
- Both business-oriented and technical in nature
- Cross boundaries within and between businesses
- Dependent on and supportive of human intelligence and judgment
- Difficult to recognize

## 1.6. Describing a Conceptual View of AIA

This diagram depicts the various concepts of AIA:




Conceptual view of AIA

This section discusses the following topics:

- What is a Service Consumer?
- What are AIA Conceptual Services?
- What are Provider Applications and Resources?

## 1.6.1. What is a Service Consumer?

A Service Consumer is the initiator of a business process, business activity, or business task in an enterprise. It has knowledge of the available AIA Conceptual Services and has the ability to present requests accordingly.

## 1.6.2. What are AIA Conceptual Services?

AIA Conceptual Services are developed using Oracle Fusion Middleware technologies. They constitute the service portfolio for the SOA implementation and enable the following:

- Reuse, granularity, modularity, composeability, componentization, and interoperability
- Standards-compliance (both common and industry-specific)
- Service identification and categorization, provisioning and delivery, and monitoring and tracking

AIA Conceptual Services are categorized as follows: Process Services, Activity Services, Data Services, Connector Services, and Infrastructure Services.

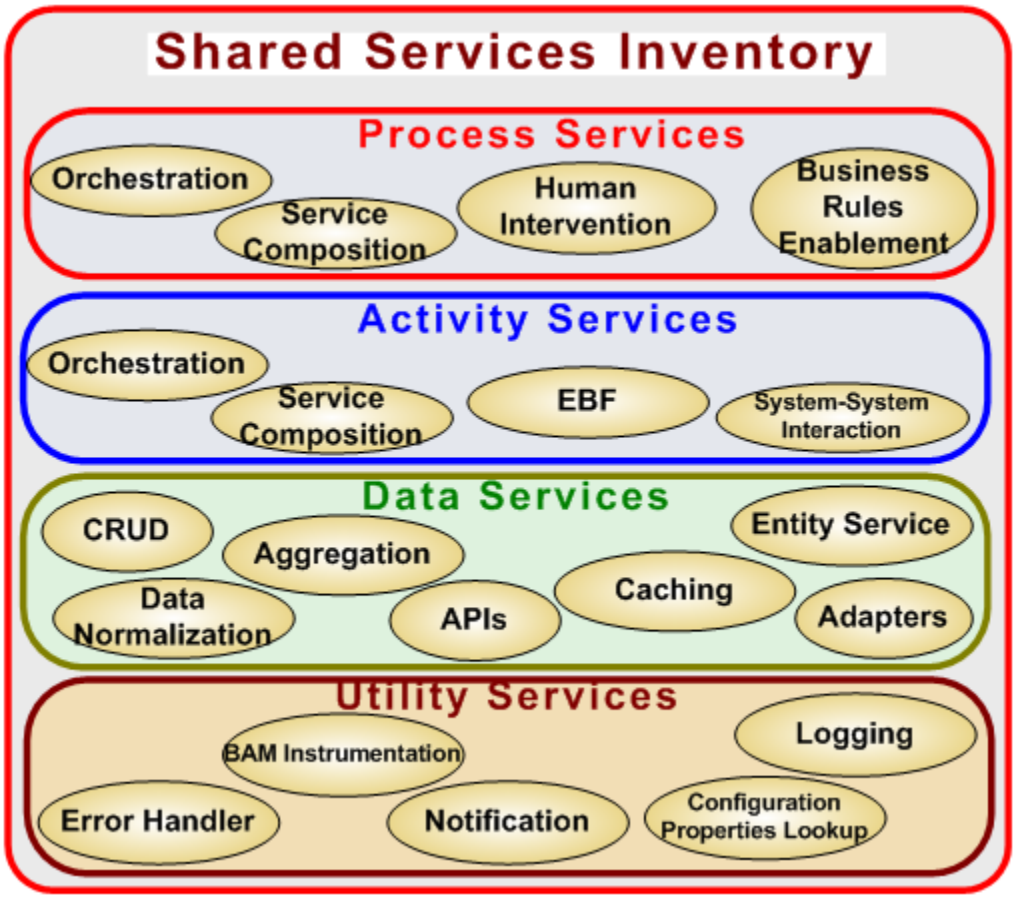
### 1.6.3. What are Provider Applications and Resources?

Various applications acquired for their best-of-breed functionalities and any legacy applications built in-house are Provider Applications and Resources. Each of these applications exposes business functions, such as APIs, and can be accessed using different modes of connectivity.

## 1.7. Describing the AIA Shared Service Inventory

Shared Services are application-agnostic services leveraged to build cross-application Composite Business Processes (CBPs). A successful AIA implementation is dependent on a robust Shared Services Inventory.

The various categories described in this section help to relate Shared Services to Reference Process Model artifacts. This diagram depicts elements in the Shared Services Inventory:




Elements in the Shared Services Inventory

## 1.7.1. What is a Process Service?

A Process Service is the implementation of major business events in an enterprise that have a significant impact on running the enterprise. It involves work being accomplished with the help of multiple resources.

Process Services automate business processes, orchestrate a series of human and automated steps, and normally span multiple information systems. They are analogous to Business Processes in AIA Reference Process Models.

The primary purpose of Process Services in AIA is to define and automate business processes that are external to and independent of the specific backend systems used in the organization. This shields the business process from backend system changes. Similarly, the applications are isolated from business process changes. Loose-coupling between the business processes and the applications simplifies changes and maintenance for business processes, as well as applications.


Process Services

The structure of the information packets that are passed around should be rich enough to capture all of the significant event details. These can be custom-defined or accepted canonical messages, and be independent of the enterprise applications. Process Services leverage the Activity Services and Data Services and are triggered by significant business event initiators, such as CRM, UI applications, and business-to-business (B2B), for example.

## 1.7.2. What is an Activity Service?

An Activity Service represents an atomic business unit of work and has a set of steps involving system-to-system interactions. Activity Services may also warrant orchestration. In some cases, there may be matching application capabilities. These are exposed as mediator services on the service bus. Activity Services can act upon multiple canonical messages and be consumed by participating applications and process services. The structure of the message will either be canonical or a user-defined format.

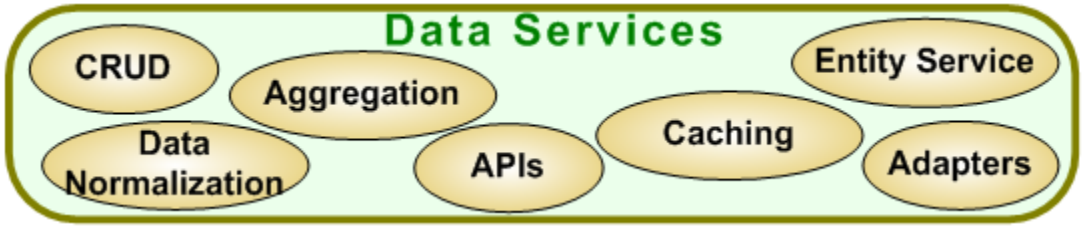

Activity Services

Activity Services are analogous to the Business Activities and Business Tasks in the AIA Reference Process Models. In some cases, multiple lower-level operations are combined to form an Activity Service, thereby hiding the complexity of the lower-level operations.

## 1.7.3. What is a Data Service?

A Data Service provides an aggregated, real-time view of enterprise data. Data consumers interact with enterprise data via Data Services. Data Services are primarily create, read, update, and delete (CRUD) operations that act upon canonical or user-defined messages. They are exposed on the service bus as mediator services.

Data Services eliminate point-to-point links at the data level and direct dependency on the data models of data sources. Data Services can be consumed by participating applications, process services, and activity services.




Data Services

Data Services are analogous to Business Tasks in AIA Reference Process Models.

## 1.7.4. What is a Utility Service?

A Utility Service provides error handling, diagnostic, and logging facilities across AIA implementations.


Utility Services

## 1.7.5. Implementing Process Services

The CBP AIA service artifact is an implementation of a Process Service.

### 1.7.5.1. Common Scenarios for Process Services

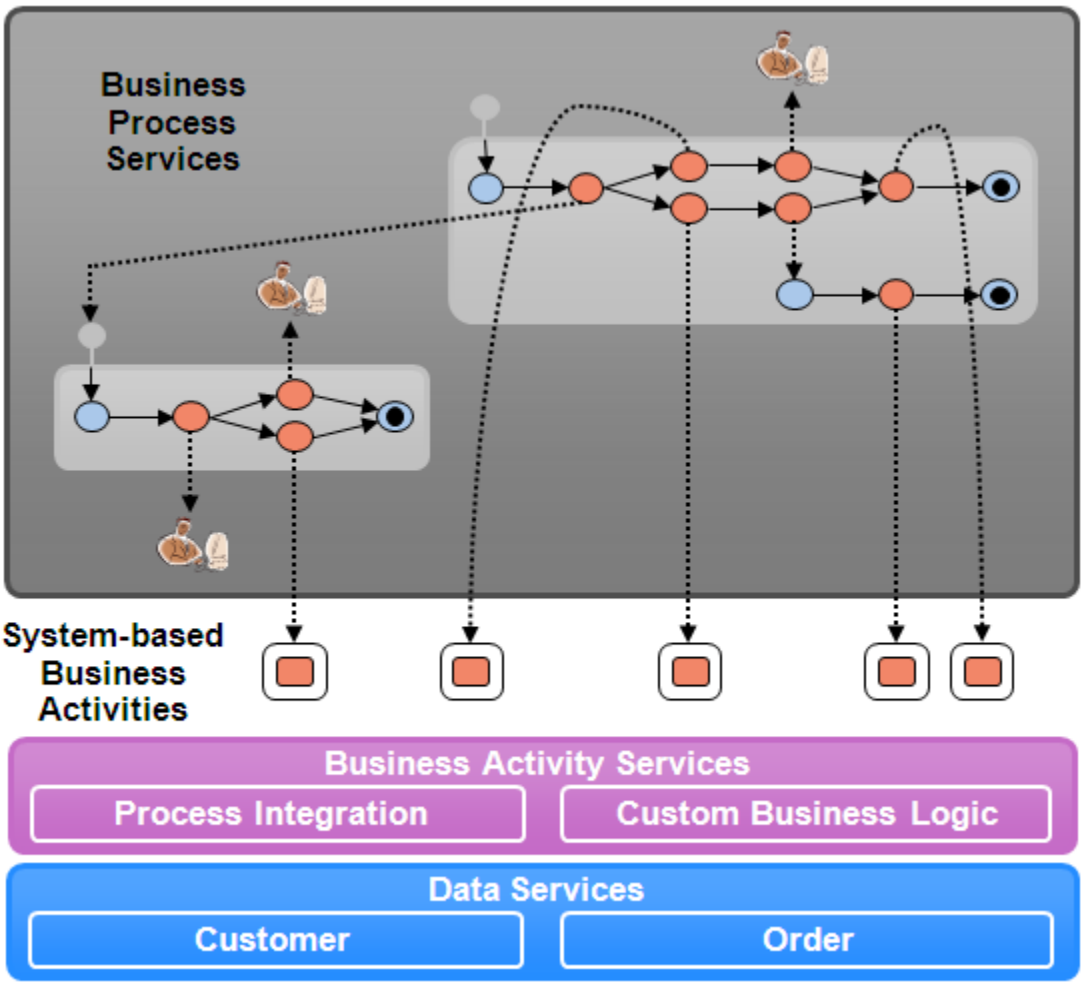
Process Services are analogous to Business Processes in AIA Reference Process Models, so they have the same goal: enabling structured accomplishment of work in an enterprise to ensure consistency and efficiency with low cost.

A Reference Process Model Business Process consists of a set of business activities and business tasks. There is a judicious mix of automated and manual tasks.

Process Services leverage technology and provide implementation software that can be executed on the application server. They leverage activity services and data services.

Process Service examples:

- **Resolve Customer Complaint Process Service:** A business process defined to resolve complaints could:
  - Analyze a complaint and identify activities.
  - Schedule call, block resources, and assign technician.
  - Track activities and send statuses.
  - Close the complaint, generate metrics, and update statuses.
- **Settle Auto Accident Claim Process:** A business process to settle auto accident claims could:
  - Analyze the claim and identify the parties and insurance agencies involved.
  - Schedule appointments and assign agents for collecting facts.
  - Compare facts collected with those from other agencies and attribute fault and settlement.
  - Send for approval.




Process Service implementation

## 1.7.6. Implementing Activity Services

The EBS AIA service artifact is the implementation of an activity service, if the EBS operation invokes an Enterprise Business Flow (EBF). In very few situations, an application might directly expose the needed functionality. In this case, an EBS operation may call an Application Business Connector Service (ABCS).

### 1.7.6.1. Common Scenarios for Activity Services

Activity Services are analogous to Business Activities in AIA Reference Process Models, so they have the same goal: providing a structure to accomplish work with the help of granular business tasks. Typically, an orchestration flow is used to implement Activity Services. The orchestration flow strings together a set of business tasks. Activity Services can leverage other activity services and data services.

Activity Service examples:

- **Creation of Customer in Billing Systems:** An activity service can be implemented to create a customer in a billing system using an order document. This service may be composed of the following steps:
  - Retrieving set of account identifiers from an order document.
  - Retrieval of account particulars from a CRM system.
  - Creation and updating of customers in a billing system.

### 1.7.7. Implementing Data Services

The EBS AIA service artifact is an implementation of a data service.

#### 1.7.7.1. Common Scenarios for Data Services

Data Services are analogous to Business Tasks in AIA Reference Process Models, so they have the same goal: providing a structure to accomplish work with the help of application provider services. Data Services leverage application provider services.

## 1.8. Describing AIA Service Artifacts

This section discusses the following topics:



### 1.8.1. What is a Composite Business Process?

A CBP is a set of coordinated tasks and activities involving both human and system interactions. It is an implementation of the AIA Reference Process Model Business Process and is a Process Service.

In AIA, CBPs are implemented, managed, and monitored as a single SOA composite using SOA BPEL along with SOA Mediator, SOA Human Workflow, and SOA Business Rules components. The BPEL components are used to drive the flow.

### 1.8.2. What is an Enterprise Business Service?

An EBS is a first-class object exposed on the enterprise service bus. An EBS is coarse-grained and performs a specific business activity or business task and is either an Activity Service or Data Service.

For example, the activity it performs could be one of the following:

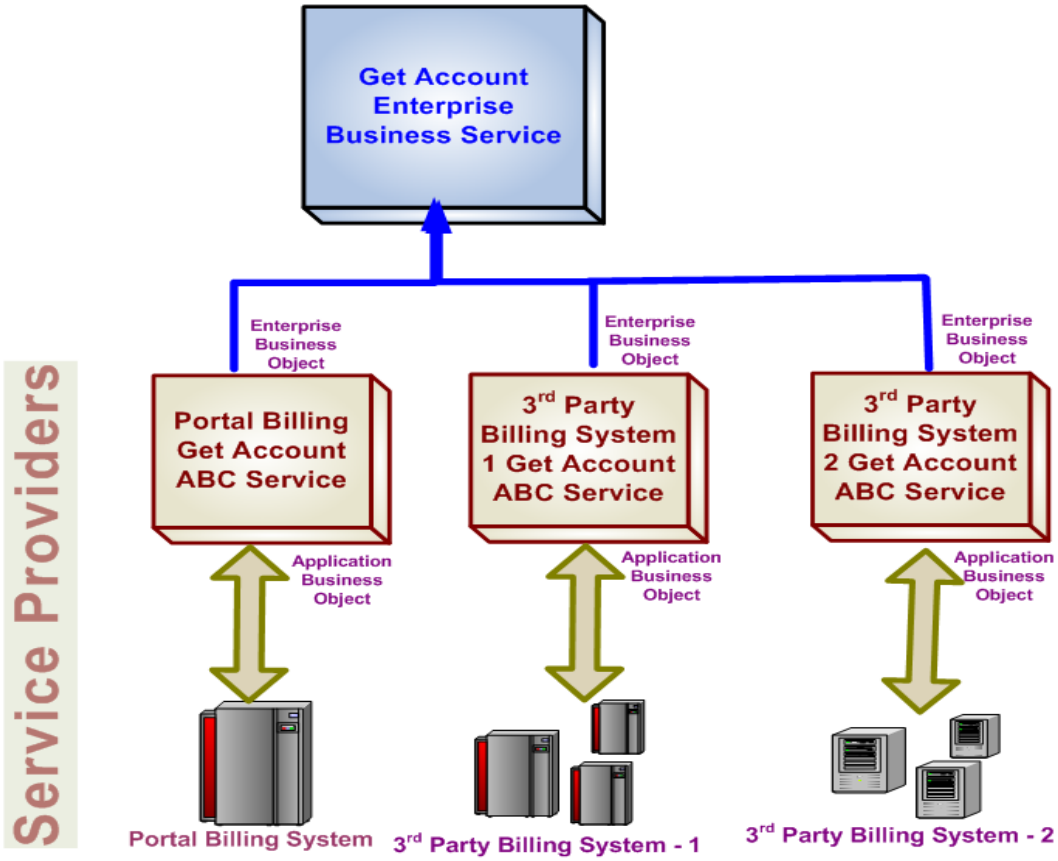
- Creating an account in a billing system.
- Checking for the presence of an account in a billing system.
- Getting the balance details for an account from a billing system.

An EBS is agnostic of a specific backend implementation. Any backend implementations that can support the interface standards defined by an EBS can be automatically considered as service providers. EBSs expose coarse-grained, message-driven interfaces for the purpose of exchanging data between applications, both synchronously and asynchronously.

The salient features of EBSs include:

- Ability to reuse available assets in the Oracle portfolio.
- Ability to substitute a service provider with another without any impact to the client.
- Content-based selection of the service provider.

The following diagrams show how an EBS enables the loose-coupling of requesters with actual service providers.




