Oracle® Fusion Middleware

Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack

11*g* Release 1 (11.1.1.7)

E17363-07

February 2013

This guide introduces Oracle Application Integration Architecture (AIA) and provides an overview of components of AIA like Enterprise Business Objects, Enterprise Business Messages, Enterprise Business Services, and Application Business Connector Services and discusses integration concepts like interaction patterns, extensibility, versioning, batch processing and security.

ORACLE

Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack, 11*g* Release 1 (11.1.1.7)

E17363-07

Primary Author: Oracle Corporation

Contents

2 Understanding EBOs and EBMs

3 Understanding Enterprise Business Services

4 Understanding Application Business Connector Services

5 Understanding Interaction Patterns

Preface

Welcome to the *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack 11g Release 1 (11.1.1.7).*

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

Oracle AIA Guides

In addition to this *Oracle Fusion Middleware Concepts and Technologies Guide for Oracle Application Integration Architecture Foundation Pack,* we provide the following Oracle AIA guides for this 11.1.1.7 release:

- *Oracle Fusion Middleware Installation and Upgrade Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack*
- *Oracle Fusion Middleware Migration Guide for Oracle Application Integration Architecture*
- *Oracle Fusion Middleware Product-to-Guide Index for Oracle Application Integration Architecture Foundation Pack*

Related Documents

The following guides are relevant to Oracle AIA development activities and are provided as a part of the overall Oracle Fusion Middleware 11.1.1.7 documentation library:

- *Oracle Fusion Middleware Installation Guide for Oracle SOA Suite and Oracle Business Process Management Suite*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*
- *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository*

Additional Resources

The following resources are also available:

Resource	Location
Release Notes	Oracle Technology Network: http://www.oracle.com/technology/
Documentation updates	Oracle Technology Network: http://www.oracle.com/technology/

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

What's New in This Guide for Release 11.1.1.7.0

The following table lists the sections that have been added or changed for Release 11.1.1.7.0. The last column denotes for which release of 11.1.1.7.0 the feature was added.

For a list of known issues (release notes), see the "Known Issues for for Oracle SOA Products and Oracle AIA Foundation Pack" at http://www.oracle.com/technetwork/middleware/docs/soa-aiafp-knownissuesindex-364630.html.

Sections	Changes Made	11.1.1.7.0
Chapter Understanding the Oracle AIA Reference Architecture		
Section 1.5.3, "What is a Task?"	The term Task was described as a Business Task. This is now changed to Task.	X

1

Understanding the Oracle AIA Reference Architecture

This chapter introduces Oracle Application Integration Architecture (AIA) and integrations and provides an overview of Reference Process Models, Shared Service Inventory, and AIA Service Artifacts.

This chapter includes the following sections:

1.1 Introduction to Oracle Application Integration Architecture

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 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 use the applications of their choice to create Composite Business Processes (CBPs) following these guiding principles, which 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 AIA Features

AIA Foundation Packs provide the methodology, framework, and content that is critical for customers to figure out their integration problems.

Figure 1–1 illustrates the main AIA features and the underlying technology.

Figure 1–1 AIA Features

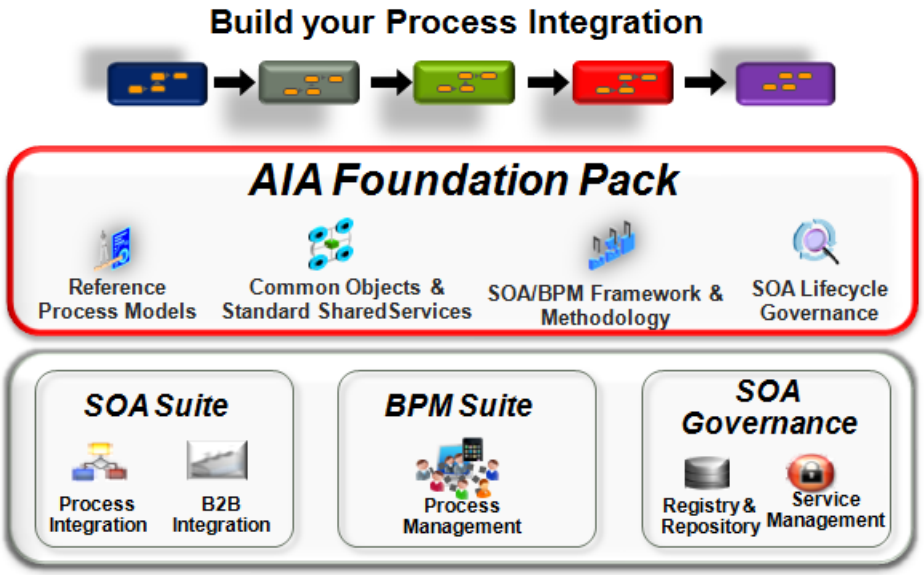


Table 1–1 lists the main features of AIA and how these features are delivered.

Table 1–1 AIA Features and Related Deliverables

Features	Deliverables
A robust architectural framework for engineering service-oriented business processes.	▪ Reference Process Models ▪ Reference Architecture ▪ Foundation Pack - infrastructure components ▪ Pre-Built Integrations
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 Pre-Built Integrations.	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 Workbench and Deployment Plan Generator

Table 1–1 (Cont.) AIA Features and Related Deliverables

Features	Deliverables
Governance of design-time and run-time AIA artifacts.	Oracle Enterprise Repository

1.3 Understanding Integrations

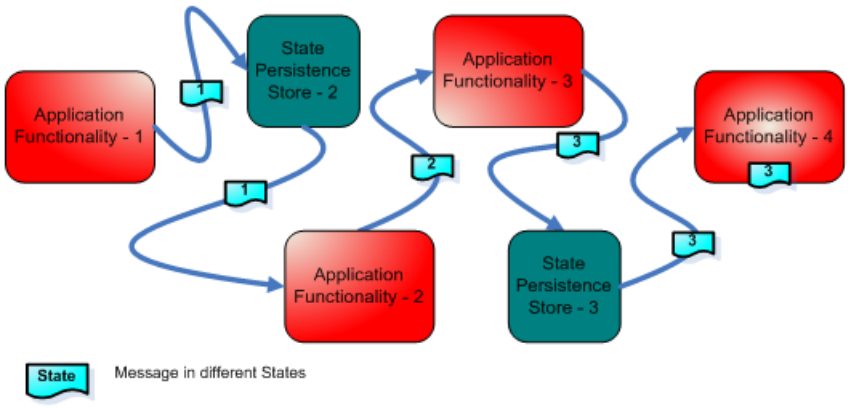
This section includes the following topics:

- Section 1.3.1, "The Integration Flow Concept"
- Section 1.3.2, "Integration Styles"
- Section 1.3.3, "Data-Centric Integrations"
- Section 1.3.4, "Integration Through Native Interfaces"
- Section 1.3.5, "Integration Through Web Services"
- Section 1.3.6, "Reference Data Query"
- Section 1.3.7, "Process-Centric Integration"
- Section 1.3.8, "What Integration Pattern Should be Used: Direct Transformation Of The Data Objects or a Canonical Data Model And EBOs?"

1.3.1 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 as shown in Figure 1–2. At each milestone, the message is stored in a different state.

Figure 1–2 An Integration Flow



An integration flow represents the run-time path of a message. It is not a design-time artifact.

There is no "one size, fits all" approach to integration. The requirements and objectives of the integration drive the selection of integration styles and supporting design patterns. AIA supports a variety of integration styles and patterns to enable the flight of a message in an integration flow. The AIA artifacts required for the collaboration between applications or functions are dependent on the integration style adopted for an integration flow.