An EBS enabling the loose-coupling of requesters with actual service providers

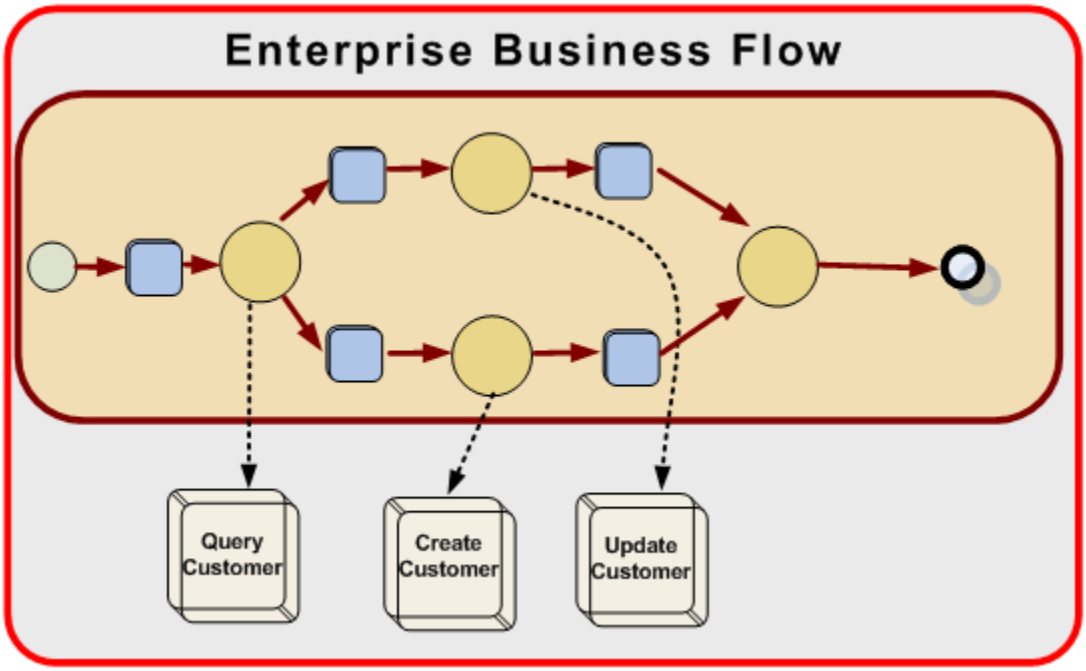
## 1.8.3. What is an Enterprise Business Flow?

An EBF is used to implement a business activity or a task that involves leveraging capabilities available in multiple applications. An EBF strings together a set of capabilities available in applications to implement a coarse-grained business activity or task and composes a new service that leverages existing capabilities. An EBF involves only system-to-system or service-to-service interactions and does not include any activity that requires human intervention.

In a canonical integration, the EBF is an implementation of an EBS operation and calls other EBSs. An EBF will never directly call an ABCS or application. In other integration styles, the caller invoking the EBF can be an application or any other service.

An EBF can be implemented using Oracle Service Bus for stateless coordination of synchronous or one-way business activities where BPEL is used to coordinate asynchronous activities.

The following activity diagrams illustrate how some of the EBFs in the Order to Cash Process Integration Pack are implemented to leverage existing capabilities.




Example of an EBF orchestrating the flow from a source to a target

This diagram illustrates the way in which the EBF orchestrates a flow for synchronizing customers between the source application and the target application. When the sync operation is invoked from the source application, the EBF first determines if the customer already exists in the target application. If so, it will update the customer in the target application. If not, it will create the customer in the target application.

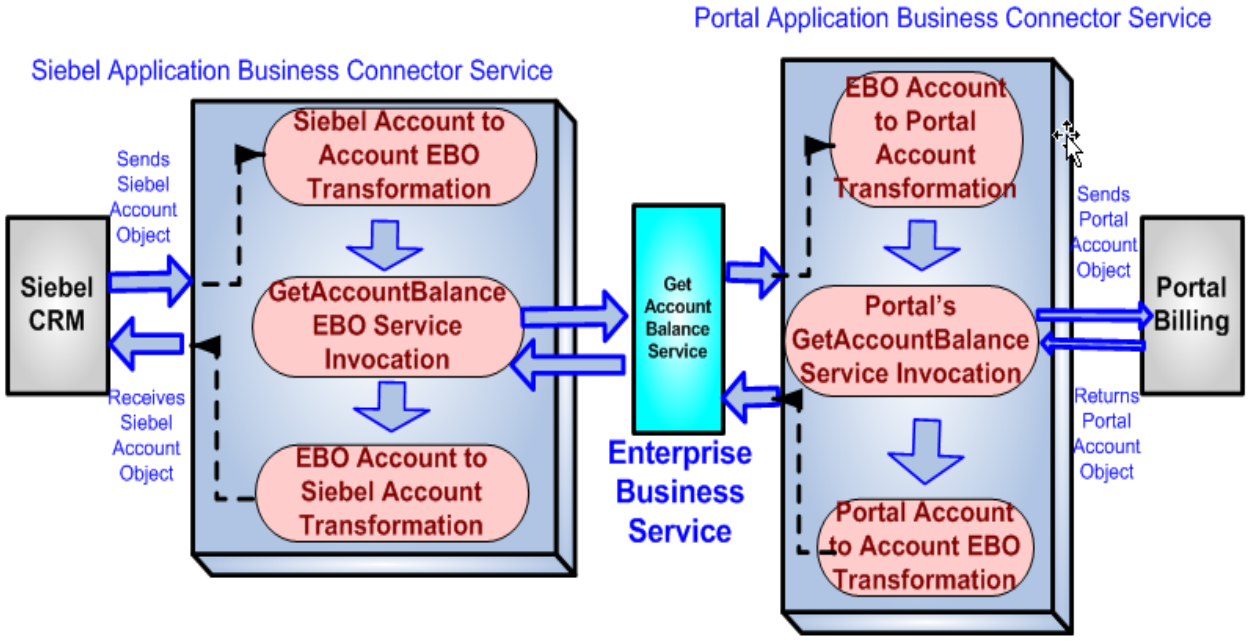
## 1.8.4. What is an Application Business Connector Service?

An ABCS implements the EBS interface by exposing the business functions provided by the participating application.

The ABCS is responsible for the coordination of the various steps provided by a number of business functions exposed by a provider application. For example:

- Validation
- Transformations: message translation, content enrichment, and normalization
- Invocation of application functions
- Error handling and message generation
- Security

**Siebel CRM Application querying account balance from Portal Billing application**




## ABCS implementations

# 2. Understanding Enterprise Business Objects and Enterprise Business Messages

This chapter introduces Enterprise Business Objects (EBOs) and explains what an EBO is and why you need an EBO to facilitate an integration. The chapter then introduces Enterprise Business Messages (EBMs) and discusses the architecture and usages as well as how the context-specific views of the EBO can be created.

This chapter discusses:

- EBOs
- EBMs
  - EBM architecture
  - EBM headers

## 2.1. EBOs

An EBO is the definition for a standard business data object and is composed of reusable data components. The library of all EBOs makes up a data model. The EBO represents a layer of abstraction on top of the logical data model and is targeted for use by developers, business users, and system integrators. In the integrations developed using Oracle Application Integration Architecture (AIA), the EBO data model serves as a common data abstraction across systems. It supports the loose coupling of systems in Oracle AIA and eliminates the need for one-to-one mappings of the disparate data schemas between each set of systems. The adoption of the EBO facilitates the mapping of each application data schema only once to the EBO data model. This significantly minimizes the manual coding for data transformation and validation because it eliminates the need to map data directly from one application to another.

EBOs have the following characteristics:

- They contain components that satisfy the requirements of business objects from the source and target application data models.
- EBOs differ from other data models in that they are not data repositories.

  Instead, they provide the structure for exchanging data. XML provides the vocabulary for expressing business data. The XML schema is an XSD file that contains the application-independent data structure to describe the common object.
- Each EBO is represented in an XML schema (XSD) file format.

An EBO represents business concepts such as a customer, a sales order, a payment, and so forth. Each EBO has a primary business component that identifies the object, and optionally multiple supplementary components required to complete the definition of the EBO. Sales Order Header and Purchase Order Header are examples of primary business components; Sales Order

Line and Sales Order Schedule are examples of supplementary business components. The following sections describe the various components that form the EBO.

### Business Component

An EBO business component is a subset of an EBO that has complex content (many properties) and exists only within the context of the EBO. These components are the high-level building blocks of any EBO.

Examples include Sales Order Header and Invoice Line. Each EBO is built from one or more EBO business components.

An EBO business component may have a primary or a supplementary role in defining an EBO. Each EBO is built from one or more EBO business components. Business components that can be used across various context-specific definitions for a single EBO are defined within the EBO schema module.

### Common Components

These are reusable common components that are used by many EBO business components. A common business component is a subset of an EBO business component that has complex content (many properties). Examples of common business components include concepts such as tax, charge, status, address, and so forth. Generally speaking, the content within a common business component is complete enough to both identify and define the component. This implies that applications could use the common components associated with an EBO to create or update application objects. For example, Address is a common business component that is used by many different EBOs. The content model of Address will contain sufficient information for an application to identify as well as create and update the address as necessary when the address is supplied as part of an EBO's content. Note that for this to happen, the application that creates the EBO instance must ensure that the address information is complete. No one-to-one relationship exists between a data model entity such as table or a foreign key relationship to a common business component. A set of attributes or a foreign key relationship in a table could resolve to a common business component. Foreign keys by themselves could resolve to either common business components, reference business components, or other business components within the EBO definition.

A customization to one of these common objects will automatically be reflected in all EBOs that reference that object. An example would be an Address definition type. If your implementation requires customizing this address format by adding a third address line, the modification of the Address definition type automatically affects the addresses referenced in EBOs. This design philosophy significantly reduces the design, development, and maintenance of common objects.

Components that are applicable to all EBOs are defined in a common components schema module.

### Reference Components

A reference business component shares the same identification properties as the corresponding EBO business component (that is, both the EBO business component and its associated reference business component can be identified by the same attributes), but additionally includes a minimal subset of attributes required to qualify the EBO business component. PurchaseOrderLineReference and InvoiceLineReference are examples of reference business components. Note that reference business components by definition are not meant to contain all the attributes necessary to define an EBO—they are expected to contain at the least all the attributes necessary to identify a supplementary business component. Beyond this, they generally contain additional attributes that help consumers of an EBO to better understand and interpret the EBO instance that uses the reference business component.

Wherever possible and practical, the EBO leverages widely adopted international standards for modeling and naming, such as the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Core Components Technical Specification (CCTS), UN/CEFACT XML Naming and Design Rules (NDR), Open Applications Group Interoperability Standard (OAGIS), and ISO 11179.

Apart from creating the complete definition for an EBO, a definition is also created for each of the contexts in which this EBO will be used.

## 2.2. EBMs

At the most basic level, EBMs are the messages that are exchanged between two applications. The EBM represents the specific content of an EBO needed for performing a specific activity. For example, an invoice might be used in three contexts: add, cancel, and update. The context for processing the invoice might warrant the presence of almost all of the elements present in the EBO; however, canceling the invoice might need to identify only the invoice instance to be canceled.

The context-specific EBM definitions are created by assembling a set of common components and EBO-specific business components. In some scenarios, the business components can be obtained from more than one EBO. These context-specific EBO definitions are then used in the appropriate context-specific EBMs. In this scenario, the process-specific invoice definition would be part of the *ProcessInvoice* EBM and the cancel-specific invoice definition would be a part of the *CancelInvoice* EBM. These EBMs can be used either as the request or response parameters.

The definitions for these context-specific EBMs are present in the EBM schema module. Hence, for every EBO, two schema modules are available—one containing the definition of the EBO and another containing the definition of the context-specific definitions for that EBO. In the case of the Customer Party EBO, a Customer Party EBO schema module is available as well as a Customer Party EBM schema module to represent the entire concept for the business object.

**For more information,** see Chapter 7: Understanding Extensibility.

### 2.2.1. EBM Architecture

Every EBM possesses the same message architecture. An EBM encompasses details about the action to be performed using the data, one or more instances (EBOs) of the same type, and the EBM header. Each service request and response is represented in an EBM by using a distinct combination of an action and an instance. For example, a single Query Customer Party EBM business document sends the request to a billing system for retrieving account details for one or several customer accounts. You can accomplish this by using a single *Query* action and several Customer Party instances. The billing application can respond to this request by sending a *Query Customer Party Response* EBM business document that comprises the Query Response action and Customer Party instances, which are populated with details.

The EBM cannot process details about more than one type of action. For example, you cannot have a *Query* and *Update* action in the same message.

When using EBMs, consider the following points:

- Application interdependencies.

Any application invoking the Enterprise Business Services (EBSs) will have to generate the EBM to pass the EBM as a payload to the EBS.

- The action in the EBM identifies the action that the sender or the requester application wants the receiver or provider application to perform on the EBM.

  The action also stores additional information that must be carried out on the EBO instance. For example, the Create action may carry information about whether it wants the target application to send a confirmation message. The Query action may carry information about the document header section of the original EBM that resulted in the performance of this action.

- The business object portion of the data area element contains the business object data element definitions that can or must be sent in the message.

  This is the content that is carried from one point to another. The element reflects the action-specific view of the EBO.

- An EBM can be defined to carry multiple instances. Only the actions that support bulk processing will use EBMs that support multiple instances.
- The information that is present in an EBM header is common to all EBMs.

  The information that is present in the data area and the action are very specific to a particular EBM.

- The message architecture is detached from the underlying transport protocol.

  Any transport protocol such as HTTP, HTTPS, SMTP, SOAP, and JMS should be able to carry these documents.

## 2.2.2. EBM Headers

The EBM header is an integral part of every EBM. You can consider the EBM header as a wrapper or an envelope around transactional data messages. It comprises representations of functional data such as Document Identification, Involved Parties (Sender, Provider, intermediary services, Security, and Transaction Rules [Transaction State and Exceptions]).

The EBM header provides the ability to:

- Carry information that associates the message with the originator.
- Uniquely identify the message for auditing, logging, security, and error handling.
- Associate the message with the specific instance of the sender system that resulted in the origination of the document.
- Store environment-specific or system-specific information.

The requirements pertaining to infrastructure-related services such as auditing, logging, error handling, and security necessitate the introduction of additional attributes to the message header section of the EBM.

**For more information,** see *Oracle Application Integration Architecture Foundation Pack: Development Guide,* "Working with Message Transformations."

# 3. Understanding Enterprise Business Services

This chapter introduces Enterprise Business Services (EBS) and explains why EBSs are needed to facilitate an integration. The chapter then discusses the types of EBSs as well as the types of operations that can exist for these services. Finally, it provides a high-level overview of the tasks involved in the creation of enterprise business services.

This chapter discusses:

- EBS
- EBS operations
- Verbs
- EBS types
- EBS architecture
- Enterprise Business Flow (EBF) processes
- EBS implementation
- EBS message exchange patterns

**For more information** about EBS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing and Developing Enterprise Business Services."

## 3.1. EBS

EBSs are the foundation blocks in the Oracle Application Integration Architecture (AIA). EBSs represent the application or implementation-independent web service definition for performing a business task. The architecture facilitates distributed processing using EBS.

An EBS is a service interface definition, currently manifested as an abstract Web Service Definition Language (WSDL) document, which defines the operations, message exchange pattern, and payload that are applicable for each operation of a service.

An EBS is self-contained; that is, it can be used independent of any other services. In addition, it can also be used within another EBS. Because EBSs are business-level interfaces, they are standard service definitions that can be implemented by the applications that want to participate in the integration. EBSs are generally coarse-grained and typically perform a specific business activity such as creating an account in a billing system or getting the balance details for an account from a billing system. Each activity in EBS has a well-defined interface described in XML. This interface description is composed of all details required for a client to independently invoke the service.

These services expose coarse-grained, message-driven interfaces for the purpose of exchanging data between applications, both synchronously and asynchronously. The request-specific and response-specific payload for each of the services is defined as an Enterprise Business Message (EBM). The EBM typically contains one or several instances of a single Enterprise Business Object (EBO), which forms the crux of the message, the action to be performed, and the metadata about the message specified in the message header section.

**For more information,** see EBM Architecture.

EBS components do not presuppose a specific backend implementation of that service. They simply provide an integration layer for the customer's choice of a backend. Regardless of the choice, customers can still achieve the seamless interaction experience in the prebuilt integrations that are delivered by AIA. Any backend implementations that can support the interface standards defined by an EBS can automatically be considered as service providers.

**For more information** about EBSs, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing and Developing Enterprise Business Services."

## 3.2. EBS Operations

An operation is a unique action that has a specific payload and results in a clearly defined, repeatable outcome.

- Each EBS contains multiple operations. A standard set of operations is defined for all Entity services. Additionally, each Entity service may have one or more nonstandard operations.
- Operations are categorized based on the *Verb* associated with the operation. *Every operation must have a Verb identified.* The Verb helps to precisely define the scope, payload, and name of the operation.
- An EBS may have synchronous and asynchronous versions of the same operation. By default, the behavior of a service operation (synchronous or asynchronous) is predetermined by the Verb associated with the operation.

**For more information,** see Verbs.

- AIA makes an explicit distinction between operations that can process a single instance of a payload versus operations that can process multiple instances of a payload. Distinct operations are provided for both cases. Only the standard operations have this distinction implemented.

## 3.3. Verbs

Every operation has a *Verb* to identify the action to be performed by the operation. The concept of a verb was originally introduced by the Open Applications Group Integration Specification (OAGIS) in their business object document (BOD) definitions and has been adopted with some modifications in the EBO definition.

Strictly speaking, the significance of a verb to identify the action to be performed by an operation is not applicable in a service-oriented web services environment because the operation definition of a web service assumes this responsibility. However, not all integrations are using web services, and message-oriented integration scenarios still exist that may require the processing action to be identified within the message.

Verbs are also critical to define the semantics of the operation to be performed and to provide a consistent, unambiguous framework for naming operations and operation payloads.

For more information about verbs and how to use them, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Constructing the ABCS."

## 3.4. EBS Types

AIA supports two categories of EBSs, entity-based services and process-based services.

### 3.4.1. Entity Services

All of the standard activities that need to be performed using an EBO are brought together under an entity service. Hence, every EBO has a corresponding EBS, which has definitions for performing standard activities such as Create, Update, Delete, Query, and so on. Sample EBSs include Customer, Party, Item, Sales Order, and Installed Asset.