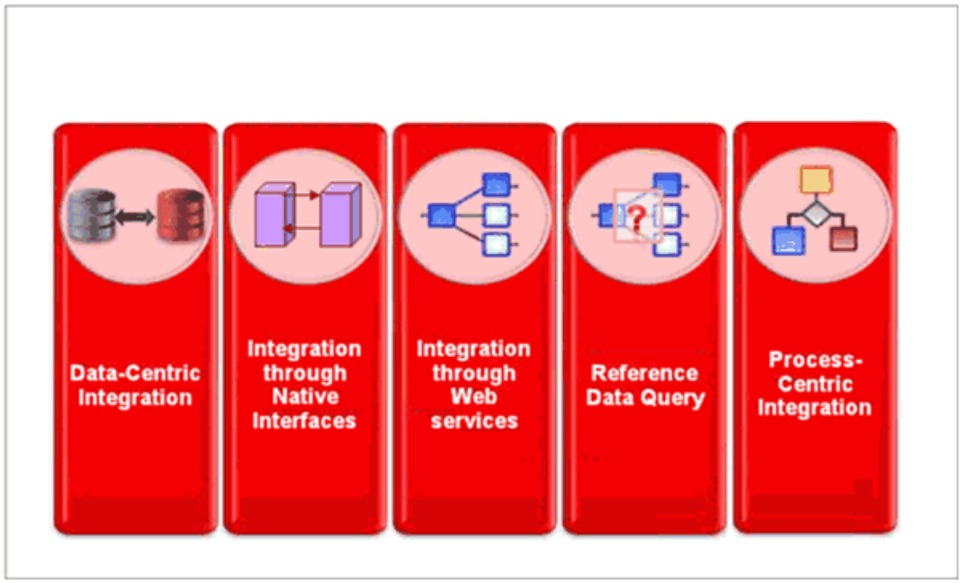
1.3.2 Integration Styles

The following sections provide details about each of these integration styles:

- Data-centric integration
- Integration through native interfaces
- Integration through Web services
- Reference data query
- Process-centric integration

Figure 1–3 illustrates the flow of each integration style.

Figure 1–3 Integration Styles



1.3.3 Data-Centric Integrations

Data-centric integrations transform data from its source format into its target format. They provide some level of logic in the transformation process to make it useful and coherent to the target environment.

Typical requirements driving data-centric integrations include bulk data replication, where no individual message based processing is required, or where multiple data enrichment activities are not required to populate the target system.

Example 1 Initial Data Synchronization

Initial data synchronization occurs between a running application that contains meaningful but nonvolatile data and a new application that is being installed. This typically makes on-going integration efforts using one or more of the integration styles much simpler to achieve.

Example 2 Bulk Upload of Recurring Transactions

An example of a bulk upload of recurring transactions is sales orders that must be tallied for commission payments to the appropriate sales people. The commission

payments are made either quarterly or monthly, and while sales activities are going on every day, there is no need to have this information available in real time.

1.3.3.1 What Oracle Provides

Technology Foundation

Oracle provides two data-centric integration tools: Oracle Golden Gate and Oracle Data Integrator.

Oracle Golden Gate uses the change data capture logs of the databases and can work across a heterogeneous database landscape, while Oracle Data Integrator works directly with the database tables and utilizes an extract, load, and transform approach to its handling of the data.

Pre-Built Integrations

Oracle delivers several pre-built data-centric integrations. Among them are the sales order and commission payments use case described previously and an automated scheduler to publish general ledger reports from one application to another. They provide capabilities for general ledger reports, such as incremental and cumulative revenue reporting, automated report regeneration, and the ability to customize reports.

1.3.4 Integration Through Native Interfaces

Every application executes on top of its own technology stack. Based on the application architecture, there may be one or more supported ways to integrate with the application. For example, Oracle E-Business Suite exposes public Java interfaces, PL/SQL APIs, business events, and batch interfaces to allow a wide range of integration options for customers and partners.

Typical requirements driving data-centric integrations include a simple exchange of business-oriented messages or consumption of more sophisticated application capabilities exposed through these interface mechanisms. One example of the use of native interfaces is providing enterprise data through multiple channels, such as web and voice, where the application itself has no such capability, and the speed of delivery is critical due to a high degree of human interaction.

1.3.4.1 What Oracle Provides

Technology Foundation

From a technology foundation perspective, each application exposes one or more mechanisms for integration based on its native technology stack. Some applications provide extended tooling to assist with the exposure, configuration, customization, and extension.

Pre-Built Integrations

Through the Oracle Validated Integration program, several pre-built integrations created by Oracle partners exist, which take advantage of the applications' native interfaces. For example, one of Oracle's partners integrates with Siebel CRM to provide multi-channel access to contact center information. This improves customer satisfaction with their interactions to the call center by allowing customers to choose their method of interaction (phone, chat, email, fax, letter, and so forth.) and get a consistent experience regardless of the medium.

1.3.5 Integration Through Web Services

XML-based Web services are a technology platform-independent way of exposing application interfaces. Using XML as the common language, business transactions are sent from one application to another in near real time. Web services are exposed using industry standards such as WSDL and SOAP (or in a REST-ful manner), shielding the implementation and connectivity details of the provider application from the consumer. The primary requirement which leads to the use of Web services is similar to that of native integrations, which involves the exchange of business messages. The key difference is the need to integrate applications across a wide variety of potentially incompatible technology platforms, including integration with third-parties (for example, business-to-business scenarios or the use of Software-as-a-Service). This eliminates the use of the native interfaces as an option.

1.3.5.1 What Oracle Provides

Technology Foundation

Oracle Fusion Middleware, including SOA Suite and Adapters (bridging from an application's native interfaces to the world of Web services), allows for the consumption, composition, and integration of applications using Web services. The inclusion of Fusion Middleware to support Web service-based integration enables an expanded set of Quality of Service characteristics, like resiliency, scalability, guaranteed delivery, and security.

Pre-Built Integrations

Through the Oracle Validated Integration program, several pre-built integrations created by Oracles partners exist, which take advantage of the applications' Web service-based interfaces. One partner has leveraged PeopleSoft's Web services interfaces and Oracle Fusion Middleware to exchange standard HR-XML schemas. As a result, ordering a background check on an applicant is a matter of just a few clicks, and after a report is complete, the detailed search results can be viewed immediately from within the PeopleSoft Enterprise interface.

1.3.6 Reference Data Query

Reference data query addresses supplemental information important to completing and or facilitating business transactions, but is not in and of itself part of the primary integration activity. Web services have further fueled the availability of reference data from third-party providers, and many companies offer reference data services on a subscription basis.

Typical use cases include geocode or tax code lookups, evaluating availability to promise (ATP) for inventory/manufactured goods, or something as common as retrieving an account balance.

1.3.6.1 What Oracle Provides

Technology Foundation

Oracle Fusion Middleware, including SOA Suite and Adapters (bridging from an application's native interfaces to the world of Web services), allows for the consumption, composition, and integration of applications using Web services. The inclusion of Fusion Middleware to support Web service-based integration enables an expanded set of Quality of Service characteristics, like resiliency, scalability, guaranteed delivery, and security.

Pre-Built Integrations

Through the Oracle Validated Integration program several pre-built integrations created by Oracle partners exist which take advantage of the applications' Web service-based interfaces for reference data query. One partner provides a pre-built integration with Oracle E-Business Suite for tax jurisdiction lookup and tax calculation. The partner's solution includes a SOAP service, deployed at the customer site, which then invokes their Software-as-a-Service system for geocode lookup and tax calculation. The interface leverages the tax partner subscription method of Oracle E-Business Tax for third-party tax vendor integration.

1.3.7 Process-Centric Integration

While many applications expose Web service interfaces, there is a growing need for a more coordinated exchange of messages between applications. This orchestrated exchange of messages is typically intended to support one or more business processes. As a part of the orchestration of messages, additional activities may take place, such as message transformation, enrichment, or validation.

An example of the process-centric integration style is the orchestration of transactional data, from the point of capture through transformation and movement to back-office systems. You can also use this style to drive the movement of transactional data through a process, based on a business event.

1.3.7.1 What Oracle Provides

Technology Foundation

From a technology foundation perspective, Oracle provides support for the definition and orchestration of business processes through the SOA Suite and BPM Suite offerings. In addition to the expanded set of QoS capabilities, Oracle provides tools to visualize and define the orchestration itself.

Pre-Built Integrations

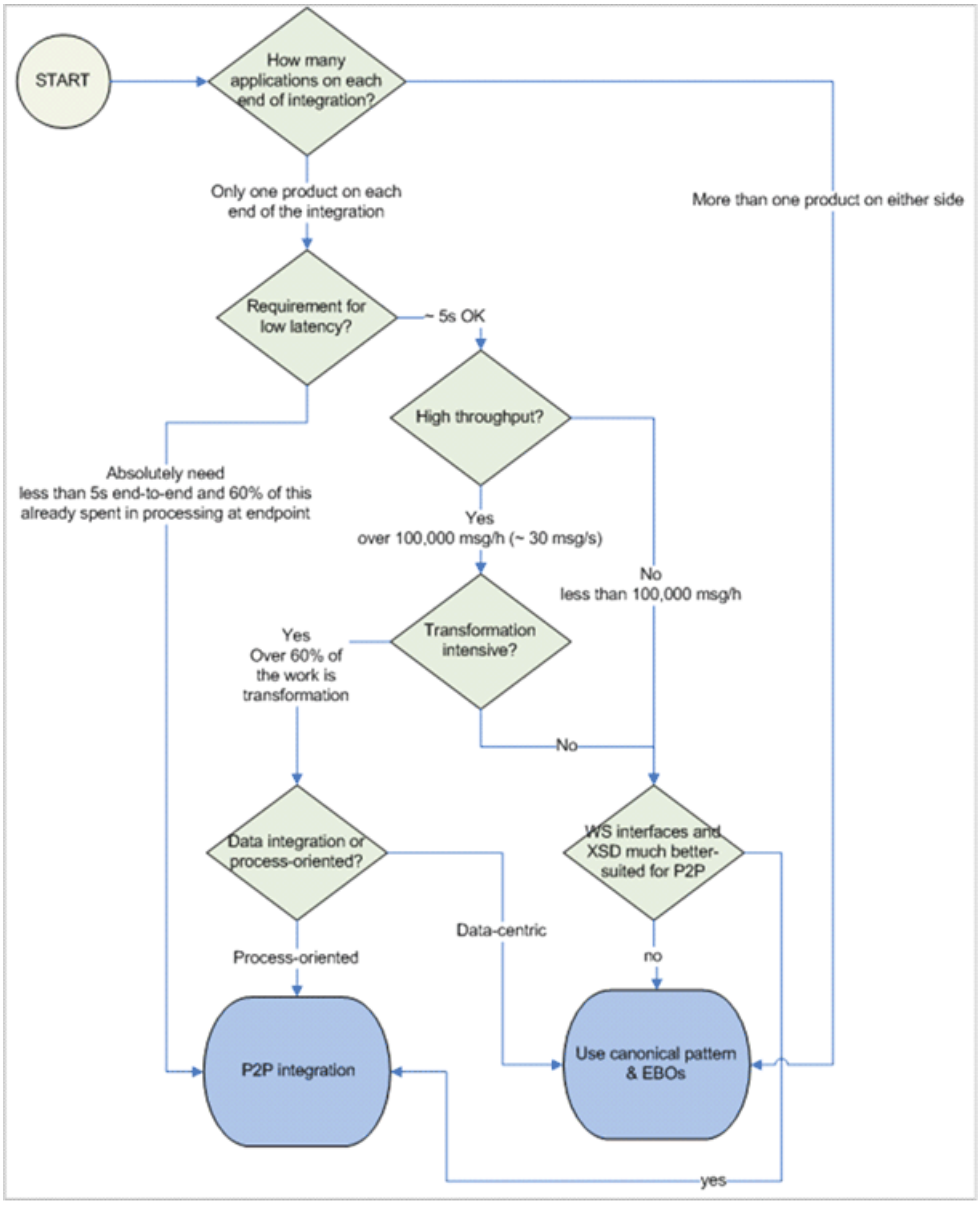
Oracle delivers several pre-built process-centric integrations. One helps orchestrate the conversion of an opportunity into a Quote or Order. Another orchestrates the bi-directional consolidation of Account, Contact, Opportunity, and Product information between an organization's on-premise solution and their On Demand instances in real time.

1.3.8 What Integration Pattern Should be Used: Direct Transformation Of The Data Objects or a Canonical Data Model And EBOs?

Use of canonical Enterprise Business Objects (EBOs) is an integration best practice, especially in integrations that involve connectivity with multiple source and destination systems. However, use of a canonical data model introduces some overhead and might introduce unnecessary engineering work.

Figure 1–4 illustrates an approach for deciding between a direct point-to-point transformation or a canonical data model and EBOs.

Figure 1–4 Decision Tree



1.4 AIA and Integration Styles

This section includes the following topics:

- Section 1.4.1, "Pre-built Integration Accelerators"
- Section 1.4.2, "Integration Through Native Application Interfaces Using the Oracle Applications Technology Infrastructure"
- Section 1.4.3, "Direct Integration Through Application Web Services Using Oracle SOA Suite"
- Section 1.4.4, "Integration Through Packaged Canonical and Standardized Interfaces Using Oracle Foundation Pack"

- Section 1.4.5, "Bulk Data Integration with an Extract, Transform, and Load Approach Using Oracle Data Integration Suite"

1.4.1 Pre-built Integration Accelerators

Oracle AIA delivers pre-built Integration Accelerators in two formats:

- Direct integrations typically use one or more styles of integration, but do NOT use the canonical pattern.
- Pre-Built Integrations typically use one or more styles of integration and DO use the canonical pattern.

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

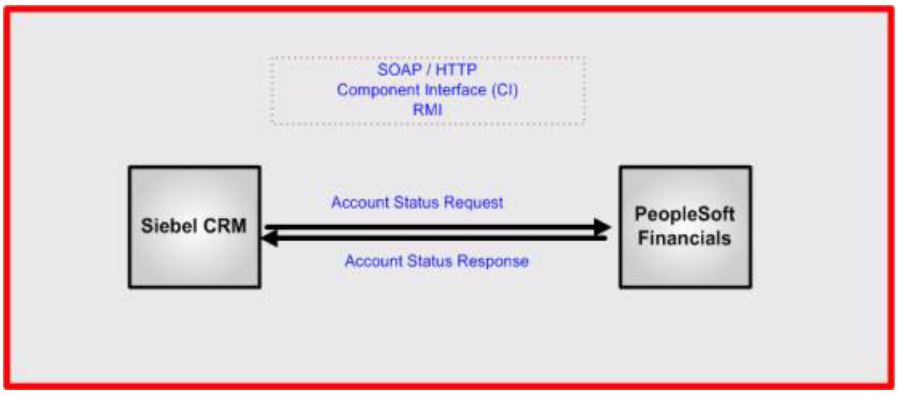
In this style, messages flow from the requester application to the provider application. The mode of connectivity could be SOAP/HTTP, queues, topics, or native adapters. No middleware is involved in this integration.

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, and 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 must be available in the individual applications.

Figure 1–5 illustrates how a requester application interacts directly with the provider application.

Figure 1–5 Example of a Requester Application Interacting Directly with a Provider Application



In more complex situations in which the integration flow consists of multiple steps involving interactions with multiple applications, one or more applications must leverage workflow-like capability.

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

1.4.3 Direct Integration Through Application Web Services Using Oracle SOA Suite

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

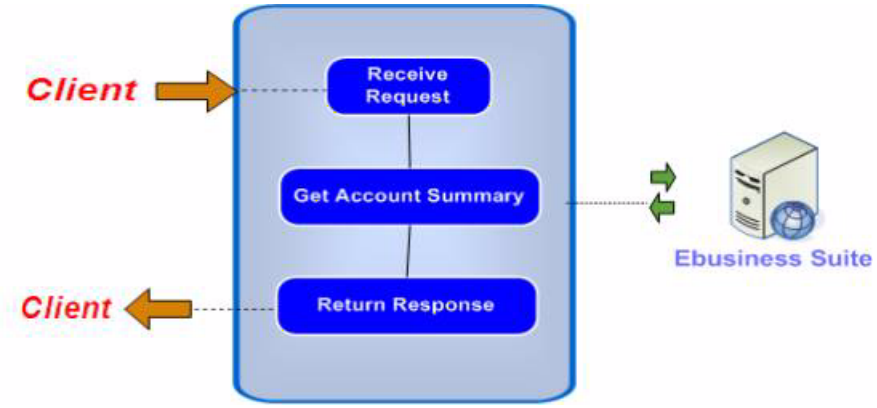
Several business initiators can invoke this AIA service. If the business initiators cannot present the request in the format understood by the provider's application-specific AIA service, a requester's application specific-AIA service is used to transform the business initiator request into the provider's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

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

The integration flow consists of a requester's application-specific AIA service artifact deployed on the middleware that manages interactions with all provider application-specific AIA services.

Figure 1–6 illustrates how a service deployed to the middleware enables the integration between the requester and the provider applications.

Figure 1–6 Example of Integration Flow Leveraging Provider Services



In more complex situations in which the integration flow involves interactions with multiple applications, the requester's application-specific AIA service implements a workflow-like capability and manages interactions with all the provider's application-specific AIA services.

The complexity of the data exchange and the various message exchange patterns involved, determine the AIA service artifacts that must be developed.