The standard activities in an entity-based EBS are:

- Create

  To create an object instance

- Update

  To update an object instance with only the changes that occurred

- Delete

  To delete an object instance

- Query

  To retrieve details about an object

- Sync

  To send a current snap shot

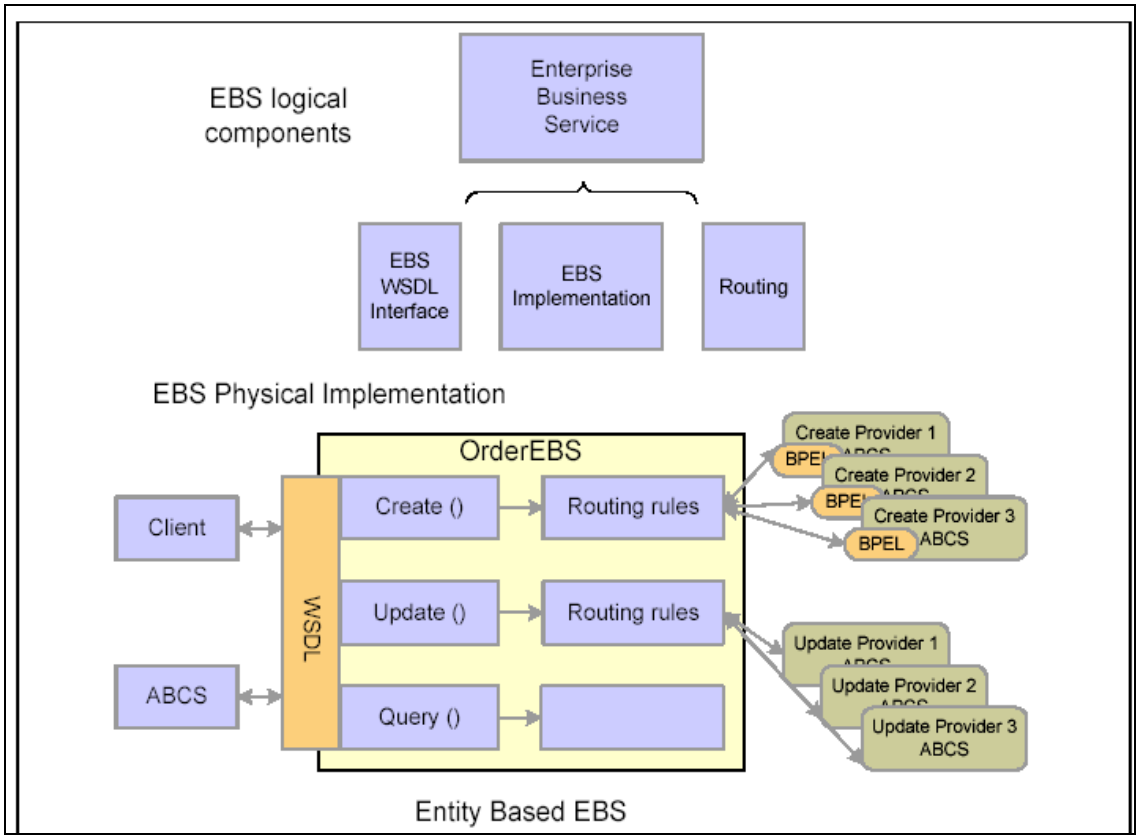
Each of the activities uses the relevant EBM that represents the activity-specific view of the EBO as input and output.

Entity services expose operations that act on a specific EBO.

Entity services have the following characteristics:

- Each of the business object entities has a corresponding EBS; all of the actions that can be performed on this object are exposed as service operations and are part of this EBS.
- The entity operations consist of standard Create, Read, Update, and Delete (CRUD) operations that are similar across services as well as any specialized operations that are specific to the EBO.
- Entity services are implemented as Mediator routing services.
- Entity services receive and return messages in the form of EBMs.
- Entity services leverage the context-based routing rules to delegate the request to the appropriate application-specific Application Business Connector Service (ABCS) containing the actual implementation.

This diagram illustrates the EBS logical components and EBS physical implementation of entity services.




Entity services

### 3.4.2. Process Services

All of the business activities that need to act upon a particular business object to fulfill a specific business process are brought together under a process service. For example, an EBS like OrderProcessOrchestration, Employee On Boarding might exist. The Employee On Boarding EBS might have operations such as Hire Employee, Terminate Employee, and so on that you might not find in Person EBS.

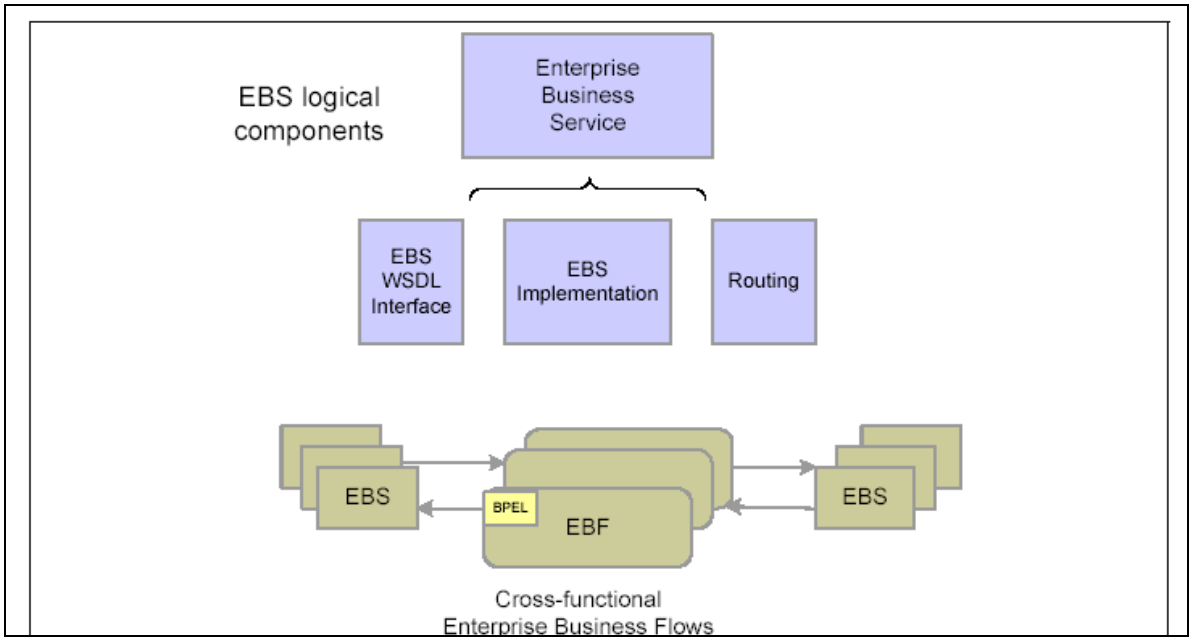
These interfaces are described using an implementation-agnostic approach. For the interface to be application-independent, the EBS expects an EBM to be passed as the input. In cases in which the EBS is expected to return a response, it will send a relevant EBM as the response. Hence, these interfaces are participating-application agnostic.

Process services expose operations related to an enterprise business process.

Process services have the following characteristics:

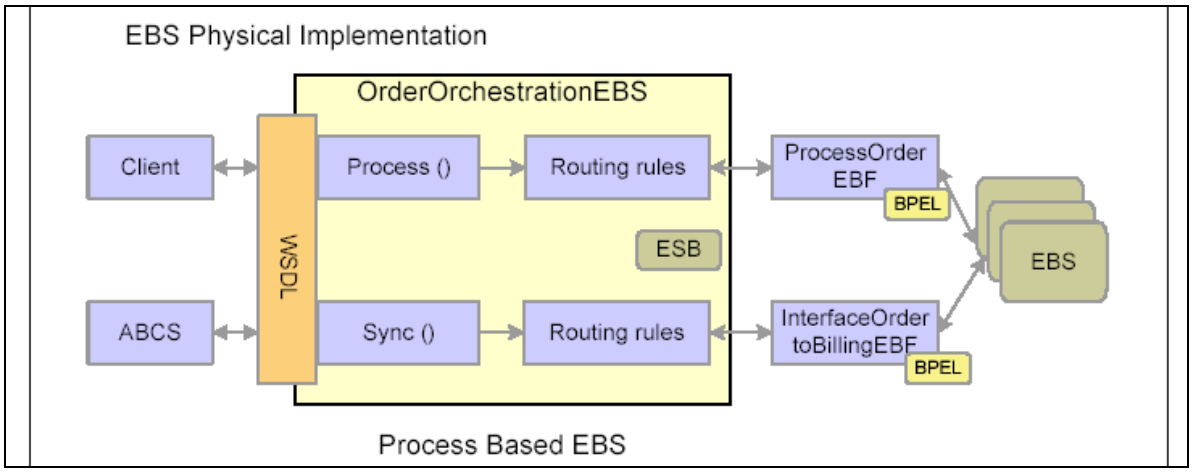
- Each EBF has a corresponding EBS; all of the actions that can be performed on this flow are exposed as service operations and are part of this EBS.
- Process services are implemented as Mediator routing services.
- Process services receive and return messages in the form of EBMs.
- Operations exposed on the EBS will initiate cross-functional Enterprise Business Flows (EBFs) that coordinate complex long-lived flows that span multiple services. These flows can interact only with EBS services. This way both the Process EBS and the related business flows are completely application-agnostic.
- Unlike Entity Services, a Process Service can act on more than one EBO.

This diagram illustrates the EBS logical components for process services.




EBS logical components for process services

This diagram illustrates the EBS physical implementation of process services.




EBS physical implementation of process services

# 3.5. EBS Architecture

The EBS architecture enables:

- Reuse of the available assets.
- Substitution of one service provider with another without any impact on the client.
- Content-based selection of the service provider.

### Reuse of Available Assets

The EBS is a web service. Like any other web service, the interface definitions for the EBS are defined in Web Service Definition Language (WSDL). The list of EBS-specific business activities (service operations), the input and output arguments for each of these service operations (input and output messages) are specified in the WSDL. As with any other web service, an EBS can be implemented using any language. The implementation takes an EBM as input and provides another EBM as output. Oracle AIA uses the Oracle SOA Suite Mediator to implement an EBS.

Even though an EBS activity such as *Create* or *Update* can be built from the ground up using web services technology, AIA takes advantage of the existing functionality in the applications. The EBS acts as a virtual service to expose the actual implementation provided by the participating application in a format that is amenable to the EBS. The intermediary services provided by the participating applications in a format amenable to the EBS are called ABCSs. The ABCS acts as the glue connecting the EBS and the participating application that is exposing the business capability. The ABCS is responsible for exposing the data access, as well as the transaction-related business functions that are available in applications as services that an EBS can invoke.

The ABCS approach does not preclude the customer from creating entirely new services for implementing the EBS.

The virtualization layer enables the following aspects of Oracle AIA:

- The physical implementation of the target service needs to be abstracted and its location needs to be hidden so that the target service (Service Provider) can eventually be replaced by some other service (endpoint virtualization).
- If the shapes of data and operations are different, data transformations and operation-to-

operation mappings are needed. This is common when old systems in existing infrastructures need to be replaced with no interruption or change to the consumers.

- The virtual service may be composed of more than one physical service, as in aggregation of services or rule-based routing. For example, a request from CA goes to the CA-Warehouse.
- You may not want to expose all operations of the actual service to the external world (partial service).
- Target service might be supporting a different transport protocol than supported by the service consumer. For example, a header transformation needs to happen from JMS to SOAP.
- If complex, content-based validations against XML are required, then Schematron should be used inside the mediator. For example, the order price must reflect the sum of prices of each order line.
- If the service consumer expects an asynchronous message exchange pattern and the service provider allows only synchronous calls, the client application invokes a virtual or proxy service (in AIA, it is an EBS) representing the target services.

**For more information,** see Understanding Application Business Connector Services.

### Substituting One Service Provider for Another

AIA enables the customer to decouple the requester's view of a service from the actual implementation. This approach increases the flexibility of the architecture by allowing the substitution of one service provider for another without the requester being aware of the change and without the need to alter the requester or EBS to abet the substitution.

For example, the delivered integration between a CRM system and a financials system may leverage a service provided by the Oracle E-Business suite. At implementation time, the customer may want to substitute this service with the one made available by another provider, such as PeopleSoft Financials. The substitution of the service provider will have absolutely no impact on the requester or the client.

This kind of decoupling is accomplished by having the requesters and service providers converse using a mediator. In AIA, the EBS publishes services to requesters. The requester binds to the EBS to access the service, with no direct interaction with the actual implementer of the service.

Hence, to achieve the loose coupling between service consumer and the provider, the EBS takes the role of mediator. Routing rules can be definitionally specified to direct the EBS regarding how to delegate the request to the right service provider as well as to the right application instance. Situations might also occur in which the services that invoke the EBS might have knowledge of whom the requests need to be delegated to. In this scenario, the services might supply the information in the EBM prior to invoking the EBS.

For example, a customer might have two billing system implementations—one dedicated to customers who reside in Northern America and the second dedicated to the customers residing in other parts of the world. The EBS evaluates this rule and deciphers the actual implementation that needs to be used for processing a specific message. Using this approach, the clients and service requesters are totally unaware of the actual implementers. Similarly, the underlying service implementers will be oblivious to the client applications that made the request.

Note that while the architecture allows for an entire message (containing all instances) to be sent to one service provider as opposed to another, it does not allow for a subset of the instances present in a message to go to one provider and the remaining instances to go to another provider.

### Content-Based Selection of the Service Provider

This architecture enables cross-application business processes to use these EBS without regard to which application vendor is actually providing the implementation or which of the multiple deployments that might exist is responsible for processing the request and providing the response.

To achieve the loose coupling, some aspects of the service interactions need to be tightly coupled between the requester and the service provider. In a service-oriented architecture (SOA), there is no way to loosely couple a service entirely between systems. One needs to choose what aspects need to be loosely coupled and what can be tightly coupled.

Oracle AIA places importance on the client being unaware of who the service provider is, the language, the platform in which the services are implemented, and finally, the communication protocol. So these aspects are totally decoupled. Making a change to one of these aspects will have no impact on the client.

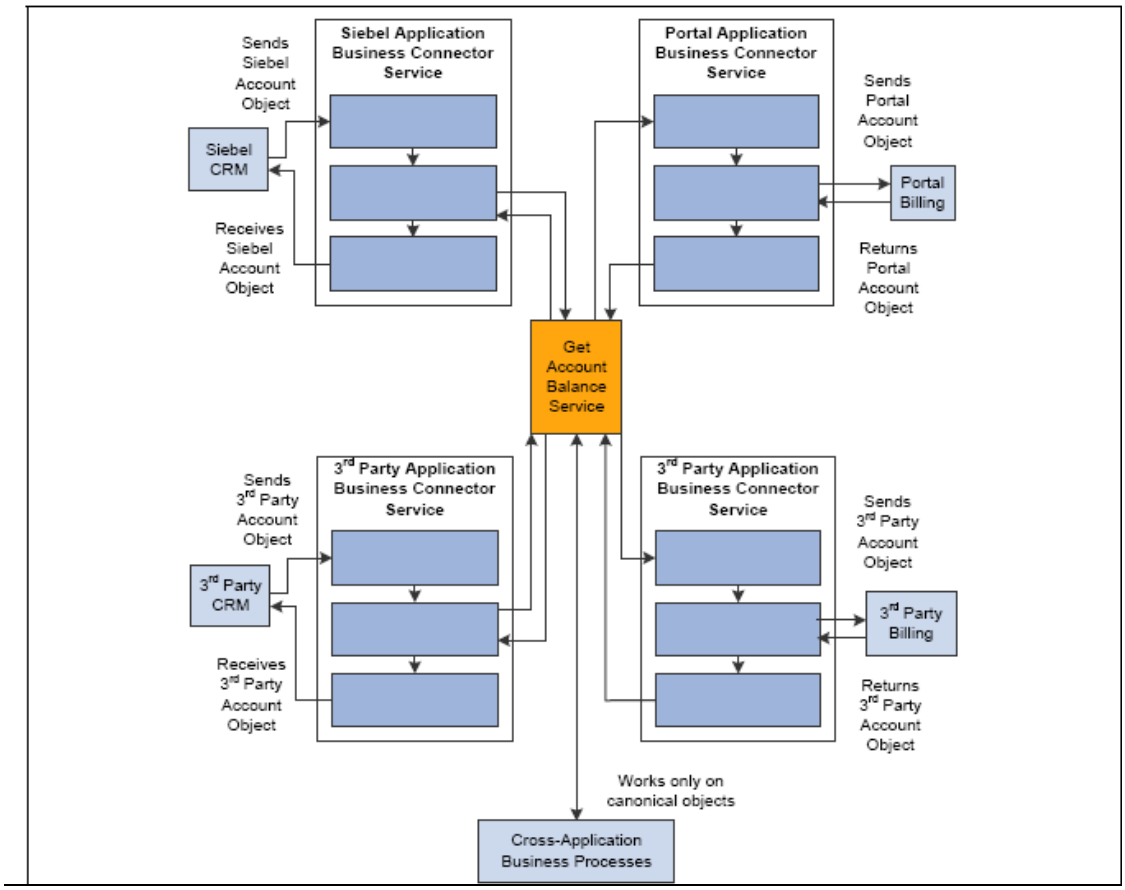
However, at design time, the requester will explicitly specify the name of the EBS operation, which in turn is responsible for identifying the actual service provider. The data formats are also specified at design time and regardless of the actual service providers, the data formats needed to pass the content will not change. So changing either the name of the EBS or the service operation or the structure of the data formats will certainly impact the client. As mentioned in the previous sections, an EBM will be passed as a request; and another EBM will be received as a response.

### EBS Purpose

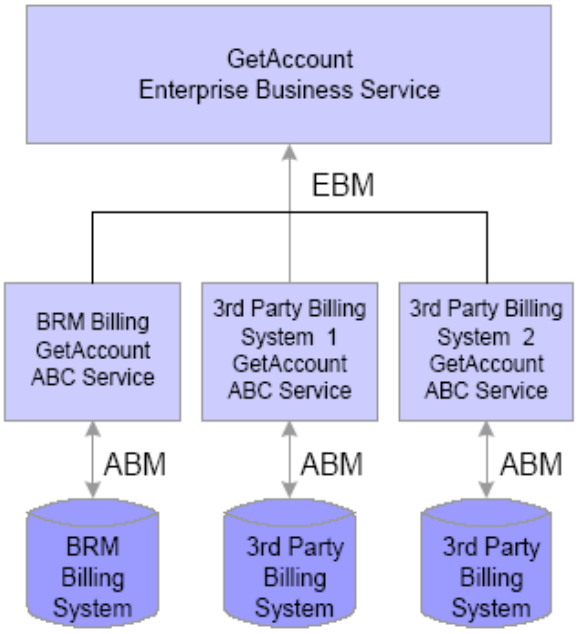
The purpose of the EBS can be summarized in the following way:

- To receive the request from the calling application.
- To identify the implementation as well as the deployment that is responsible for providing the requested service.
- To delegate the requested task to the right implementation.
- To receive the response and return the EBM to the calling application.

The following diagrams show how EBS enables the loose coupling of requesters with the actual service providers.




EBS enables loose coupling of requesters with service providers




Example of GetAccount EBS

**For more information** about EBS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing and Developing Enterprise Business Services."

## 3.6. Enterprise Business Flow Processes

Business processes define and orchestrate a series of discrete steps to complete an integration task, such as synchronizing a product across multiple applications or submitting an order from CRM to the back office for fulfillment.