1.4.4 Integration Through Packaged Canonical and Standardized Interfaces Using Oracle Foundation Pack

Use SOA for loose coupling through a canonical (application-independent) model to simplify integration involving many instances of the same or similar participating applications. Participating applications in loosely coupled integrations communicate through a virtualization layer. Instead of direct mappings between data models, transformations 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 integrations at the business logic tier, this is the ideal integration pattern, because the reusability gained is worth the increased overhead

cost. Most important, this isolates the impact on the overall integration of changes to one of the participating applications. This effectively brings the cost of maintaining these more complex integration scenarios in line with the costs of a single point-to-point integration.

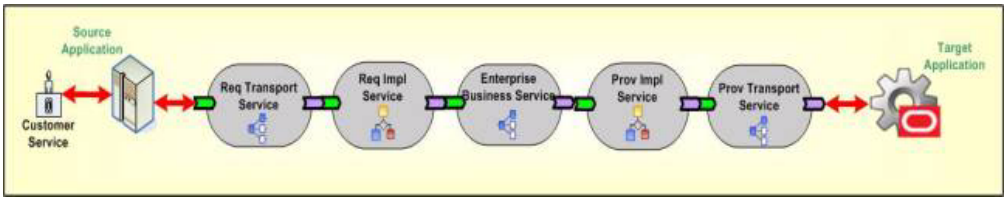
In this case, an Enterprise Business Service (EBS) based on relevant Enterprise Business Objects (EBOs) and Enterprise Business Messages (EBMs) is created as a mediator service.

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 transforms the business initiator request to the provider service format.

Figure 1–7 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. The request and provider transport services are optional. They are needed only for non-SOAP-based transports.

Figure 1–7 Example Using Canonical Model-based Virtualization



In more complex situations in which the integration flow involves interactions with multiple applications, the requester's 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.

The mediator AIA service interface chosen must be 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 complexity of the data exchange and the various message exchange patterns involved, determine the AIA service artifacts that must be developed.

1.4.5 Bulk Data Integration with an Extract, 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. This is a point-to-point integration for data movement with high performance but less reusability than a canonical model.

1.4.5.1 Integrations for High-Volume Transactions without an Xref Table

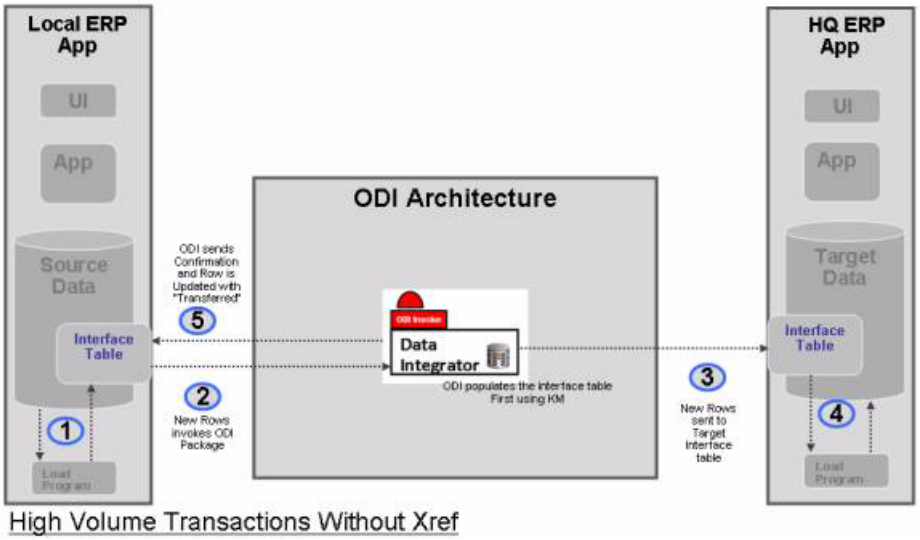
When storing cross-reference (Xref) data for high-volume transactions does not make sense, AIA recommends using a point-to-point integration using Oracle Data Integrator.

For example, suppose a retail store chain headquarters receives business transaction data from individual retail stores at the end of each day after each store closes. Each

store sends a day's worth of business transactions to headquarters according to its closing time and time zone. Because no data manipulation language (DML) operates on such data sets, no storage of Xref data between local stores and headquarters is needed.

Figure 1–8 illustrates a high-volume transaction integration without an XRef table.

Figure 1–8 Point-to-Point Integration Using Oracle Data Integrator



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. After 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.

For more information about bulk data integration, see "Using Oracle Data Integrator for Bulk Processing" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

1.5 AIA Reference Process Models

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

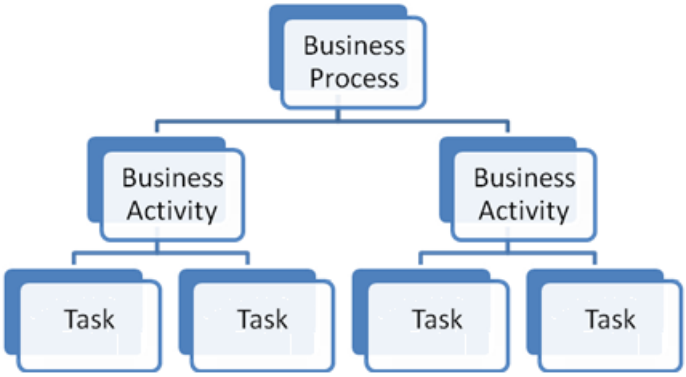
Reference Process Models serve the following purposes:

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

For more information about Reference Process Models, see the *Oracle Fusion Middleware Reference Process Models User's Guide for Oracle Application Integration Architecture Foundation Pack*.

Figure 1–9 illustrates the relationship of business processes, activities, and tasks that is described in the following sections.

Figure 1–9 Relationships Between Business Processes, Business Activities, and 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. Coordination of business processes from different business areas creates business value.

1.5.2 What is a Business Activity?

A Business Activity is a coordinated set of Tasks. A Business Activity is equivalent to a subprocess. Examples include Process Payment, Ship Goods, and so on. One application or a combination provides business activity functionality.

1.5.3 What is a Task?

A 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, which leads 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
- Crossing boundaries within and between businesses
- Dependent on and supportive of human intelligence and judgment

- Difficult to recognize

1.6 The Conceptual View of AIA

This section includes the following topics:

- Section 1.6.1, "What is a Service Consumer?"
- Section 1.6.2, "What are AIA Conceptual Services?"
- Section 1.6.3, "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 task in an enterprise. It has knowledge of the available AIA Conceptual Services and can present requests accordingly.

1.6.2 What are AIA Conceptual Services?

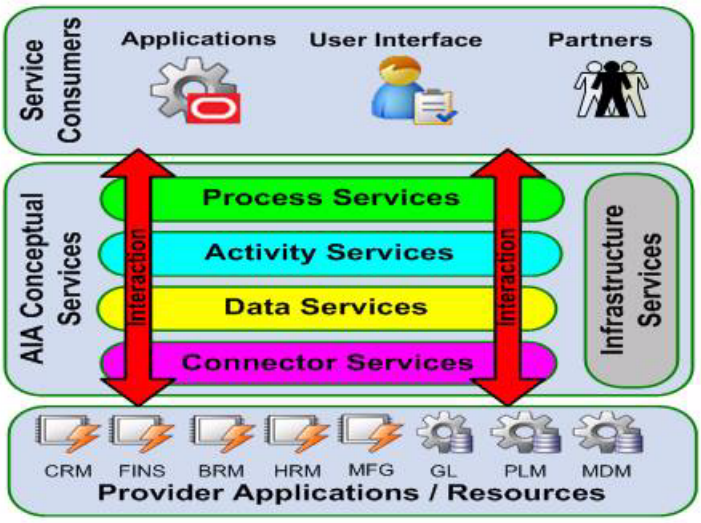
AIA Conceptual Services are developed using Oracle Fusion Middleware technologies. They constitute the service portfolio for the SOA implementation and enable the following:

- 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 into Process Services, Activity Services, Data Services, Connector Services, and Infrastructure Services.

Figure 1–10 depicts the conceptual services of AIA.

Figure 1–10 Conceptual View of AIA



1.6.3 What are Provider Applications and Resources?

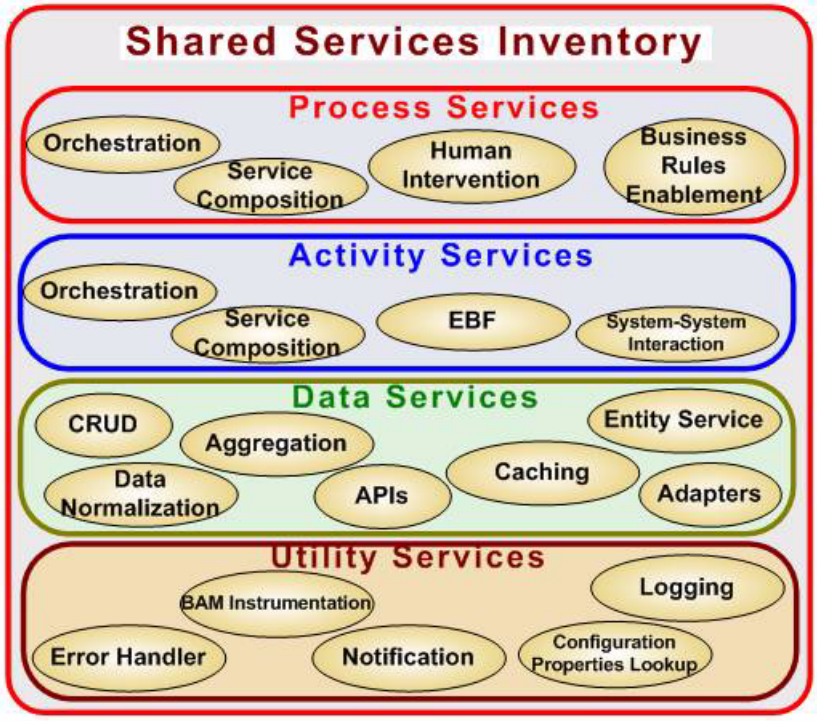
Various applications acquired for their best-of-breed functionalities and any legacy applications built in-house are Provider Applications and Resources. Each of these applications exposes business functions, such as APIs, and can be accessed using different modes of connectivity.

1.7 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 relate Shared Services to Reference Process Model artifacts. Figure 1–11 depicts elements in the Shared Services Inventory.

Figure 1–11 Elements in the Shared Services Inventory



1.7.1 What is a Process Service?

A Process Service is the implementation of major business events 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.

Process Services in AIA define and automate business processes that are external to and independent of the specific back-end systems used in the organization. This shields business processes from back-end system changes. Similarly, applications are

isolated from business process changes. Loose coupling between business processes and applications simplifies changes and maintenance for both.

Figure 1–12 depicts some characteristics of Process Services.

Figure 1–12 Process Services

The structure of the information packets passed around must be rich enough to capture the significant event details. These can be custom-defined or accepted canonical messages, 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 is either canonical or a user-defined format.

Figure 1–13 depicts some characteristics of Activity Services.

Figure 1–13 Activity Services

Activity Services are analogous to the Business Activities and 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 using 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.

Figure 1–14 depicts some characteristics of Data Services.

Figure 1–14 Data Services

Data Services are analogous to 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.

Figure 1–15 depicts some characteristics of Utility Services.

Figure 1–15 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 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.

Following are two Process Service examples.

Example 1 Resolve Customer Complaint Process Service

A business process defined to resolve complaints could:

- Analyze a complaint and identify activities.
- Schedule a call, block resources, and assign a technician.
- Track activities and send statuses.
- Close the complaint, generate metrics, and update statuses.

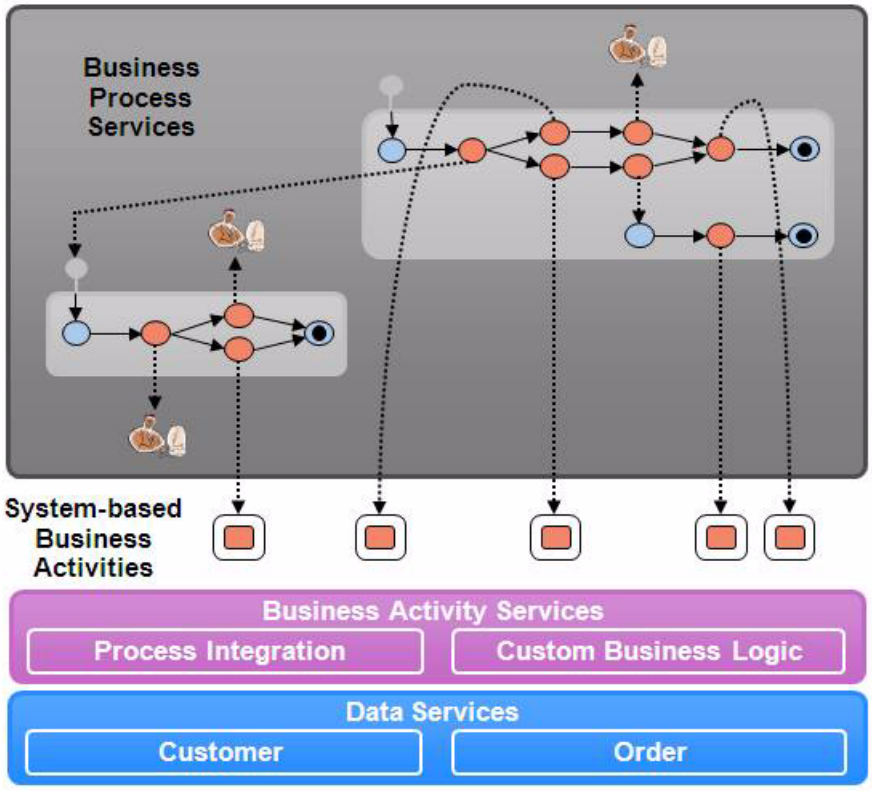
Example 2 Settle Auto Accident Claim Process

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.

Figure 1–16 illustrates a process service implementation.

Figure 1–16 Process Service implementation



1.7.6 Implementing Activity Services

If the EBS operation invokes an Enterprise Business Flow (EBF), the EBS AIA service artifact is the implementation of an activity service. 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 tasks. Typically, an orchestration flow implements Activity Services. The orchestration flow connects a set of tasks. Activity Services can leverage other activity services and data services.

Following is an Activity Service example.

Example 1 Creation of Customer in Billing Systems

An activity service can create a customer in a billing system using an order document. This service may be composed of the following steps:

- Retrieval of a set of account identifiers from an order document
- Retrieval of account particulars from a CRM system
- Creation and update of customers in a billing system

1.7.7 Implementing Data Services

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

Data Services are analogous to 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 AIA Service Artifacts

This section includes the following topics:

- Section 1.8.1, "What is a Composite Business Process?"
- Section 1.8.2, "What is an Enterprise Business Service?"
- Section 1.8.3, "What is an Enterprise Business Flow?"
- Section 1.8.4, "What is an Application Business Connector Service?"

1.8.1 What is a Composite Business Process?

A Composite Business Process (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 drive the flow.

1.8.2 What is an Enterprise Business Service?

An Enterprise Business Service (EBS) is a first-class object exposed on the enterprise service bus. An EBS is coarse-grained and performs a specific business activity or 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 to a specific back-end implementation. Back-end implementations that support EBS interface standards can be considered service providers. EBSs expose coarse-grained, message-driven interfaces for exchanging data between applications, both synchronously and asynchronously.

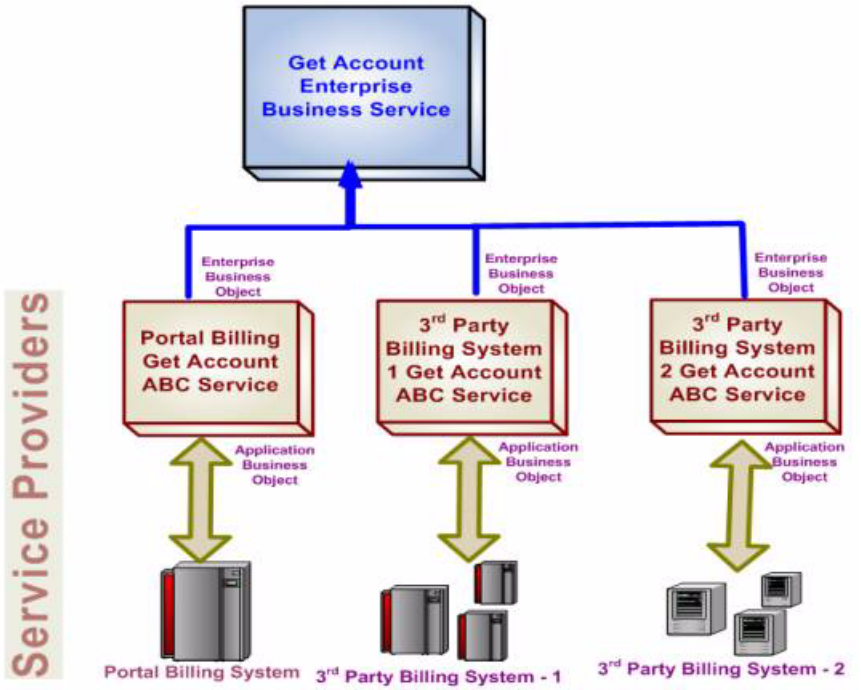
The salient features of EBSs include:

- Reuse of available assets in the Oracle portfolio.