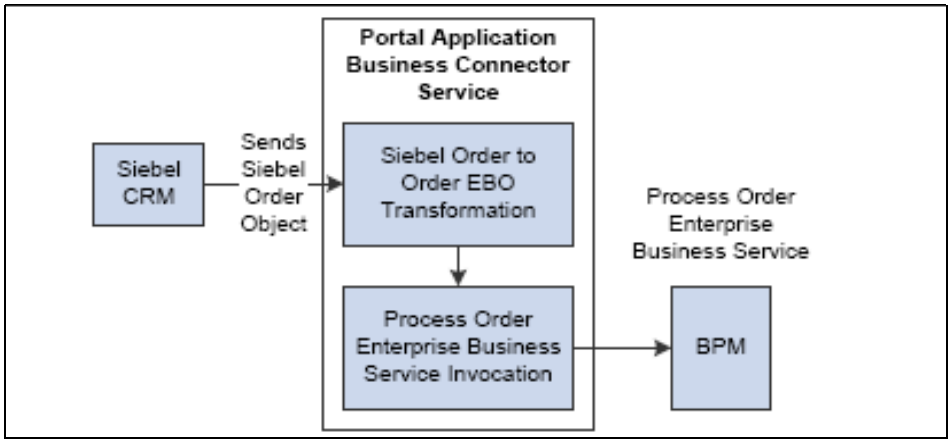
Business processes are defined independently of the underlying applications, simplifying the process of integrating applications from multiple vendors. Business processes will always use the services of the EBS.

**For more information,** see EBS.

EBSs have application-independent interfaces. They are used by BPEL processes to interact with different applications. This helps cross-application processes to be application-independent. The EBM containing the EBO is the payload of the EBS and contains business-specific messages.

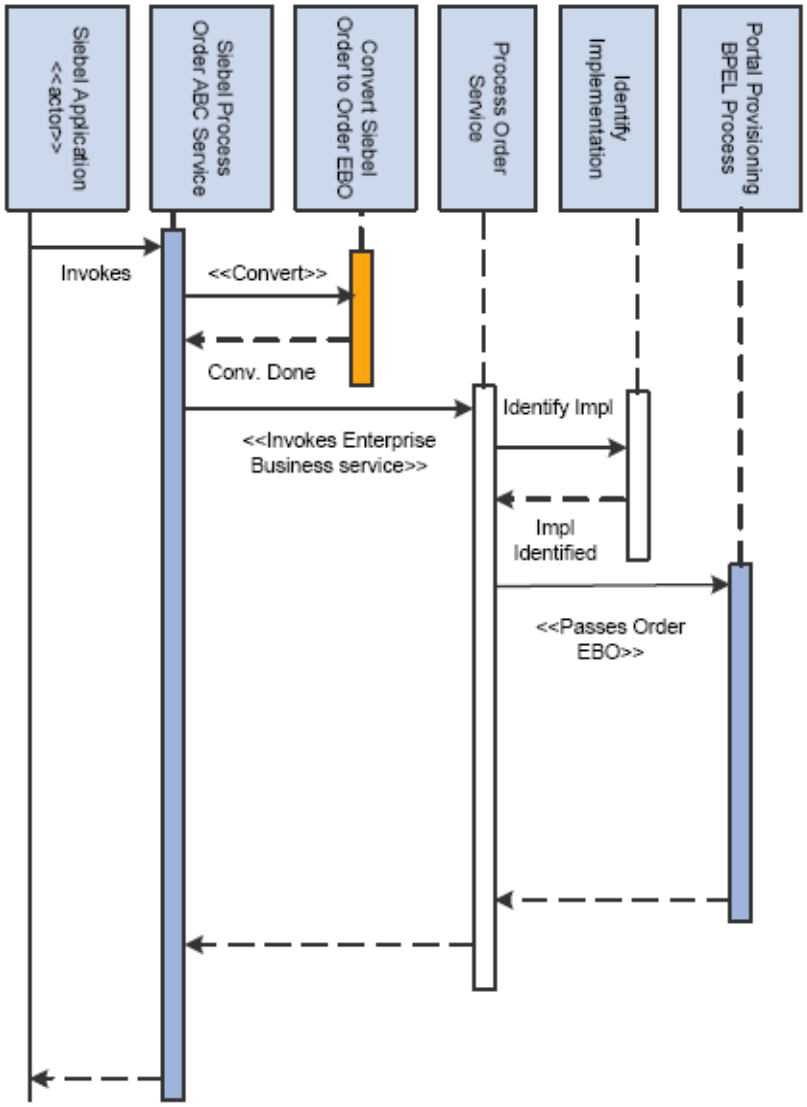
Within a cross-application business process, the EBO is used as the structure that holds the in-memory representation of the message that is sent back and forth between applications.

The following process flow diagram shows how a business process is initiated by a request coming from a participating application.




Participating application request initiates a business process

The following sequence diagram shows the occurrence of events that lead to a successful initiation of a BPEL process.




### Invocation of process order

This diagram illustrates how to keep the BPEL processes application-independent.

For the BPEL processes to remain application-independent, they should not contain any steps that are relevant to any one particular participating application.

**For more information** about EBF, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing and Constructing Enterprise Business Flows."

## 3.7. EBS Implementation

Every EBO will have an EBS. Each action of the EBO will have a service operation as its interface. The EBS is a very lightweight service. It is implemented as a Mediator routing service. Every service operation will have its own set of routing rules.

This WSDL example shows the interface definition for each of the actions that can be carried out on the Invoice EBO.

```
<portType name="SalesOrderInterface">
 <portType name="SalesOrderInterface">
 <documentation>
         <svcdoc:Interface>
      <svcdoc:Description>This interface contains operations that
can
       act upon the=>
 Sales Order object</svcdoc:Description>
      <svcdoc:DisplayName>Sales Order Interface</svcdoc:DisplayName>
      <svcdoc:Status>Active</svcdoc:Status>
         </svcdoc:Interface>
       </documentation>

       <operation name="QuerySalesOrder">
        <documentation>
         <svcdoc:Operation>
     <svcdoc:Description>This operation is used to query an Sales
Order
      object<=>
/svcdoc:Description>
      <svcdoc:MEP>SYNC_REQ_RESPONSE</svcdoc:MEP>
      <svcdoc:DisplayName>Query Sales Order</svcdoc:DisplayName>
         <svcdoc:Status>Active</svcdoc:Status>
              <svcdoc:Scope>Public</svcdoc:Scope>
            </svcdoc:Operation> </documentation>
     <input message="sordsvc:QuerySalesOrderRequestMsg"/>
     <output message="sordsvc:QuerySalesOrderResponseMsg"/>
     </operation>

          <operation name="CreateSalesOrder">
            <documentation>
              <svcdoc:Operation>
          <svcdoc:Description>This operation is used to Create an
Sales
           Order object<=
    /svcdoc:Description>
          <svcdoc:MEP>ASYNC_REQ_RESPONSE</svcdoc:MEP>
          <svcdoc:DisplayName>Create Sales
Order</svcdoc:DisplayName>
          <svcdoc:Status>Active</svcdoc:Status>
                <svcdoc:Scope>Public</svcdoc:Scope>
    <svcdoc:CallbackService>SalesOrderEBS</svcdoc:CallbackService>
    <svcdoc:CallbackInterface>UpdateSalesOrderEBM</svcdoc:Callback=>
     Interface>
               <svcdoc:CallbackOperation>UpdateSalesOrder</svcdoc:
                CallbackOperation>
             </svcdoc:Operation> </documentation>
        <input message="sordsvc:CreateSalesOrderMsg"/>
       </operation>
```

### EBS Responsibilities

Here is an overview of the responsibilities of the EBS:

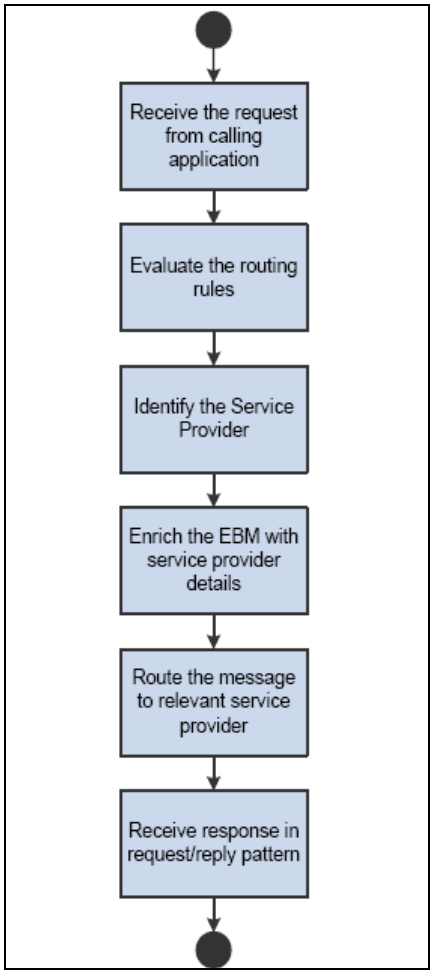
1. The EBS passes the EBM to the routing rules.

2. The routing rules evaluator applies the relevant rules to the EBM to decipher the ABCS it should invoke as well as the end-point details.
3. In scenarios in which multiple instances for a single application occur, the EBS enriches the EBM header section of the document with details about the service provider and the end-point location.

   The prebuilt integrations to be delivered by Oracle leverage the AIA Configuration Properties file to identify the end-point location.

4. The EBS routes the message to the relevant ABCS.
5. The EBS receives the response from the service provider.
6. The EBS returns the response to the calling application.

Steps 5 and 6 will occur only in the case of an integration scenario that uses the request/reply pattern. The following activity diagram illustrates the steps that are performed by an EBS:




## EBS steps

For example, clients that want to invoke the Query operation on *Customer Party* should invoke the *Query Customer Party* operation of Customer Party EBS service.

# 3.8. EBS Message Exchange Patterns

Business requirements drive the need for the use of different message exchange patterns. This section defines the various message exchange patterns that can be implemented for various operations of EBSs.

A *synchronous* operation is one that waits for a response before continuing on. This forces operations to occur in a serial order. People often say that an operation blocks or waits for a response. Synchronous operations will open a communication channel between the parties, make the request, and leave the channel open until the response occurs. This method is effective unless large numbers of channels are being left open for long periods of time. In this case, asynchronous operations may be more appropriate. Also, the synchronous pattern may not be necessary or appropriate if the end user does not need an immediate response.

An *asynchronous* operation is one that does not wait for a response before continuing. This allows operations to occur in parallel. Thus, the operation does not block or wait for the response. Asynchronous operations will open a communication channel between the parties, make the request, and close the channel before the response occurs. Message correlation is used to relate the inbound message to the outbound message. This method is effective when large numbers of transactions occur that could take long periods of time to process. In the case in which the operations are short or need to run in serial, synchronous operations may be more appropriate. The asynchronous pattern is effective if the end user does not need immediate feedback.

The EBS is modeled to have multiple operations. Each operation leads to the execution of the EBS for a particular business scenario requirement and is granular in nature. Each operation can be modeled to have the following interaction style or message exchange pattern:

- Synchronous request – response
- Fire-and-forget
- Asynchronous request – delayed response

**For more information** about design patterns, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Working with AIA Design Patterns" and "Establishing Resource Connectivity."

## 3.8.1. Synchronous Request-Response Patterns in EBSs

A synchronous request-response EBS operation pattern is synchronous in nature. The request-response has two participants.

The requester sends a request and waits for a response message. The service provider receives the request message and responds with either a response or a fault. After sending the request message, the requester waits until the service provider responds with a message. Both the request and the response messages are independent.

The operations in the portType will have input, output, and fault. Here is a snippet from SalesOrderEBS.wsdl for a synchronous request response operation:

```
<!-- operation support for read/query -->
<operation name="QuerySalesOrder">
   <documentation>
         <svcdoc:Operation>
```

```
         <svcdoc:Description>This operation is used to query a
SalesOrder EBO</svcdoc:Description>
         <svcdoc:MEP>SYNC_REQ_RESPONSE</svcdoc:MEP>
         <svcdoc:DisplayName>QuerySalesOrder</svcdoc:DisplayName>
         <svcdoc:LifecycleStatus>Active</svcdoc:LifecycleStatus>
         <svcdoc:Scope>Public</svcdoc:Scope>
         </svcdoc:Operation>
         </documentation>
   <input message="ebs:QuerySalesOrderReqMsg"/>
   <output message="ebs:QuerySalesOrderRespMsg"/>
   <fault name="fault" message="ebs:FaultMsg"/>
</operation>
```

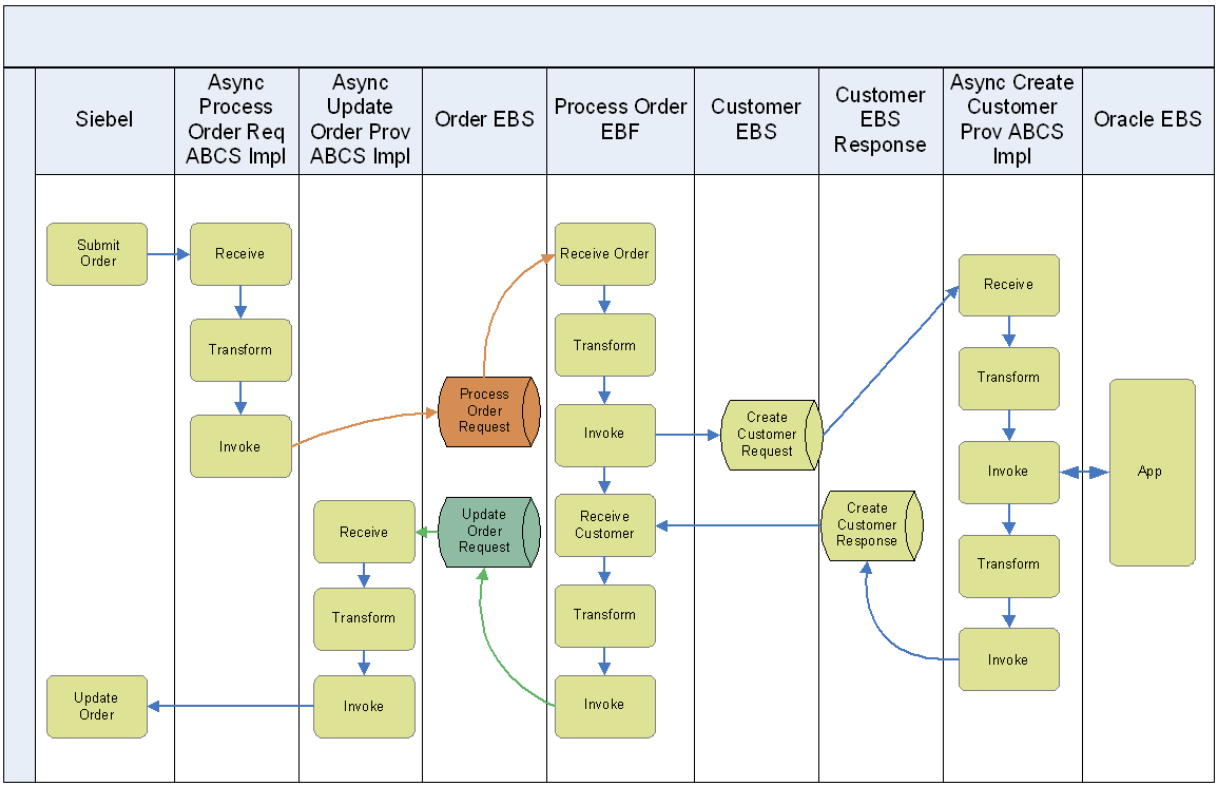
### 3.8.2. Asynchronous Fire-and-Forget Patterns in EBSs

A fire-and-forget EBS operation pattern is asynchronous in nature. This is an event that leads to a request message being posted to an endpoint or placed in a channel/queue/topic. This pattern is also characterized as a one-way call.

In a fire-and-forget pattern, the requesting service invokes a one-way operation in an EBS. The EBS invokes the providing service. No concept of a response exists even in the case of an error. In the Order EBS WSDL, the one-way operation meant for request is defined in the portType having all the operations, which are either request-response or request only.

In the case of a fire-and-forget pattern, once the request is made and presented to the provider, it cannot be re-presented. In case of any error in the provider service, it has to be handled locally. This includes either retrying or terminating. In case the error cannot be handled locally, then compensation needs to be initiated, if required.

In the diagram, the usage of the Order EBS depicts a fire-and-forget scenario. The Order EBS has two operations—Process Order Request and Update Order Request. Both of these operations are in a fire-and-forget pattern using the one-way calls in the EBS WSDLs.




## Order EBS showing fire-and-forget scenario

Here is a snippet from SalesOrderEBS.wsdl for a request-only operation.

```
<!-- operation support for creation -->
<operation name="CreateSalesOrder">
   <documentation>
         <svcdoc:Operation>
         <svcdoc:Description>This operation is used to create a
SalesOrder EBO.</svcdoc:Description>
         <svcdoc:MEP>REQUEST_ONLY</svcdoc:MEP>
         <svcdoc:DisplayName>CreateSalesOrder</svcdoc:DisplayName>
         <svcdoc:LifecycleStatus>Active</svcdoc:LifecycleStatus>
         <svcdoc:Scope>Public</svcdoc:Scope>
         </svcdoc:Operation>
   </documentation>
<input message="ebs:CreateSalesOrderReqMsg"/>
</operation>
```

The operations in the portType will have input only. This contract necessitates the invoker to make one-way calls. This could be a request-only operation. For Entity EBS, the WSDLs are part of the Foundation Pack Enterprise Service Library. For Process EBS, the WSDLs will be hand-coded based on template WSDLs provided.

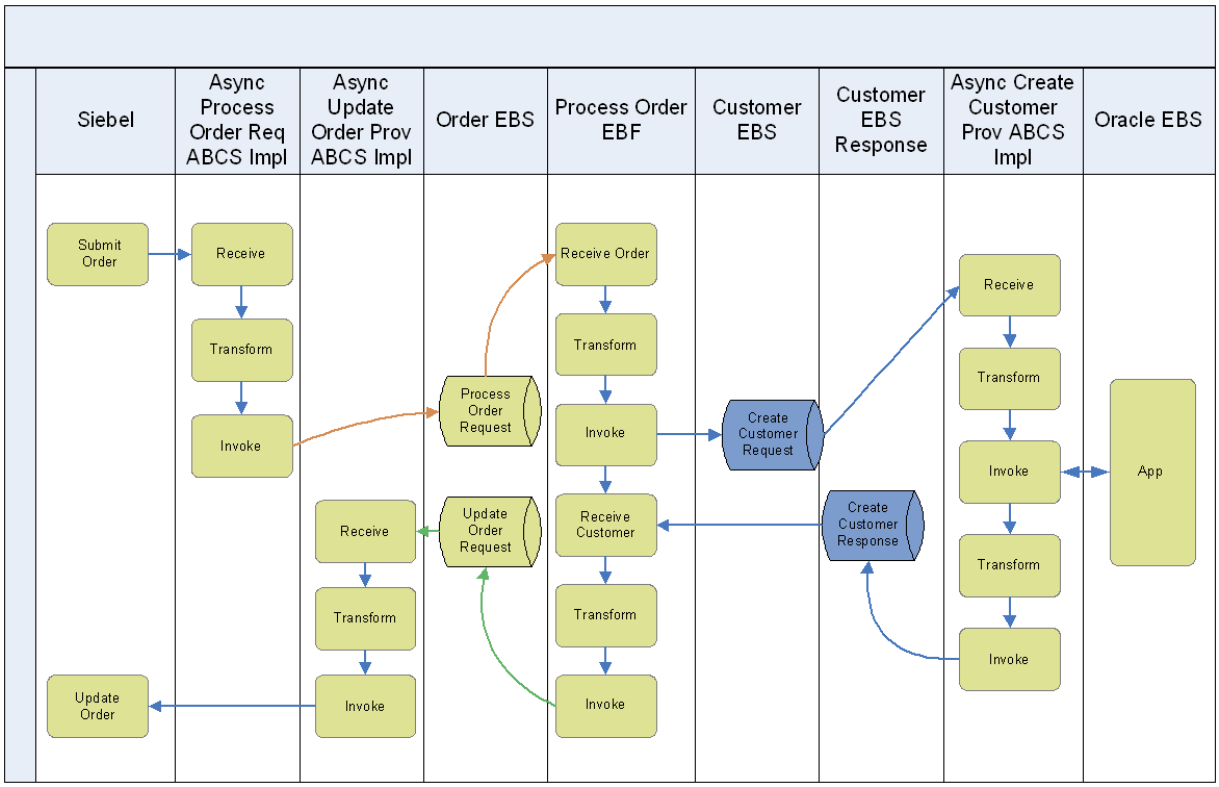
### 3.8.3. Asynchronous Request–Delayed Response Patterns in EBSs

A request-delayed response EBS operation pattern is asynchronous in nature. In this situation, the requester sends the request message and sets up a callback for a response. The requester will not be waiting for the response after sending the request message. A separate thread listens for a response message. When the response message arrives, the response thread invokes the appropriate callback and processes the response. The EBS would have a pair of operations—one for sending the request and another for receiving the response. Both of the operations will be independent and atomic. A correlation mechanism is used to establish the caller's context.

In a request-delayed response pattern, the requesting service invokes a one-way request operation in an EBS. The requesting service waits for the response. The EBS invokes the providing service. The providing service, after ensuring that the request is serviced, invokes the response EBS. The response EBS pushes the response to the waiting requesting service. If an error occurs in the providing service, the response is sent with fault information populated. In the Customer EBS WSDL, the one-way operation meant for request is defined in the portType having all the operations, which are either request-response or request-only. The one-way operation meant for response is defined in the portType having all the operations, which are Response.

In the diagram, the usage of the Create Customer EBS Request depicts a one-way request and Create Customer EBS Response depicts a one-way response. The Customer EBS is based on the Customer EBS portType and has the operation Create Customer Request. The Customer EBS Response is based on the Customer EBS Response portType and has the operation Create Customer Response. Both are modeled as one-way calls in the EBS WSDL.

This pattern allows for more flexibility in error handling. In case of error, a suitable response can be sent to the requester service and the request re-presented or resubmitted to the provider after error correction. In some situations, an error in the provider service can be handled locally too, similar to the fire-and-forget pattern. The methodology to be followed will be dictated by the business use case scenario.




## One-way request and one-way response

To increase the reliability of message delivery, a need exists to resort to persistence at strategic asynchronous points.

**For more information** about guaranteed message delivery designs**,** see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Configuring Oracle AIA Processes for Error Handling and Trace Logging," Implementing Error Handling and Recovery for the Asynchronous Message Exchange Pattern to Ensure Guaranteed Message Delivery.

Here is a snippet from SalesOrderEBS.wsdl for an async request response operation:

```
<!-- operation support for creation response-->
<operation name="CreateSalesOrderResponse">
        <documentation>
              <svcdoc:Operation>
              <svcdoc:Description>This callback operation will be
used to provide the Create Sales Order Response</svcdoc:Description>
              <svcdoc:MEP>ASYNC_REQ_RESPONSE</svcdoc:MEP>

   <svcdoc:DisplayName>CreateSalesOrderResponse</svcdoc:DisplayName>

   <svcdoc:LifecycleStatus>Active</svcdoc:LifecycleStatus>
              <svcdoc:Scope>Public</svcdoc:Scope>

   <svcdoc:InitiatorService>SalesOrderEBS</svcdoc:InitiatorService>

   <svcdoc:InitiatorInterface>CreateSalesOrderRequestEBM</svcdoc:Ini
tiatorInterface>
```

```
   <svcdoc:InitiatorOperation>CreateSalesOrderRequest</svcdoc:Initia
torOperation>
               </svcdoc:Operation>
         </documentation>
   <input message="ebs:CreateSalesOrderRespMsg"/>
</operation>
```

For Entity EBS, the WSDLs are part of the Foundation Pack Enterprise Service Library. For Process EBS, the WSDLs will be hand-coded based on template WSDLs provided.

# 4. Understanding Application Business Connector Services

This chapter provides an overview of Application Business Connector Services (ABCS) and discusses:

- ABCS architecture
- ABCS characteristics
- Architectural considerations
- Implementing ABCS
- Reviewing implementation technologies for ABCS
- ABCS transformations

**For more information** about ABCS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development."

## 4.1. ABCS

The role of the ABCS is to expose the business functions provided by the participating application in a representation that is agreeable to a service interface.

In canonical integration style, the common service interface will be the one exposed by Enterprise Business Services (EBSs). It can also serve as a glue to allow the participating application to invoke the EBSs.

In noncanonical integration styles, an EBS to expose the service interface might not be present. In these situations, the client application or its ABCS can directly invoke the provider ABCS. In other situations, the client application or its ABCS can invoke the EBS exposed using the service interface defined by the provider.

An ABCS can be requester-specific or provider-specific. A *requester ABCS* accepts the request from the client application through a client-specific Application Business Message (ABM) and returns the response to the client application through a client-specific ABM. The role of the requester ABCS is to act as a vehicle to enable the participating application to invoke the EBS either to access data or to perform a transactional task. The client side ABM will be the payload that is passed by the requester application to the requester ABCS.

The requester application that wants to leverage an action needs to define the requester-specific ABCS. The requester application that wants to implement this ABCS could be Siebel CRM, PeopleSoft Enterprise CRM, or Oracle eBusiness Suite CRM. The requester application-specific ABCS needs to take the requester application-specific ABM as input and provide the requester application-specific ABM as output.

The role of the *provider ABCS* is to expose the business capabilities that are available in the provider application according to the EBS specification. In certain noncanonical integration styles, the EBS interface will be defined by the provider. In other situations, the EBS artifact might be absent. The service-provider-side ABCS will accept the request either from the EBS through the Enterprise Business Message (EBM) or directly from either Requester Application or Requester ABCS (in case of certain noncanonical integration styles) and will send the response using the same format. ABCS are needed because every application has a different representation of objects, and any communication between applications necessitates the transformation of these objects to the canonical definition.

The ABCS is responsible for the coordination of the various steps that are provided by a number of services, including:

- Validation (if any)
- Transformations - message translation, content enrichment, and normalization
- Invocation of application functions
- Error handling and message generation
- Security

For each of the activities that can be performed with an Enterprise Business Object (EBO), an ABCS must be defined by the participating requester application and another ABCS must be defined by the service provider application.

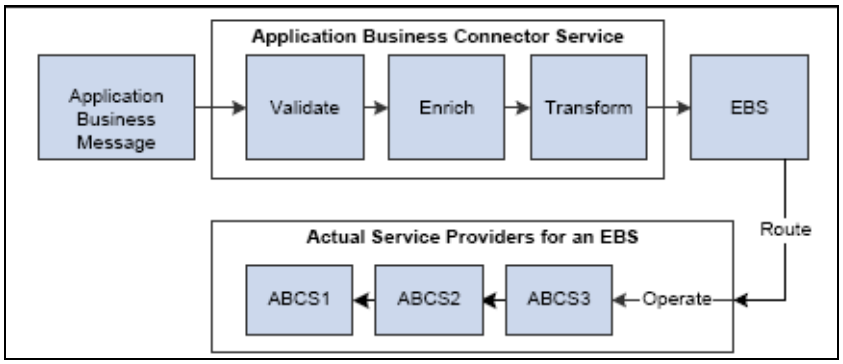
## 4.2. ABCS Architecture

For an application providing a business function to be a part of the Oracle Application Integration Architecture (AIA) ecosystem, it needs to be able to send messages that comply with either EBS or provider ABCS (in noncanonical integration styles) expectations. This enables the application to participate in cross-application business processes and prebuilt data integrations that exist in the Oracle AIA ecosystem.

Similarly, the application that receives messages from EBS must be able to understand the message types. Very few applications are built to readily interact with EBS. For applications that are not able to readily interact with EBS, ABCS act as conduits to expose the application-specific business functions in the Mediator.

The ABCS leverages a variant of the industry-standard integration pattern called the VETO - pattern. VETO is a common integration pattern that stands for Validate, Enrich, Transform, and Operate. The VETO pattern and its variations ensure that consistent, validated data is routed throughout the Mediator. The VETO pattern has many variations. A commonly known variation is the VETRO pattern. This includes the Route step.

The ABCS architecture adopts an extended variation of the VETRO pattern called the VETORO pattern.




## VETORO pattern

The VETORO pattern consists of the following steps:

- Validate

  This step can be used to ensure that the incoming message has the right content so that it can be transformed into what the target system is expecting. In certain cases, the validate step can be used to check whether the incoming message contains a well-formed XML document that is in conformance with the particular schema associated with the receiving application. The latter validation will often be performed as one of the first steps in the EBS and is needed to allow dissimilar versions of XML-producing and XML-consuming applications to coexist.

- Enrich

  This step involves adding additional data to a message to make it more meaningful and useful to a target service or application.

- Transform

  This step converts the message to a target format by translating an application-specific object representation into an EBO, or translating an EBO-specific representation into the application-specific object representation of the Operate step.

- Operate

  This step is the invocation of the target service or an interaction with the target application. In Oracle AIA, it could be either an invocation of the EBS or a specific ABCS, or an interaction with the participating application using the provider ABCS.

- Route

  The Route step deciphers the appropriate service provider that will be responsible for performing the service. In Oracle AIA, the Route step is implemented in the EBS using the content-based routing technologies that are available in Oracle Mediator.

In some cases, the validate, enrich, and transform steps can be accomplished in one ABCS implementation. Also, note that a Mediator routing service may use XSL-based validation as well as content-based routing directly in the service itself, rather than using a separate routing service.

The ABCS will not handle transport protocol abstraction itself. The ABCS will have inbound as well as outbound interactions using only the SOAP/Mediator protocol. A transport adapter will be used to integrate the participating application-specific native protocols with the ABCS-specific standard. This facilitates the reuse of the same ABCS for multiple transport adapters and vice versa.

## 4.3. ABCS Characteristics

The ABCS has the following characteristics:

- For each of the activities that can be done on an EBO (canonical pattern-based integration style), an ABCS must be provided by each of the participating requester and provider applications.
- In the case of a noncanonical pattern-based integration style, an ABCS must be provided by the provider application.
- The requester ABCS has participating application-specific ABMs as input as well as output.

  The service accepts requester application-specific ABMs as input and provides requester application-specific ABMs as output.
- The provider ABCS has EBMs (representing specific content of EBO needed for performing the operation) as input as well as output.

  The service accepts EBMs as input and returns EBMs as output.
- Although a single ABCS can be used to handle multiple activities, *Oracle highly recommends allowing only a single ABCS per action*. This approach greatly reduces the complexity of designing a generic ABCS.

If you do design a single ABCS to handle multiple activities, remember that the service needs to have the activity information accepted as a part of the input. This enables the ABCS to decipher the actual action to be performed and enables it to perform the appropriate transformations and invocations. In addition, allowing a single requester ABCS to handle multiple activities means that a single requester application-specific ABM will encompass all of the information pertaining to all of the activities.

## 4.4. Architectural Considerations

The architectural issues discussed in the following sections play a large role in determining your choice of implementation technologies as well as the design paradigms that you will use to construct ABCS.

### 4.4.1. Participating Application's Service Granularity

Perform some analysis to identify how the participating applications intend to expose their business functionality to the Mediator. If the applications expose their functionality using a web service interface, verify that the granularity of functionality matches exactly that of an application-agnostic or provider application-specific interface

If an exact match exists, then much of your new effort will be to transform the application-specific ABM into the enterprise EBM and vice versa. Mediator would certainly be a candidate as the implementation technology for building the ABCS for this design paradigm.

In situations in which granularity of functionality exposed by the application through the web service doesn't match that of the EBS, attempts should be made to have appropriate modifications or enhancements made in the application. In situations in which these modifications cannot be made to the application, the ABCS will not only be responsible for transformations, but will also need to aggregate and disaggregate services.

For example, perhaps a single business-level activity cannot be mapped to a single API or operation in the server application. The provider application might have very fine-grained operations and might also require that transaction management be handled by calling applications. In this case, the provider ABCS will probably have a chatty conversation with the provider application. The provider ABCS will also be responsible for state management.

This type of ABCS can be implemented only through BPEL technologies and not through Mediator services.

Although Oracle AIA allows for the existence of this type of ABCS, Oracle highly recommends that much of this application logic be encapsulated within native applications as opposed to having them handled in ABCS.

### 4.4.2. Support for EBMs

Because the EBS operates only on EBMs, you need to determine whether the applications that implement the services provide support for EBMs. In scenarios in which the application-provided services provide native support for EBMs, the effort for transforming the EBO into an EBM is minimal. In situations in which the applications that implement the services don't provide EBM support, you should determine whether their services can provide inherent support for EBMs.

If these applications cannot provide support for EBMs, transformation-related work needs to be done by the ABCS.

### 4.4.3. Application Interfaces

Perform a check to determine how the participating applications intend to allow the business logic to interact with the Mediator. Some applications may have inherent support for web service interfaces. This is the preferred scenario. The WSDL defines the interface that will be used to communicate directly with the application business logic. In this situation, the ABCS will use the web service interface to invoke the application business logic.

In the case of packaged applications such as Siebel, PeopleSoft, J.D. Edwards, and SAP, the much-preferred route is to use the respective packaged-application adapters. These adapters can be deployed as J2CA resource adapters. This is a better solution than using the conventional SOAP interface. In situations in which the participating applications don't expose their business logic as web services, interactions with these applications will need to occur by means of technology adapters such as database adapters, advanced queuing (AQ) adapters, and so forth.

Investigate whether the services exposed by the participating applications provide support for proprietary message formats, technologies, and standards. If the applications that implement the functionality don't have inherent support for standards and technologies such as XML, SOAP, and JMS, then the transformations need to happen in the ABCS.

For example, the application might be able to receive and send messages only through files, and EDI is the only format that it recognizes. In this case, the ABCS becomes responsible for integrating with the application using a file adapter, translating the EDI-based message into XML format, and exposing the message as a SOAP message.

**For more information**, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Establishing Resource Connectivity."

### 4.4.4. Support for Logging and Monitoring

The ABCS is responsible for facilitating logging and monitoring capabilities. The ABCS will invoke the convenience services for logging and auditing.

**For more information,** see *Oracle Application Integration Architecture Foundation Pack: Infrastructure Components and Utilities Guide*.

### 4.4.5. Support for Insulating the Service Provider

Situations will occur in which the granularity of the service that is provided by the service provider will not match that of the EBS. In this case, the ABCS must construct the message needed for interacting with the EBS from multiple fine-grained transactions.

For example, the BRM Billing application provides an API for retrieving only immediate child details for either a bill or bill detail. However, the interface that is defined by the EBS warrants that the service provider return complete details about a bill, including all of the details. In this case, it becomes the responsibility of the ABCS to provide protection to the application implementing the service by doing the workload buffering. The ABCS will make a series of sequencing calls to the application to retrieve all of the required information, consolidate the data into a single message, and send it to the calling application.

### 4.4.6. Support for Security

The security model that you choose will play a large role in determining which responsibilities are owned by the ABCS. Support for a point-to-point security model will require only that the ABCS authorize the service requests. Support for an end-to-end security model will necessitate the transmission of requester credentials to the service provider.

In the latter scenario, Web Services Security can be used if all participating systems provide support for it. Otherwise, transmitting security credentials either as application data or as a part of the SOAP header is an option. Depending upon the route taken, the ABCS must be coded accordingly so that it can be authenticated.

### 4.4.7. Validations

As the message travels from one participating application to another, validations need to occur at various stages throughout the journey to ensure that accurate and valid data is being sent to the target system. The specificity of the validations varies significantly depending on how far or close the message is from the target system.

Validations are based on the constraints that are imposed by the participating application providing the service. The ABMs generated from the EBMs need to comply with the constraints imposed by the participating applications. In addition, the target system may have its own built-in validation rules, which necessitates that the transformation step alter incoming data to prevent rejection of the message by the target system.

Because not all participating application-specific constraints can be enforced at the EBM level, these validations must be enforced by the specific ABCS. For example, the PeopleSoft CRM application might mandate that the product description be present for creating a product, while the product EBM might not enforce that constraint. In this situation, the PeopleSoft Create Product ABCS will be responsible for ensuring that the ABM generated from the EBM includes the product description. This validation will ensure that the ABM that is passed to the PeopleSoft CRM Create Product service is compliant with the constraints imposed by the service.

If the previously mentioned constraint is very specific to the PeopleSoft CRM application, then this validation needs to reside only in the PeopleSoft CRM Create Product ABCS. It is inappropriate to have this validation present in the early stages of message flow, for example, in the client-side ABCS.

In the case of asynchronous interaction styles (request-only), where the user experience will be hindered if validations are not performed at the time of capture, the requester application should use another integration point to perform the validation prior to submitting the message for processing. For example, in an Order Capture application, prior to submitting the order for fulfillment, the requester application will make a synchronous call to validate the order to ensure that the order contains the content required to enable successful provisioning. A Validate Order or Validate Billing Information EBS might exist that can be implemented by various service providers. If the response from the Validate Order service is favorable, the requester application will then make the request to process the order.

### 4.4.8. Support for Internationalization and Localization

The ABCS that pass the EBMs to the actual service providers are responsible for translating the document into a locale-specific ABM. Similarly, the ABCS is responsible for translating the locale-specific ABM into the locale-independent EBM.

The ABCS deciphers the locale based on the locale preferences of the user of the relevant participating application. This data provides information about how the locale-specific ABM needs to be constructed, as well as how the locale-specific ABM needs to be interpreted.