- Substitution of a service provider with another without any impact to the client.
- Content-based selection of the service provider.

Figure 1–17 shows how an EBS enables the loose coupling of requesters with actual service providers.

Figure 1–17 An EBS Enabling the Loose-coupling of Requesters with Actual Service Providers



1.8.3 What is an Enterprise Business Flow?

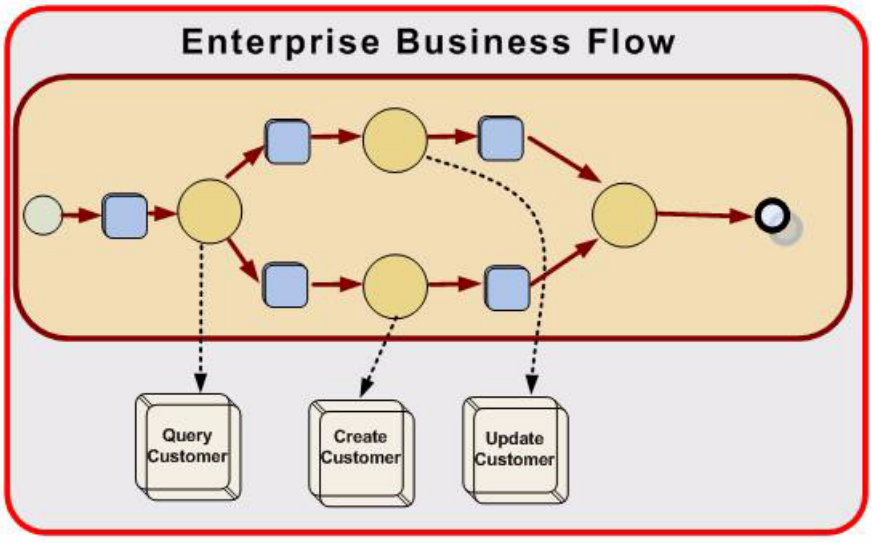
An Enterprise Business Flow (EBF) implements a business activity or a task that involves leveraging capabilities available in multiple applications. An EBF strings a set of capabilities available in applications to implement a coarse-grained business activity or task and composes a new service that leverages 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 never calls an ABCS or application directly. In other integration styles, the caller invoking the EBF can be an application or any other service.

Oracle Service Bus can implement an EBF for stateless coordination of synchronous or one-way business activities, where BPEL coordinates asynchronous activities.

Figure 1–18 illustrates how the EBF orchestrates a flow for synchronizing customers between the source application and the target application. When the source application invokes the sync operation, the EBF first determines if the customer exists in the target application. If so, it updates the customer in the target application. If not, it creates the customer in the target application.

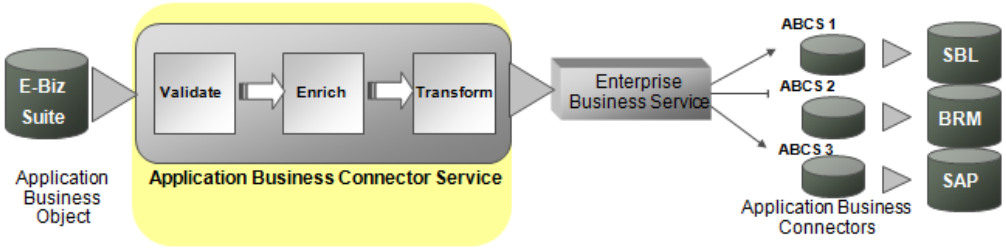
Figure 1–18 Example of an EBF Orchestrating the Flow from a Source to a Target



1.8.4 What is an Application Business Connector Service?

An Application Business Connector Service (ABCS) implements the EBS interface by exposing the business functions provided by the participating application. Figure 1–19 illustrates how the ABCS functions in the integration flow.

Figure 1–19 ABCS in the Integration Flow

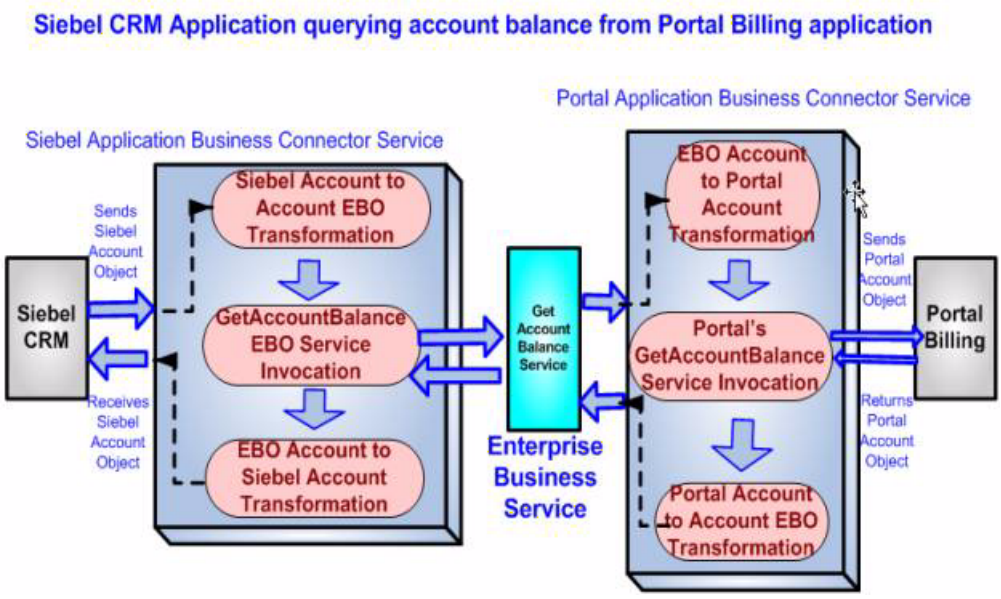


The ABCS coordinates the various steps provided by several business functions exposed by a provider application. For example:

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

Figure 1–20 illustrates an ABCS implementation, in this case the CRM application querying an account balance from the billing system.

Figure 1–20 ABCS implementation



2

Understanding EBOs and EBMs

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 architecture, uses, and creation of context-specific EBO views.

This chapter includes the following sections:

2.1 Introduction to EBOs

An EBO is a standard business data object composed of reusable data components. The library of all EBOs comprises a data model. The EBO represents a layer of abstraction on top of the logical data model. Developers, business users, and system integrators use EBOs. 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. Each application data schema is mapped to the EBO data model only once. This significantly minimizes 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.
- 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.
- They are represented in an XML schema (XSD) file format.

An EBO represents a business concept, 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.

2.1.1 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.

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

2.1.2 Common Components

Common components are reused 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 used by many different EBOs. The content model of Address contains sufficient information for an application to identify, create, and update it as necessary when it is supplied as part of an EBO's content. 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 common business component and a data model entity such as table or a foreign key. 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 of one of these common objects is automatically reflected in all EBOs that reference that object. For example, if your implementation requires customizing an Address format by adding a third address line, this modification automatically affects EBOs that reference the Address. This design philosophy significantly reduces the design, development, and maintenance of common objects.

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

2.1.3 Reference Components

The same attributes identify both an EBO business component and its associated reference business component, but the latter includes a minimal subset of attributes required to qualify the EBO business component. PurchaseOrderLineReference and InvoiceLineReference are examples of reference business components. 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.

In addition to the complete definition of an EBO, each context in which the EBO is used has its own definition.

2.2 Introduction to EBMs

At the most basic level, EBMs are the messages 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 require most elements present in the EBO. However, the context for canceling the invoice might require only the specific invoice instance to be canceled.