The ABCS should specify the locale in which the response needs to be provided by the real service provider. For example, the ABCS for Get Product Details EBS needs to specify the locale in which the product details should be provided by the Oracle eBusiness Suite application. If the requester wants the product details in Spanish, the Get Product Details EBS needs to instruct the real service provider that the product details need to be returned according to the Spanish locale.

### 4.4.9. Message Consolidation and Decomposition

Situations will occur in which a need will exist to combine responses to a request that originates from multiple sources. For example, in the case of convergent billing in the telecommunication solution, the ABCS for the getBillDetails EBS might have to retrieve details from multiple participating applications. This ABCS will also be responsible for consolidating them into a single response.

### 4.4.10. Support for Multiple Application Instances

Situations will occur in which multiple instances of a packaged application with the same business capabilities will exist in a customer's ecosystem. The routing rules defined in the EBS will be responsible for deciphering the right application instance to which the request needs to be routed. Regardless of the number of application instances for a packaged application, only one ABCS will exist for that packaged application to perform a specific business task.

## 4.5. Implementing ABCS

Each of the participating applications implementing a business activity or task will have its own ABCS.

An ABCS for a particular participating application should be responsible for implementing a single action. The action would be the same in the case of a synchronous request/response pattern. In the case of a delayed response pattern, the carrying out of the request will be implemented by one action and the carrying out of the response will be implemented by another action. Hence, two ABCS would exist.

The ABCS can be implemented in two ways:

- The first approach is to make complete use of components that are built using Oracle Fusion Middleware technologies to make up the ABCS.

  The service is implemented as a Mediator service or a BPEL process. This Mediator service or BPEL process performs the tasks listed in the following sections.

- The second approach is to build the transformation services, to a large extent, as part of the participating application.

  This approach can be taken when the participating application's technology stack has the infrastructure to perform the transformations. However, a lightweight ABCS will still need to perform the translations related to cross-reference details.

**For more information** about ABCS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development."

Regardless of how an ABCS is implemented, it is still a service, so an interface will exist that will be exposed as a WSDL. This is what the client applications will use to invoke the ABCS.

The transformations for each of the directions in both the client-side and server-side ABCS are implemented XSL scripts.

The ABCS itself should be independent of deployment. In situations in which multiple deployments of a single client or provider application exist, only one instance of an ABCS will still exist. For example, two deployments of BRM Billing applications could exist, one for customers who reside in North America and the second for customers living in the rest of the world. In this case, only one BRM application-specific ABCS would still exist for a particular action. Under no circumstances should two versions of the ABCS exist, one for each deployment. A single ABCS should be responsible for invoking the application business service that is available in the appropriate deployment.

In many situations, a single action cannot be mapped to a single API or operation in the provider application. The provider application might have very fine-grained operations and might also require that transaction management be handled by calling applications. In this case, the provider-side ABCS will probably have a chatty conversation with the server application. The provider-side ABCS will also be responsible for state management.

This type of ABCS can be implemented using only BPEL technologies and not through Mediator services.

Although Oracle AIA allows the existence of this type of ABCS, Oracle highly recommends that much of this application logic be encapsulated within native applications as opposed to having them handled in ABCS.

The service is responsible for populating the message header section of the EBM with values. The service deciphers some of the values by itself, whereas for other values it relies on the content being passed by participating applications through an ABM.

**For more information,** see EBM Architecture.

### 4.5.1. Requester-Side ABCS

The requester-side ABCS has the following core responsibilities:

- Enrichment or augmentation of the ABM.

  This may be required in situations in which the ABM received from the participating application does not contain all of the required content. The enrichment can be done by issuing additional calls to the participating application to get more information. For example, a CRM application might pass only the order identifier as part of the ABM. The interface for ProcessOrder EBS might warrant that the entire order object be passed as the payload. In this situation, the ABCS will be responsible for interacting with the participating application to get all of the required information to enrich the ABM.

- Transformation of requester application-specific ABMs into EBMs.
- Creation of an EBM that encompasses the previously mentioned EBM.
- Population of the message header with the appropriate values.

  **For more information,** see EBM Architecture.

- Invocation of any extension handler that the customer may have registered.

This extension handler could be used by the customer to perform any additional transformations. The extension handler is passed to the EBM and the transformed EBM is passed back as the response. This extension handler will enable customers to perform additional transformations on the EBO before the EBS is invoked.

**Note.** This functionality is not used by the Oracle AIA Process Integration Packs (PIPs) for Communications.

- Invocation of the EBS.

  The EBS is responsible for taking an EBM and providing the response using an EBM.

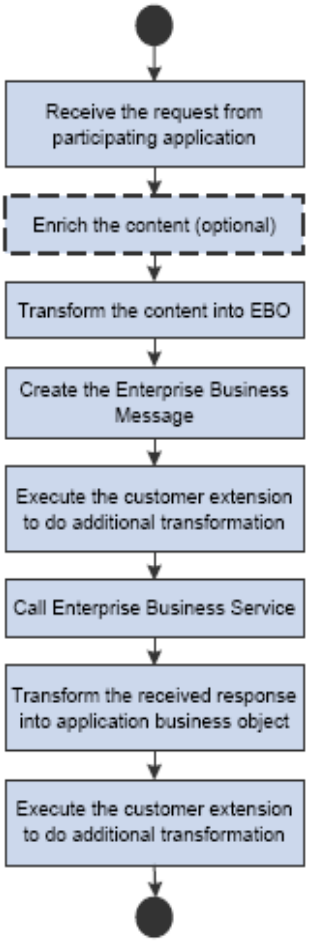
- Transformation of an EBM into an ABM.
- Invocation of any extension handler that the customer may have registered.

  This extension handler could be used to perform any additional transformations. The extension handler is passed to the ABM and the transformed ABM is passed back as the response. This extension handler enables customers to perform additional transformations on the ABM before the ABM is passed back to the calling application.

**Note.** This functionality is not used by the Oracle AIA Process Integration Packs for Communications.

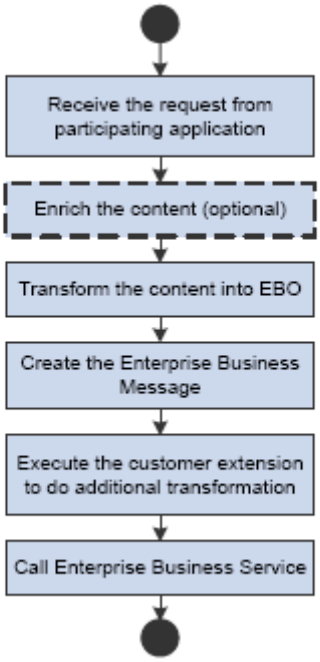
- Perform any necessary validations to ensure that the ABM that needs to be passed to the participating client application complies with the constraints enforced by the participating application.
- Return the response to the calling application.

The following activity diagram illustrates the high-level flow of activities in a requester-specific ABCS. The diagram makes an assumption that the EBS with which it is interacting employs a request/response interaction style. Note that the steps for running the customer extension to do additional transformations are optional.



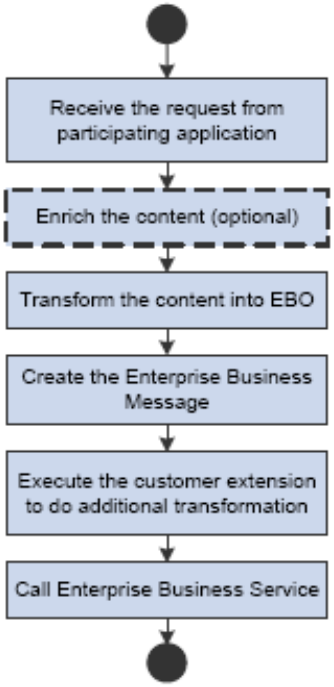

## Requester-specific ABCS interacting with Request/Response Interaction Style

The following activity diagram depicts the high-level flow of activities in a requester-specific ABCS. The diagram makes an assumption that the EBS with which it is interacting employs a fire-and-forget interaction style. Note that the steps for running the customer extension to do additional transformations are optional.




## Requester-specific ABCS interacting with request/response interaction style

The following activity diagram illustrates the high-level flow of activities in a requester-specific ABCS. The diagram makes an assumption that the EBS with which it is interacting employs a fire-and-forget interaction style. Note that the steps for running the customer extension to do additional transformations are optional.




## Requester-specific ABCS interacting with fire-and-forget interaction style

### 4.5.2. Provider-Side ABCS

The provider-side Application Business Connector Service has the following core responsibilities:

- Transformation of an EBM into a provider application-specific ABM.
- Invocation of any extension handler that a customer may have registered.

  This extension handler could be used by a customer to perform any additional transformations. The extension handler is passed to the ABM and the transformed ABM is passed back as the response. This extension handler enables customers to perform additional transformations on the ABM before the ABM is passed to the provider application business service.

  **Note.** This functionality is not used by the Oracle AIA Process Integration Packs for Communications.

- Performance of any necessary validations to ensure that the ABM that needs to be passed to the participating provider application complies with the constraints enforced by the participating application.
- Invocation of the provider application business service.

  One or multiple calls may be made to the provider application business service.

  **For more information,** see Architectural Considerations.

- Transformation of an ABM into an EBM. The response message sent by the provider application needs to be returned to the caller application. In this case, it would be EBS. Because EBS expects only EBM as the output, the transformation needs to be done to transform the ABM into an EBM.
- Creation of an EBM that encompasses the previously mentioned EBO.
- Population of the message header section with the appropriate values.

  **For more information,** see EBM Architecture.

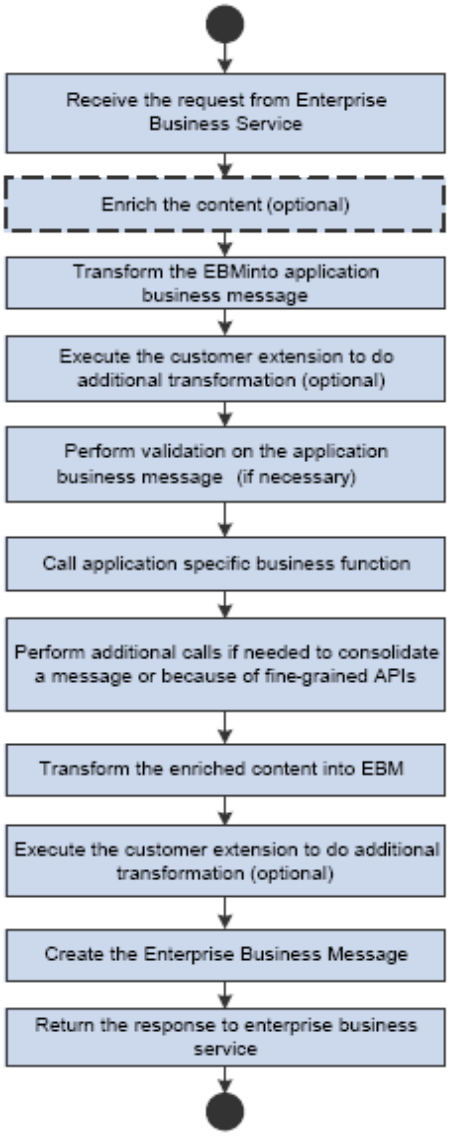
- Invocation of any extension handler that a customer may have registered.

  This extension handler could be used by a customer to perform any additional transformations. The extension handler is passed to the EBM and the transformed EBM is passed back as the response. This extension handler enables customers to perform additional transformations on the EBM before the document is passed to the EBS.

  **Note.** This functionality is not used by the Oracle AIA Process Integration Packs for Communications.

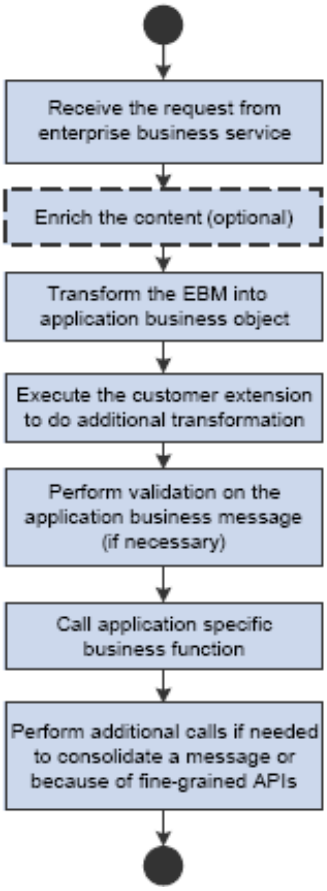
- Return of the document to the EBS.

The following activity diagram depicts the high-level flow of activities in a provider-specific ABCS. The diagram makes an assumption that the EBS with which it is interacting employs a request/response interaction style




## Provider-specific ABCS interacting with request/response interaction style

The following activity diagram depicts the high-level flow of activities in a provider-specific ABCS. The diagram makes an assumption that the EBS with which it is interacting employs a fire-and-forget interaction style.




Provider-specific ABCS interacting with fire-and-forget interaction style

# 4.6. Reviewing Implementation Technologies for ABCS

Oracle AIA provides two blueprints for implementing an ABCS: Oracle Mediator and BPEL.

## 4.6.1. Oracle Mediator

The Oracle Mediator blueprint can be applied in situations in which you do not need the ABCS to do additional enrichment and validation of the content. In this model, the ABCS are implemented as Mediator services.

This enables customers to easily add transport adapters either before the client-side ABCS or after the provider-side ABCS to use a different transport. This will involve only configuration changes.

**For more information** about using Oracle Mediator, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing and Developing Enterprise Business Services" and "Tuning AIA Process Integration Packs."

### 4.6.2. BPEL

BPEL is used when the ABCS must augment content, validate content, or both. In most situations, the ABCS will need to have a conversation with one or more participating applications to enrich the content. It may also have to handle state management.

In this scenario, BPEL is the preferred technology. BPEL enables you to perform the tasks listed previously and also enables you to extend the connector. This architecture doesn't preclude you from implementing the ABCS using procedural object-oriented languages such as Java or C++.

BPEL will also be used when a Mediator service can't be used to implement the ABCS either due to the constraints in Mediator technology or due to the complexities.

**For more information** about using BPEL, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development."

## 4.7. Extending or Customizing ABCS Processing

Oracle AIA facilitates the use of different transports without having to modify the delivered artifacts. For example, you could add a third-party billing application-specific implementation without modifying the Query Customer Party EBS. Similarly, you can change the transport mechanism by which the services provided by the participating application are invoked without having to modify the ABCS.

Each of the client-side and provider-side ABCS implementing request/response pattern provides four extensibility points. In the case of the requester ABCS, two extensibility points are provided prior to the invocation to the EBS and two after the receipt of response message from EBS. In case of provider ABCS, two extensibility points are provided prior to the invocation of application-specific service and two after the receipt of response from the application service. In case of fire-and-forget patterns, the ABCS will have only two extensibility points. These extensibility points can be used to inject additional behavior. The services for injecting additional behavior can have capabilities such as custom validation or custom transformations. Transformations are used primarily for any additional elements that have been introduced at the implementation site. You can use this feature to introduce any additional processing that needs to be done without having to customize the delivered ABCS. The extension service is passed either the EBM or the ABM, depending on the situation. The content is passed as context to the extension service and the extension service returns the content after performing the alterations.

**For more information** about ABCS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*"Completing ABCS Development."

**Note.** This functionality may not be supported by all process integration packs.

## 4.8. Processing Multiple Instances

Because an EBM can carry multiple instances, the provider ABCS should be able to iterate through each of the instances and process them. Not all EBMs have support for carrying multiple instances. But EBMs that are created specifically for supporting bulk processing can be defined to support multiple instances.

The ABCS can decide whether its application has the ability to process the instances in bulk form. If the application has the ability, then the ABCS transforms the content into ABM format for all of the instances and hands them over to the participating application by invoking the service. Upon receiving the response, the ABCS transforms the content of each instance back into EBM format, consolidates them in an EBM, and returns the message to the EBS.

If the provider application doesn't have the capability to process all of them in bulk, then the provider ABCS must invoke the services of the provider application for each of the instances, consolidate them, and return that message to the EBS.

## 4.9. Participating Applications Invoking ABCS

When a requester application encounters a business event, it might send a request to get details from another application by invoking an ABCS. At the time of invocation, the requester application will pass the ABM to the ABCS. The requester application can either pass everything that an ABCS will ever need or it can pass just the bare minimum, which in turn could be used by ABCS as the driver to fetch relevant details from the client-side participating application. In the latter approach, the ABCS might need to engage in conversations with the participating application to get all of the details relevant to compose an EBM. Although the architecture can support both approaches, Oracle highly recommends that the participating applications resort to the first approach to minimize overhead.

Because the ABCS will be responsible for enclosing an EBO in an EBM, much of the information pertaining to population of the EBM header with attributes also needs to be passed by the participating application. For this reason, in addition to passing the business content, the participating application will also be passing data that is related to the source environment. This information is needed to associate an EBM with the originator.

**For more information,** see EBM Architecture.

The participating application is also responsible for specifying the locale that was used to construct the ABM. This locale can be used to interpret the locale-specific content. This enables the translation of locale-specific content into locale-independent content and vice versa.

# 4.10. ABCS Transformations

The transformations found in ABCS are participating application-specific components. The main responsibility of ABCS is to perform transformations and invoke the services provided by the participating application. The transformations result in replacement of application-specific fields with some generic fields and vice versa. The transformation also involves the replacement of any static and non-static application-specific identifiers with a common identifier. The nonstatic identifier-related transformations are done with the help of the Mediator cross-reference facility. The types and number of transformations done in a single ABCS depend upon the design patterns employed and the type of participating application with which the service is interacting. A single ABCS can interact with either a client-side or provider-side participating application.

**For more information** about ABCS, see *Oracle Application Integration Architecture Foundation Pack: Development Guide, "*Working with Message Transformations."

## 4.10.1. Transformation: Implementation Approach

Two ways are available by which transformation can be implemented. In the case of ABCS that are specific to a particular action, the entire set of transformations is present in an XSL file. The ABCS invokes the XSL file to perform the transformations.

The architecture also provides a facility for you to provide transformations for additional elements that were introduced as a part of their implementation. The customer-specific transformations don't modify the artifacts delivered by Oracle AIA and hence will survive upgrades.

The ABCS transformations handle:

- Cross-referencing
- Error-handling
- Validation rules (such as format validation)

Transformations can be simple or complex. The following transformation map patterns are handled by the Oracle AIA service:

- Data field mapping
- Static data cross-referencing
- Dynamic data cross-referencing
- Structural transformation

### 4.10.2. Static Data Cross-Referencing

Different applications and common component objects frequently use different values for enumerated types. For example, the values for the Country common component object may be the full country name, such as *United States of America*, while it could be a two-letter code, such as *US* in an ERP application. Static cross-referencing maps the values between an application and the common component object model. It would map the *US* and *United States of America* values presented in this example. Similar to dynamic cross-referencing, static cross-referencing uses a scheme to store and resolve the cross-references. However, in this case, the mapping is static and the mapping table is populated only at design time. The domain value mapping facility that is available in the Mediator is used to facilitate static data cross-referencing.

### 4.10.3. Dynamic Data Cross-Referencing

Typically, each application generates its own set of identifiers or keys for the data objects that it stores. A data object replicated in multiple applications ultimately has different IDs or keys in different applications. Identifying that given data objects are the same becomes an issue of identifying the mapping between these keys. The EBS provides a dynamic cross-referencing scheme that assigns a common or global key to data objects and maintains mappings between application-specific keys and the common key in a dynamic cross-referencing table.

### 4.10.4. Structural Transformation

Structural transformation provides the transformation between two different, but related, structures. Some examples include:

- Joining a sibling to a single child
- Converting rows to columns
- Converting columns to rows

# 5. Understanding Interaction Patterns

Oracle Application Integration Architecture (AIA) solutions are delivered as Mediator and BPEL services to create specific integration scenarios between named participating applications. The services interact with each other in various ways, giving rise to diverse interactive styles and patterns for exchanging messages between services. This chapter lists various patterns, highlights the features, and presents guidelines for choosing patterns based on their suitability to an integration scenario.

This chapter provides an overview of patterns for exchanging messages and discusses:

- Request/response
- Fire-and-forget
  - Message routing
  - Message splitting and routing
- Data enrichment
- Data aggregation
- Asynchronous request–delayed response pattern
- Publish-and-subscribe

**For more information**, see *Oracle Application Integration Architecture Foundation Pack: Development Guide*.

## 5.1. Patterns for Exchanging Messages

Business requirements drive the need for different patterns to exchange messages between participating applications in an integration scenario.

In any application integration, the core functionalities of the best applications are leveraged to accomplish tasks by linking them to business processes. The functionalities are exposed as APIs. Events in applications trigger information interchange as a straight-through process consisting of multiple tasks spanning multiple applications. This can be real-time or batch mode. From the perspective of events triggering the information interchange, the interactions can be invocations that are synchronous, asynchronous, or a combination of the two.

*Synchronous operations* wait for a response before continuing. This forces the operations to occur in a serial order. It is often said that an operation blocks or waits for a response. Synchronous operations open a communication channel between the parties, make the request, and leave the channel open until the response occurs. This method is effective unless large numbers of channels are being left open for long periods of time. In that case, asynchronous operations may be more appropriate. The synchronous pattern may not be necessary or appropriate if the end user does not need an immediate response.

*Asynchronous operations* do not wait for a response before continuing. This allows operations to occur in parallel. The operation does not block or wait for the response. Asynchronous operations will open a communication channel between the parties, make the request, and close the channel before the response occurs. Message correlation is used to relate the inbound message to the outbound message. This method is effective when large numbers of transactions could take long periods of time to process. If the operations are short or need to run in serial, synchronous operations may be more appropriate. The asynchronous pattern is effective if the end user does not need immediate feedback.

The basic patterns for exchanging messages along with variations of the basic patterns are detailed in the following sections.

## 5.2. Request/Response

In this pattern, a requester sends a request message to a replier system, which receives and processes the request, ultimately returning a message in response. This enables two applications to have a two-way conversation with one another over a channel.

### 5.2.1. Synchronous Response

Consider the following business problem: An application making a request to get information from an external system has to wait until the response is received.

- Use case

  A CRM application needs to get account details from a billing application. When the service rep clicks a button in the CRM application to get the account details for a customer, the CRM application sends an Application Business Message (ABM) to the Siebel Get Account Details Application Business Connector Service (ABCS), which is responsible for invoking the Get Account Details Enterprise Business Service (EBS). The CRM application waits for the ABM to be returned by the Siebel Get Account Details ABCS before it can render the information on the screen.

- Synchronous requester/response

  The requester sends a request and waits for a response message. The service provider receives the request message and responds with either a response or a fault message. After sending the request message, the requester waits until the service provider responds with a message or a time-out. Both the request and the response messages are independent.

## 5.3. Fire-and-Forget

Consider the following business problem: A customer integrating two applications doesn't want the sender application to be affected due to nonavailability of the receiving application.

- Use case

The customer has CRM On Demand and CRM On Premises applications. The customer wants the opportunities that are created in the CRM On Demand application to be created as quotes in the CRM On Premises application. The conversion of opportunity to quote happens in real time to near real time. But at the same time, nonavailability of the CRM On Premises application should have no impact on the functioning of the CRM On Demand application. Also, no work that is done on CRM On Demand during nonavailability of CRM On Premises can be lost.

- Asynchronous transaction using queue/topic

  Oracle AIA uses queues for asynchronous and reliable delivery of messages. CRM On Demand, upon occurrence of a business event, can either push the message directly into a queue or send a SOAP message over HTTP to a JMS message producer (a Mediator adapter) that will be responsible for entering the message in the queue. CRM On Demand can consider the message as sent as soon as the message is dropped into the queue. With this mechanism, the CRM On Demand application can continue sending new messages regardless of whether the CRM On Premises is available. A JMS consumer (another Mediator/BPEL service with JMS adapter) is responsible for dequeuing the messages and invoking the CRM On Demand ABCS. Having the CRM On Demand ABCS, EBS, the CRM On Premises ABCS, and the web services as part of the transaction initiated by the JMS message producer will ensure that the message gets removed from the queue only after the successful completion of the task in the CRM On Premises application.

### 5.3.1. Message Routing

Consider the following business problem: A customer with multiple applications providing the same business function wants to employ certain criteria to identify the application that can provide service for a specific request.

- Use case

  The customer has two billing systems—one from vendor A and one from vendor B. The customer uses vendor A's billing system for servicing EMEA customers and vendor B's billing system for servicing North American customers. The customer has a CRM system that needs to get bill details for their customers. They don't want the CRM system to be responsible for routing the request to the appropriate billing system.

- Message routing/mediation

  Oracle AIA architecture enables the requester to be completely decoupled from the provider. For this use case, Get Bill Details will be the EBS operation. Each of the billing applications provides ABCS for Get Bill Details. The ABCS of the CRM application is integrated only with the Get Bill Details EBS. The Get Bill Details EBS is implemented as a Mediator routing service. The customer can define the routing rules at this level to identify the appropriate ABCS that needs to be invoked. For EMEA customers, vendor A's ABCS will be invoked; and for North America customers, vendor B's ABCS will be invoked. Each of the vendor's ABCS will be responsible for interacting with their applications to get the bill details and hand them over to EBS in Enterprise Business Message (EBM) format

**For more information,** see Understanding Enterprise Business Services.

Consider the following business problem: A customer with multiple instances of the same application providing the same business function wants to employ certain criteria to identify the application instance that can provide service for a specific request.

- Use case

  The customer has two instances of the Oracle BRM application in their ecosystem—BRM Instance A and B. The customer uses BRM instance A for servicing EMEA customers and instance B for servicing North American customers. The customer has a CRM system that needs to get bill details for their customers. They don't want the CRM system to be responsible for routing the request to the appropriate BRM instance.

- Message routing/mediation

  Oracle AIA architecture enables the requester to be completely decoupled from the provider. For this use case, Get Bill Details will be the EBS operation. Only one ABCS for the BRM application will exist. This ABCS will be responsible for routing the request to the right instance. The ABCS of the CRM application will be integrated only with the Get Bill Details EBS. The Get Bill Details EBS is implemented as a Mediator routing service. The customer can define the routing rules at this level to identify the appropriate ABCS that needs to be invoked. The customer needs to define a transformation service for populating the target system information in the EBM header. The BRM ABCS will use the information present in the EBM header to route the request to the right BRM instance.

**For more information,** see Understanding Enterprise Business Services.

## 5.3.2. Message Splitting and Routing

Consider the following business problem: A business document with multiple line entries needs each one of the line items to be handled differently

- Use case

  The customer has two billing systems—one from vendor A and another from vendor B. The customer uses vendor A's billing system for broadband customers and vendor B's billing system for wireless customers. The customer has a CRM system that sends a Process Order Request EBM that can contain requests for both the services. In this situation, the broadband-related portion of the order needs to be sent to vendor A's billing system and the wireless product-related portion of the order needs to be sent to vendor B's billing system.

- Message splitting and routing

  The CRM system (CRM ABCS) will invoke the Process Order EBS operation. The implementation will be a BPEL process that is responsible for splitting the order business document into multiple business documents each having data for only one order line. After splitting the documents, each of these is handed over to another EBS, such as Activate Service. This EBS will use the routing rules to decipher the billing system that needs to be used.

## 5.4. Data Enrichment

Consider the following business problem: The requester application does not send all of the required data that is necessary for invoking the EBS.

- Use case

  The CRM application passes only the order identifier as part of the ABM to the requester ABCS for invoking the Order Fulfillment process. However, the ProcessOrder EBS expects the entire order object to be passed as the payload.

- Data enrichment of the ABM

  This may be required in situations in which the ABM that is received from the participating application does not contain all of the required content. The enrichment can be done by issuing additional calls to the participating application to get more information. In this situation, the ABCS is responsible for interacting with the participating application using web services (or JCA-based adapters, if available) to get all of the required information to enrich the ABM. The participating application is responsible for exposing the needed business capabilities as web services.

## 5.5. Data Aggregation

Consider the following business problem: The provider application does not return all of the required data expected by EBS.

The provider application does not have a service that matches the granularity of the EBS operation.

- Use case

  The EBS operation Get Bill Details provides complete bill information for a particular customer. It passes the customer identifier and the specific month as the input parameters to the service provider. It expects the complete bill details from the provider. The billing application has a web service that can provide only bill summary. The application doesn't have a single service that can provide the complete details. However, it does have services to get details for every part of the bill.

- Data aggregation of the ABM

  Situations will occur in which you need to make multiple interactions with the provider application to get all of the content; and then combine them to produce a single response that can be returned to the EBS. For this use case, the ABCS for the billing application will interact three times with the provider application using three web services—for getting bill header, bill summary, and for bill details. After retrieving all of the content, the ABCS will be responsible for combining the three ABMs and producing a single EBM.

Consider the following business problem: Information needed for providing the response is spread across multiple applications.

- Use case

The EBS operation Get Bill Details provides complete bill information (information for all of the services in one bill) for a particular customer. It passes the customer identifier and the specific month as the input parameters to the service provider. It expects the complete bill details from the provider. However, the customer has one billing application that has information about wireless services only, another that has information about broadband only, and a third system that has information about land line-related services.

- Data aggregation of the ABM

  The ABCS for the Get Bill Details EBS needs to retrieve details from multiple billing applications. It is responsible for interacting with each of these applications using the services provided by them. This ABCS is also responsible for consolidating them into a single response. After retrieving all of the content, the ABCS will combine the three ABMs and produce a single EBM.

## 5.6. Asynchronous Request – Delayed Response Pattern

A request – delayed response pattern is asynchronous in nature. In this situation, the requester sends the request message and sets up a callback for a response. The requester will not be waiting for the response after sending the request message. A separate thread listens for the response message. When the response message arrives, the response thread invokes the appropriate callback, and processes the response. The EBS would have a pair of operations—one for sending the request and another for receiving the response. Both the operations will be independent and atomic. A correlation mechanism is used to establish the caller's context.

In a request – delayed response pattern, the requesting service invokes a one-way request operation in an EBS. The requesting service waits for the response. The EBS invokes the providing service. The providing service, after ensuring that the request is serviced invokes the response EBS. The response EBS pushes the response to the requested service waiting for the asynchronous response. If an error occurs in the providing service, the response is sent with fault information populated. In the Customer EBS WSDL, the one-way operation meant for request is defined in the portType having all the operations, which are either Request – Response or Request only. The one-way operation meant for response is defined in the portType having all the operations, which are for Response.

## 5.7. Publish-and-Subscribe

Multiple integration scenarios exist in which participating applications publish events and messages that are subscribed to by multiple participating applications. This pattern is transactional in the sense that changes are made to the entities in the participating applications. These scenarios require an asynchronous and durable implementation model.

**For more information** about the publish-and-subscribe interaction pattern and implementing the publish-and-subscribe programming model, see *Oracle Application Integration Architecture Foundation Pack: Development Guide,* "Establishing Resource Connectivity."

# 6. Understanding Extensibility

This chapter provides an overview of extensibility and discusses:

- Schema extensions
- Transformations extensions
- Transport/flow extensions
- Process extensions
- Routing extensions

## 6.1. Extensibility

One of the capabilities of the architecture is to allow for various artifacts of prebuilt integrations to be extended by customers. It also ensures that these extensions are protected during the upgrades, although for some extensions, configurations may have to be done after the upgrade to point to the artifacts. The Oracle Application Integration Architecture (AIA) artifacts have been designed and constructed from the ground up to have native support for extensibility.

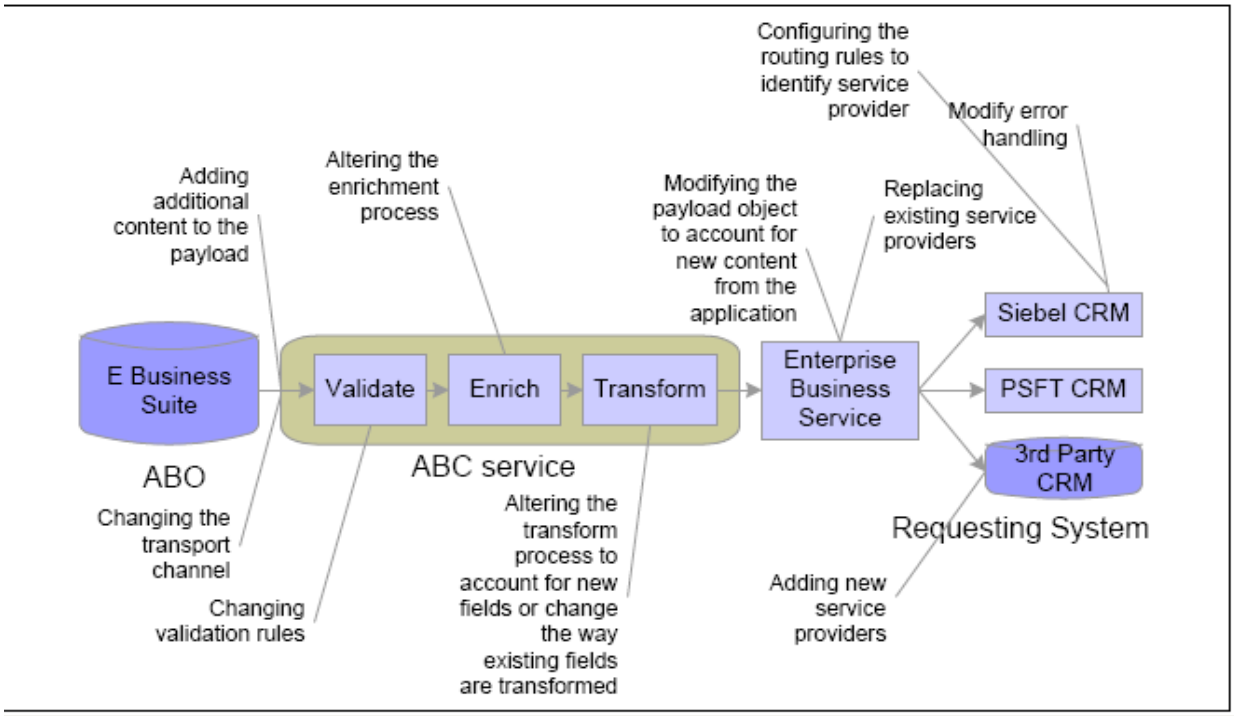



Illustration of customer extensibility points

## 6.2. Schema Extensions

Oracle AIA facilitates the extension of delivered Enterprise Business Objects (EBOs) to accommodate the introduction of industry-specific and customer-specific needs.

The EBO is extensible to meet the specific needs of an implementation. Extensions are clearly separated from the original structure of the EBO delivered by Oracle AIA so that future delivered versions will not override any extensions that have been defined.

All of the EBO extensions have been designed to reside in the extension-specific namespaces. Extensions must use their own namespace name for two reasons:

- Each family of extensions must be distinguishable from the core components of the XML format and other extensions.

  Without providing such identification, naming conflicts might occur between different extensions or between extensions and future additions to the core specification. Hence, the customer and vertical-specific extensions will each reside in their own namespace.

- A straightforward path should exist from identifying an extension to learning more about it.

Two approaches are available for implementing extensibility:

- Customer-specific extensions
- Industry-specific extensions

### 6.2.1. Customer Extensions

Every component that is available in an EBO is enabled to accommodate customer-specific extensions. Every component has an additional element added at the very end that is designed to accommodate customer-specific attributes that may be applicable to that component. In an implementation, a customer may decide to complete the definition for the data types for which data elements are of interest.

**For more information** about extension-enabling an EBO and about how to add customer-specific attributes to an existing EBO, see the *Oracle Application Integration Architecture Foundation Pack: Development Guide*.

### 6.2.2. Industry-Specific Extensions

The architecture allows for incorporating industry-specific attributes as overlays. An industry-specific object can be created by assembling a set of business components that are available at the core with a set of industry-specific components. This approach is available only for delivering industry-specific EBOs by Oracle. Customers can add industry-specific content using the approach listed in Customer Extensions.

### 6.2.3. Schema in the Use Case

When implementing the Get Account Balance use case, you might want business-specific information such as usage details about the customer to be retrieved from the billing system. Assume that this information is available in BRM and that this information is made available by BRM web service. You want the usage details to be rendered on a Siebel screen along with the other information that has been brought from BRM. For this additional content to be sent to the calling application through integration, you need to extend the Enterprise Business Message (EBM), in this use case, the Query Customer Party Response EBM.

## 6.3. Transformation Extensions

The transformation scripts that are delivered as part of prebuilt integrations are made extension-ready. This allows for customer-defined transformations to be introduced in a nonintrusive manner and ensures that customer-specific transformation-related extensions are durable.

Every component in the EBM, including the EBM header, contains an extensibility point that can be configured to add the necessary transformations. A transformation script exclusively dedicated for housing customer extensions is delivered. You can add transformation code to the templates that are available in the customer-specific transformation script to specify transformation rules for the newly introduced content.

In this release, the XSLT extensibility programming model focuses on providing hooks for customers to add maps only to the new elements they have added to EBM. The programming model at this time does not provide mechanisms for overriding the existing maps for certain elements. Similarly, adding maps to existing elements that have no maps provided by the PIPs are not addressed.