To create context-specific EBM definitions, assemble common and EBO-specific business components. You can often obtain business components from multipls EBOs. You can then use these context-specific EBO definitions in context-specific EBMs. In this scenario, the ProcessInvoice EBM includes the process-specific invoice definition and the CancelInvoice EBM includes the cancel-specific invoice definition. Use these EBMs as 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. For example, a Customer Party EBO schema module is available and a Customer Party EBM schema module represents the entire concept for the business object.

For more information, see Chapter 6, "Understanding Extensibility".

2.2.1 EBM Architecture

Every EBM has 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 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 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 multiple types of action. For example, you cannot have *Query* and *Update* actions in the same message.

When using EBMs, consider the following points:

- Any application invoking the Enterprise Business Services (EBSs) has 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 use EBMs that support multiple instances.

- The information present in an EBM header is common to all EBMs.

  The information 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.

2.3 EBM Headers

The EBM header is an integral part of every EBM. The EBM header is like a wrapper or envelope around transactional data messages. It comprises functional data representations. For example, Involved Parties might include 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.

Infrastructure-related service requirements such as auditing, logging, error handling, and security necessitate additional attributes in the EBM message header section.

For more information, see "Working with Message Transformations" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

3

Understanding Enterprise Business Services

This chapter introduces Enterprise Business Services (EBS) and explains why an EBS is needed to facilitate an integration. The chapter then discusses EBS types and EBS operations. Finally, it provides a high-level overview of the tasks involved in the EBS creation.

This chapter includes the following sections:

For more information about EBS, see " Designing and Developing Enterprise Business Services" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

3.1 Introduction to EBS

EBSs are the foundation blocks in the Oracle Application Integration Architecture (AIA). An EBS represents the application or implementation-independent web service definition for performing a task. The architecture facilitates distributed processing using an 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 applicable for each operation of a service.

An EBS is self-contained; that is, it can be used independently of any other services. In addition, it can also be used within another EBS. EBSs are business-level interfaces. They are standard service definitions that applications participating in the integration can implement. 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 an EBS has a well-defined interface described in XML. This interface description includes all details required for a client to independently invoke the service.

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

For more information, see Section 2.2.1, "EBM Architecture".

EBS components do not presuppose a specific back-end implementation of that service. They simply provide an integration layer for your choice of a back end. Regardless of the choice, you can still achieve the seamless interaction experience in the prebuilt integrations delivered by AIA. Any back-end implementations that support EBS interface standards are considered service providers.

For more information about EBS, see "Designing and Developing Enterprise Business Services" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

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 associated Verb.

For more information, see Section 3.3, "Verbs".

AIA makes an explicit distinction between operations that can process a single instance of a payload versus operations that can process multiple instances. 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. The Open Applications Group Integration Specification (OAGIS) originally introduced the concept of a verb in their business object document (BOD) definitions. The EBO definition adopted this concept with some modifications.

A web service does not use a Verb because its operation definition identifies the action it performs. However, not all integrations use web services, and message-oriented integration scenarios still exist that may require the processing action to be identified within the message.

Verbs define the semantics of operations and provide a consistent, unambiguous framework for naming operations and operation payloads.

For more information about verbs and how to use them, see "Constructing the ABCS " in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

3.4 EBS Types

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

3.4.1 Entity Services

An entity service includes all standard activities that an EBO must perform. 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 activity uses the EBM that represents the activity-specific view of the EBO as input and output.

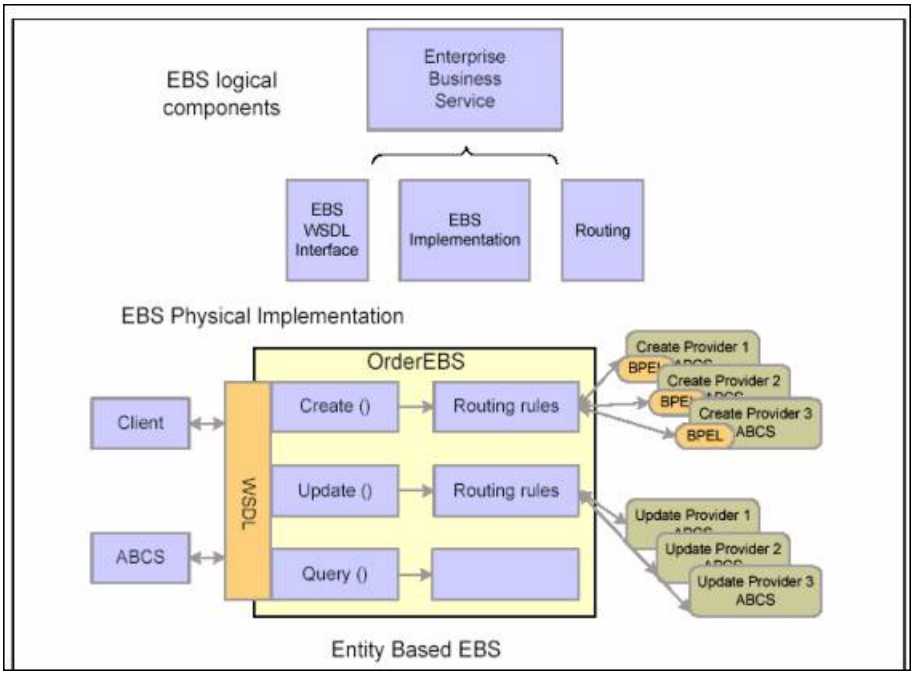
Entity services expose operations that act on a specific EBO.

Entity services have the following characteristics:

- The corresponding EBS for a business object entity exposes all actions that can be performed on this object as service operations.
- The entity operations consist of standard Create, Read, Update, and Delete (CRUD) operations, which are similar across services, and 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 requests to the appropriate application-specific Application Business Connector Service (ABCS) containing the actual implementation.

Figure 3–1 illustrates the EBS logical components and the EBS physical implementation of entity services.

Figure 3–1 Entity Services



3.4.2 Process Services

A process service includes all business activities acting upon a particular business object to fulfill a specific business process. 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 the Person EBS.

EBM interfaces are implementation-agnostic. Because of this application-independence, an EBS both expects an EBM as input and returns an EBM as output.

Process services expose operations related to an enterprise business process.

Process services have the following characteristics:

- The corresponding EBS for an Enterprise Business Flow (EBF) exposes all actions that can be performed on this flow as service operations.
- Process services are implemented as Mediator routing services.
- Process services receive and return messages in the form of EBMs.
- Operations the EBS exposes initiate cross-functional EBFs that coordinate complex long-lived flows spanning 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 an Entity Service, a Process Service can act on multiple EBOs.

Figure 3–2 illustrates EBS logical components for process services.

Figure 3–2 EBS Logical Components for Process Services

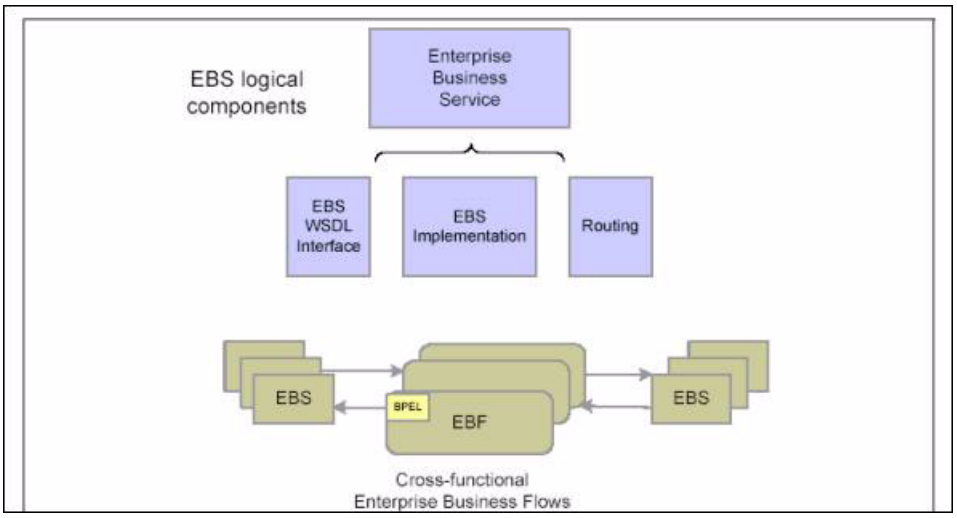
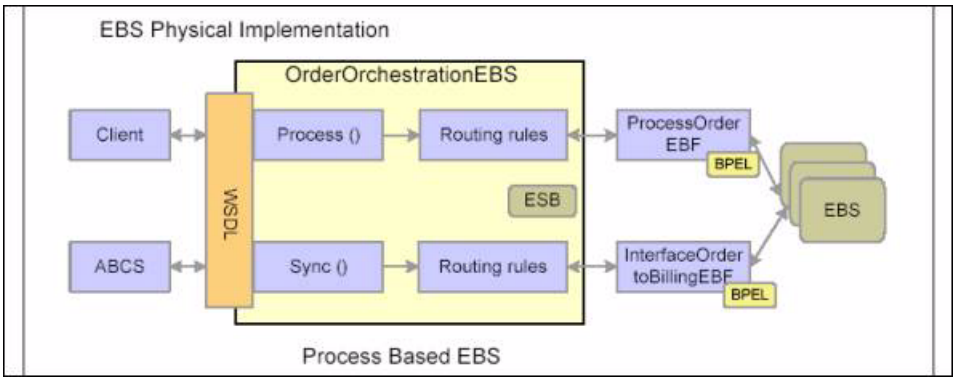


Figure 3–3 illustrates the EBS physical implementation of process services.