### 6.3.1. Extensions in the Use Case

You will see what needs to be done to send the extensions to the Siebel screen in the Get Account Balance use case. The Oracle BRM Application Business Connector Service (ABCS) for Query Customer Party is responsible for transforming the Application Business Message (ABM) into an EBM. Assume that the BRM web service has already retrieved the details and made them available in the ABM. Now you need to make sure that the content present in ABM is made available in EBM.

You leverage the predefined extensibility point and include the code for transformations in the extensions script. The customer-specific content Usage Details is available in the Query Customer Party Response EBM.

The Enterprise Business Service (EBS) operation Query Customer Party returns the Customer Party Response EBM back to the Siebel Query Customer Party ABCS. The transformation script that is present in this ABCS is responsible for transforming the EBM into an ABM, which is then sent to the Siebel application. You will again use the transformation script that is exclusively meant for customer extensions and include the code pertaining to this transformation. Now the Siebel ABM will get the ABM from the ABCS and can display the Usage Details on the screen.

## 6.4. Transport/Flow Extensions

Oracle AIA enables you to change the transport channel by which the messages travel between the participating application and the connector services in a nonintrusive manner. For example, the prebuilt integration that is delivered with the system might use SOAP/HTTP to transport the message between Siebel CRM and the connector service. But at implementation time, you might decide to ship the data using a file. This change can be made in a configurable manner without making any customizations to the delivered artifacts.

## 6.5. Process Extensions

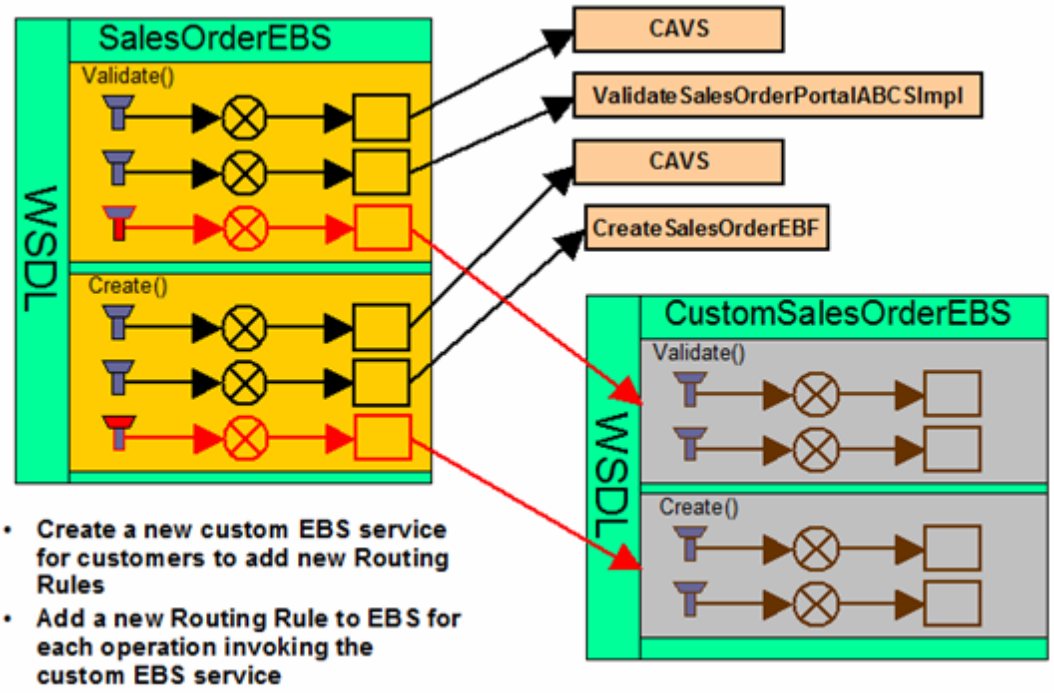
The architecture provides recommendations on how the ABCS as well as the orchestration processes (Composite Business Processes [CBPs] and Enterprise Business Flows [EBFs]) can be designed to allow the customers to introduce the extensions to augment the functionality.

Each of the BPEL processes can have its own set of extensibility points based on the functional needs. You can implement the interface that will be defined for each of the extensibility points to either augment or override the behavior. ABCS are recommended to have a minimum of four extensibility points, in case of request/response pattern, to enable customers to inject additional behavior. In case of the fire-and-forget pattern, the services are expected to have a minimum of two extensibility points. The orchestration processes might decide not to have any extensibility points.

Refer to the appropriate Process Integration Packs (PIPs) to check whether they have support for process extensions.

## 6.6. Routing Extensions

Oracle AIA enables you to add custom routing rules using the custom extension points provided in the custom EBS. This enables you to route the message to any other homegrown applications or services plugged into the Oracle AIA to extend your service provisioning for any service request.




## Routing extensions

**For more information** about whether a PIP supports routing extensions, see the respective PIP implementation guide.

# 7. Understanding Versioning

This chapter provides details about how Oracle Application Integration Architecture (AIA) handles versions for Enterprise Business Objects (EBOs), services, and participating applications. Major and minor versions, backward compatibility, and naming conventions are discussed.

This chapter discusses:

- Schema versioning
- Service versioning
- Participating applications versioning

## 7.1. Schema Versioning

Oracle AIA allows for the natural evolution of EBOs. Each of the EBOs continues to evolve over multiple generations as you add content, remove existing content, or change the semantics or characteristics of existing content. The primary reasons for changes in EBOs are:

- Product enhancements
- Bug fixes
- Adoption of new technologies and language enhancements

### 7.1.1. Major and Minor Versions

Each of the EBOs in the library has its own release life cycle, so each object has a version number that is used to differentiate versions. EBO version numbers are not aligned with participating applications release numbers, and new releases of participating applications do not necessarily result in introduction of new versions of the objects.

The version number is composed of two parts—major and minor version number.

A new major version number is introduced when the object undergoes the following types of changes, which could break the backward compatibility of the object:

- Changing the meaning or semantics of existing components
- Adding required components
- Removing required components
- Restricting the content model of a component, such as changing a choice to a sequence
- Changing the type of an element or attribute

A new minor version number is introduced when the object undergoes the following types of changes:

- Adding optional components, such as elements, attributes, or both
- Adding optional content, such as extending an enumeration
- Adding, changing, or removing default initializations, changing the order of elements in a choice, and so forth

*Backward compatibility* means that newer clients must be able to interpret data from older services. *Forward compatibility* means that older clients must be able to interpret data from newer services.

The major and minor terminology used in this section refers to characteristics of the change, not to any quantitative measure. One small change could be sufficient to qualify for a major version change if that change breaks backward compatibility. Similarly, enormous changes may result in a minor version change if backward compatibility is not broken.

The Enterprise Object Library will always be cumulative. Within a single major version, the schema module for that EBO might have undergone several iterations of changes that warranted incrementing of minor version number. For every iteration of backward-compatible changes, the minor version number will be incremented. Hence, the schema file that is present in the folder related to the major version will always contain the latest and greatest. The Enterprise Object Library will contain the latest version of the schema module for every one of the major versions introduced for each of the EBOs. In the future, deprecation rules will be laid out pertaining to how the earliest major versions can be brought to end of life.

A release that updates the major version number of an EBO contains changes that might sometimes make it incompatible with the prior major release. This means that consumer applications that depend on an earlier major release might need to be modified to work with the new release. On the other hand, a release that updates the minor version number of an EBO is a backward-compatible change. This means that an application written against version 1.0 will work when targeted against versions 1.1 and 1.2, but may fail if moved to version 2.0 of the EBO.

Because Oracle AIA leverages a service-oriented architecture that involves the common adoption of the request/response interaction style by the web services, backward and forward compatibility surface at the same time. When a provider application is upgraded, the provider application needs to be backward compatible to understand requests from older requesters. At the same time, requesters need to be forward compatible to recognize the responses of the provider application. Compatibility in both directions, at least among minor versions, ensures the utmost degree of independence of providers and requesters.

Because the architecture will not mandate (in most situations) that the requester and provider of the message be upgraded at the same time, additional transformations must be provided to transform XML messages written against previous major versions into a format to work against the newer versions and vice versa. In some cases, these transformations may not be technically feasible or may not make functional sense. In these situations, the applications receiving the messages will cause a fatal error.

The version of the schema is identified with the help of the *schema declaration version attribute* that is available in the XML schema and with the help of a required custom attribute on the XML instance document. An XML instance specifies exactly which namespace and minimal version it is structured to validate against. The XML instance does not use the schemaLocation attribute. The XML instance documents provide the schemaVersion attribute on the top-level Enterprise Business Message (EBM) element to indicate the version of the schema used to generate the document. For example:

```
<GetAccount ... schemaVersion="1.1"> ... </GetAccount>
```

### 7.1.2. Namespaces

Each of the EBOs has its own namespace. This is advantageous because it minimizes the duplication of names and it provides the flexibility for letting each of the business objects have its own release cycle. The namespace uses the following format:

http://xmlns.oracle.com/EnterpriseObjects/Core/EBO/[object name]/v[version number]

The namespace name will be the same across multiple minor and major versions and will be changed only when the schemas undergo major architectural changes. Introducing backward-incompatible changes alone don't warrant namespace changes.

Here is a sample namespace:

http://xmlns.oracle.com/EnterpriseObjects/Core/EBOParty/v1

The innermost layer of the objects library is a set of namespaces that contain those constructs that are commonly used throughout the Enterprise Objects Library. Some of these namespaces include core component types, business datatypes, and core datatypes. In earlier releases, only one namespace will hold all of the common components and reference components.

These namespaces are imported by the next layer of namespaces, which denote functional areas. Each second-layer namespace has a set of declarations that are specific to a business process or functional area. For example, the documents used for placing a purchase order all reside in the PurchaseOrder namespace.

In addition, customer-specific namespaces exist that are designed to house customer-specific extensions.

When the innermost layer namespaces are versioned, the next layer namespaces are also versioned if they have to leverage the new common constructs. The functional layer-specific namespaces can be versioned independently because the functional-layer namespaces have no necessary dependency on them. The innermost layer does not import the functional layer-specific namespaces. This scheme implies that the entire snapshot as a whole has no actual version—it is merely a group of interdependent schema modules that are versioned independently.

## 7.2. Service Versioning

Oracle AIA allows for the natural evolution of Enterprise Business Services (EBSs). A change, either in the interface definition or the implementation that could affect the contract that the consumer relies upon will lead to the creation of a new version of the service.

With this concept, Oracle AIA facilitates the co-location of multiple implementations of a single EBS with each version being totally identifiable. Multiple versions of the same service allows for consumers to use a particular version of the service that caters to their needs. Introduction of a new version of the service doesn't force the consumers of a specific version of the service to switch to the latest version immediately.

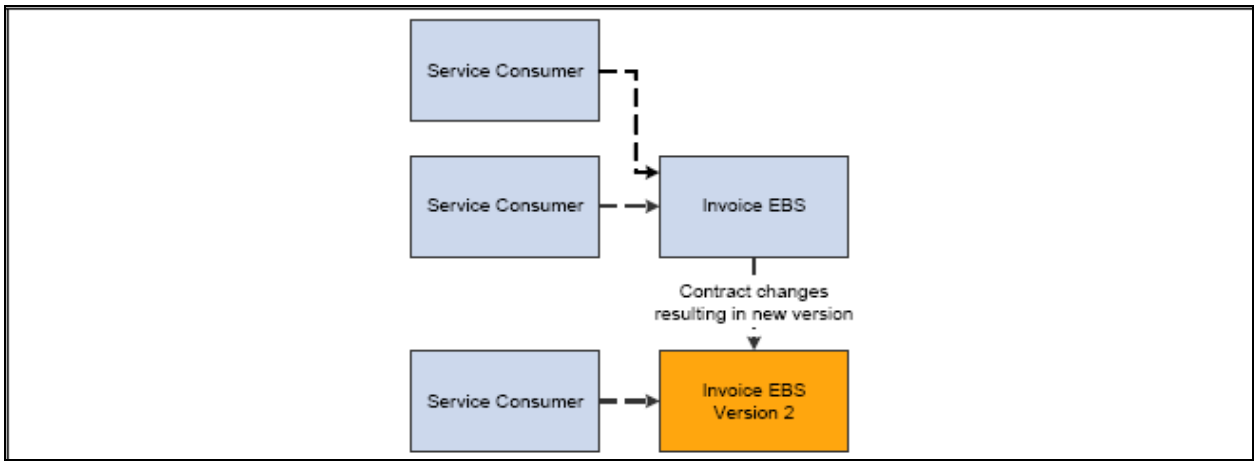



Illustration of versioning

### 7.2.1. Naming Conventions

This section discusses the naming conventions only with respect to versioning.

Similar to EBOs, each of the EBSs in the library will have its own release life cycle and each of the services will have a version number. The first version of the service will not have a number affixed to it. The default value will be 1.0. Subsequent versions of the service will have numbers affixed to the name of the service to differentiate different versions of the object. This construct allows for multiple versions of the service to be co-located in the same ecosystem and enables you to recognize the multiple versions of the same service. The EBS version numbers will not be in alignment with those of participating applications release numbers. New major releases of participating applications do not necessarily result in the introduction of new versions of the services.

## 7.3. Participating Applications Versioning

The applications that participate in the integration will also continue to evolve regardless of whether they are playing a requester or provider role. The new versions of the applications can introduce enhancements to their native functionality, to the underlying connecting technologies, or to the web services standards. The Application Business Connector Services that are specific to participating applications will not have any impact if the new version of the participating application does not introduce any changes pertaining to the connectivity/transport protocol, web service definitions, or payloads.

# 8. Understanding Batch Processing

In situations that warrant the high-performance movement and transformation of very large volumes of data between heterogeneous systems in batch, real time, and synchronous and asynchronous modes, Oracle Application Integration Architecture (AIA) leverages Oracle's extract, transform, and load (ETL) tool called Oracle Data Integrator. By implementing an ETL architecture, based on the relevant RDBMS engines and SQL, Oracle AIA can perform data transformations on the target server at a set-based level, giving a much higher performance.

Oracle AIA leverages batch processing technology for the following types of use cases:

- To perform an initial synchronization of reference data across disparate applications
- To load an Operational Data Store to provide fresh, integrated information
- To load production databases from data entered over the Internet (by sales forces, agencies, suppliers, customers, and third parties) that strictly respects security constraints
- To leverage the use of Cross Reference and Domain Value Map, for those cases in which the data transferred is used for the running of services from an Integration Scenario

**For more information** about batch processing, see the *Oracle Application Integration Architecture Foundation Pack: Development Guide,* “Using Oracle Data Integrator for Bulk Processing.”

# 9. Understanding Security

Oracle Application Integration Architecture (AIA) provides support for all security-related functions including:

- Identification
- Authentication (verification of identity)
- Authorization (access controls)
- Privacy (encryption)
- Integrity (message signing)
- Non-repudiation
- Logging

The service-oriented architecture (SOA)-based integration approach allows for clear separation between the interface and the actual business logic. This provides the security architect with a number of choices in deploying security for SOA and web services.

For example, a SOAP web service interface such as *CreateSalesOrder* can be hosted as a proxy instead of the real endpoint that hosts the business logic implementation. The gateway proxies communication to and from the web service and performs security functions on behalf of the service endpoint. The actual endpoint is virtualized. Even though the client thinks it is talking directly to the service provider, it communicates through the proxy.

Oracle AIA leverages web service administration tools such as Oracle Web Services Manager (OWSM) in a nonintrusive manner to ensure the validity as well as safety of the XML messages exchanged between services. This methodology ensures that no enforcement of web services security is in silo mode. This approach enables integration architects and developers to focus on integration logic, and the security architects and administrators to focus on security and management. Having security policies enforced via a centralized tool enables the administrators to ensure that the corporate rules are applied as well as to apply the policy changes centrally instead of applying them in each of the web services. With tools such as OWSM, an administrator creates security and management policies using a browser-based tool as and when needed during deployment. Security should be avoided unless it is absolutely needed as it degrades performance. A typical web service security policy could:

- Decrypt the incoming XML message
- Extract the user's credentials
- Perform an authentication for this user
- Perform an authorization check for this user and this web service
- Write a log record of the preceding information
- Pass the message to the intended web service, if all steps are successful
- Return an error and write an exception record, if all of the steps are not successful

To apply the security policy, OWSM intercepts every incoming request to a web service and applies any one of the policy items listed previously. As the policy is executed, OWSM collects statistics about its operations and sends them to a monitoring server. The monitor displays errors, service availability data, and so on. As a result, each web service in an enterprise network can automatically gain security and management control, without the service developer coding extra logic.

## 9.1. Point-to-Point or End-to-End Security

Because a typical interaction in the Oracle AIA framework will be part of multiple discrete interactions involving a service requester, client-specific Application Business Connector Service (ABCS), Enterprise Business Service (EBS), server-specific ABCS and the service provider, choosing a security model plays a critical role.

Oracle AIA provides support for point-to-point and end-to-end security models. The architecture enables you to choose one over the other at the implementation time for each of the transactions.

To choose a specific security model and implementation technique, the following issues should be discussed:

- Can an entire transaction be considered as secured as long as the individual discrete transactions are secured?
- Can the *trusted model* expressed previously be agreed to in principle, or must it be enforced using certificates provided by a certificate authority for the discrete interactions?
- Can the communication-level security methods such as SSL encryption be used to secure the individual discrete transactions within a trusted model?

  Adoption of the industry-standard WS-Security security model is possible, provided that all participating applications in the transactions provide inherent support.

## 9.2. Transport-Level Security

Existing technologies such as SSL can be used to secure the transport channel. SSL enables two applications to securely connect over a network and authenticate each other. It also enables you to encrypt the data exchanged between the applications. In Oracle's Web Services Security model, this transport security mechanism can be used to provide point-to-point security, data integrity, and data confidentiality.

## 9.3. Message-Level Security

Oracle AIA places strong emphasis on message-level security. For a web service, XML encryption provides security for applications that require a secure exchange of data. While SSL was considered the standard way to secure data exchanges, it has limitations. For example, assume that a document visits several web services before hitting its eventual endpoint. By using XML encryption, the document can be encrypted while at rest or in transport. Encrypting only portions of a document instead of the whole document is also possible.

## 9.4. Securing ABCS

The ABCS passes the participating application-specific security credentials, such as the user ID, under which the transaction in the participating application should run.

The identity information of service requester, such as user name, is propagated from end-to-end. This identity information can then be used for authorization by the application providing the service.

## 9.5. Implementation Techniques for Enterprise Service Bus Security

Consider whether the client identity and password need to be transmitted from one end to another via the Enterprise Business Message (EBM) header.

Use custom headers in the WS-Header section to transmit the security information to the server-side ABCS. The ABCS will be responsible for interpreting the custom header information and invoking the participating application in the appropriate manner.

**For more information** about security, see *Oracle Application Integration Architecture Foundation Pack: Development Guide,* “Working with Security.”

# Index