Figure 3–3 EBS Physical Implementation of Process Services



3.5 EBS Architecture

The EBS architecture enables:

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

3.5.1 Reuse of Available Assets

An EBS is a web service, so EBS interfaces are defined in Web Service Definition Language (WSDL). WSDL specifies EBS-specific service operations (business activities) and input and output arguments (messages) for each service operation. As with any web service, an EBS can be implemented in 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.

Although you can build EBS activities such as Create or Update from the ground up using web services technology, AIA takes advantage of existing application

functionality. Participating applications provide intermediary services in Application Business Connector Service (ABCS) format. The ABCS exposes application data and transaction-related business functions as services that an EBS can invoke.

The virtualization layer enables the following aspects of Oracle AIA:

- The physical implementation of the Service Provider (also called the endpoint or target) must be abstracted and its location hidden so another service can potentially replace it.
- 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 must be replaced with no interruption or change to the consumers.
- The virtual service may be composed of multiple physical services, 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).
- The target service and the service consumer might support different transport protocols.
- If complex, content-based validations against XML are required, then use Schematron 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 Section 1.8.4, "What is an Application Business Connector Service?".

3.5.2 Substituting One Service Provider for Another

AIA enables you to decouple the requester's view of a service from the actual implementation. This architectural flexibility allows service provider interchangeability. The requester is unaware and the EBS needs no alteration if the service provider is changed.

For example, the delivered integration between a CRM system and a financials system may use the service of the Oracle E-Business suite. At implementation time, you may want to substitute this service with one from another provider, such as PeopleSoft Financials. Substituting the service provider has no impact on the requester or the client.

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.

Routing rules can tell the EBS how to delegate a request to the right service provider and application instance. The services that invoke the EBS might have request delegation knowledge. In this scenario, the services might supply the information in the EBM before invoking the EBS.

For example, you may have two billing systems, one for Northern American customers and another for all others. The EBS evaluates this rule and deciphers the actual implementation necessary for processing a specific message. The clients and

service requesters are unaware of the implementers. Similarly, the implementers are unaware of the client applications that made the request.

The architecture allows an entire message (containing all instances) to be sent to only one service provider. Different messages can go to different providers, but instances within a message cannot be divided among two or more providers.

3.5.3 Content-Based Selection of the Service Provider

In a service-oriented architecture (SOA), there is no way to loosely couple a service entirely between systems. You must choose what aspects should 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 has no impact on the client.

However, at design time, the requester explicitly specifies 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 do not change. So changing the name of the EBS, the service operation, or the structure of the data formats impacts the client. As mentioned in the previous sections, an EBM is passed as a request, and another EBM is received as a response.

3.5.4 EBS Purpose

The purpose of the EBS is:

- To receive the request from the calling application.
- To identify the implementation and the deployment 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.

Figure 3–4 and Figure 3–5 show how EBS enables the loose coupling of requesters with the actual service providers.

Figure 3–4 EBS Enables Loose Coupling of Requesters with Service Providers

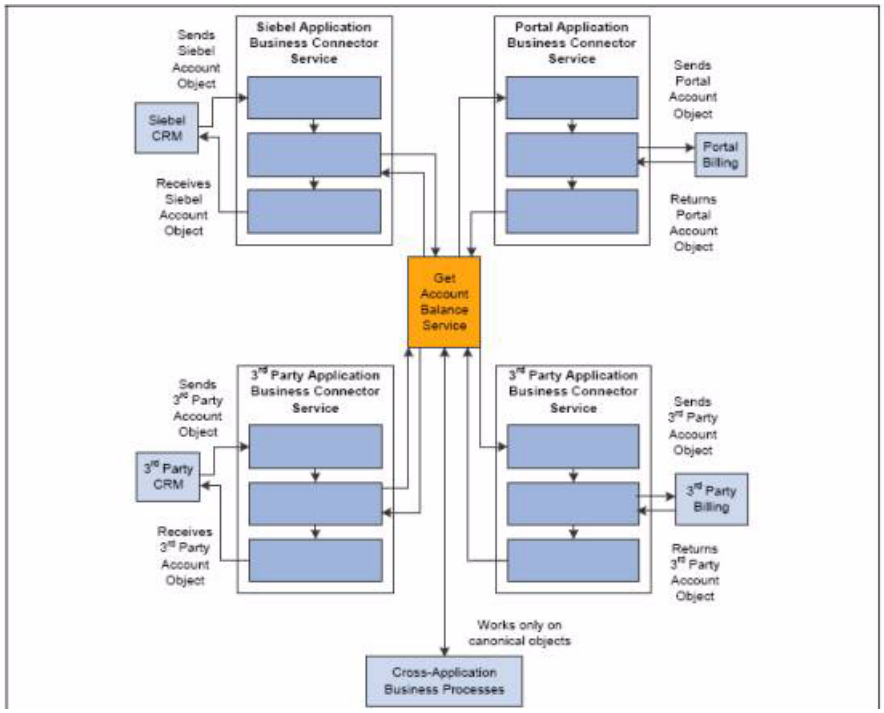
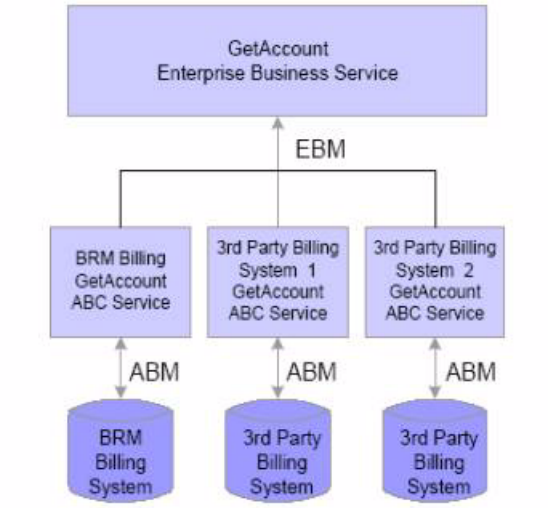


Figure 3–5 Example of GetAccount EBS



For more information about EBSs, see "Designing and Developing Enterprise Business Services" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

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 integration of applications from multiple vendors. Business processes always use an EBS.

For more information, see Section 3.1, "Introduction to EBS".

EBSs have application-independent interfaces. BPEL processes use them to interact with different applications. The payload of the EBS is the EBM. The EBM contains the EBO. Within a cross-application business process, the EBO holds the in-memory representation of the message that is sent back and forth between applications.

Figure 3–6 illustrates how a request coming from a participating application initiates a business process.

Figure 3–6 Participating Application Request Initiates a Business Process

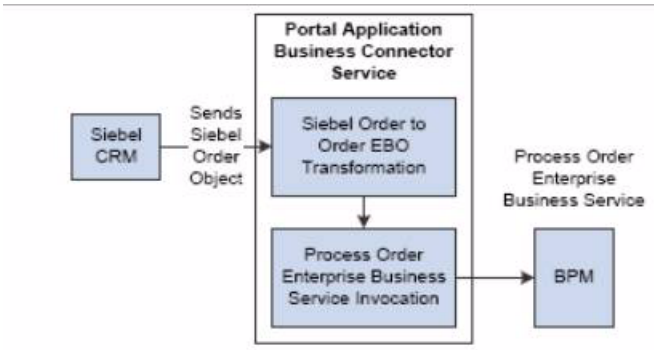
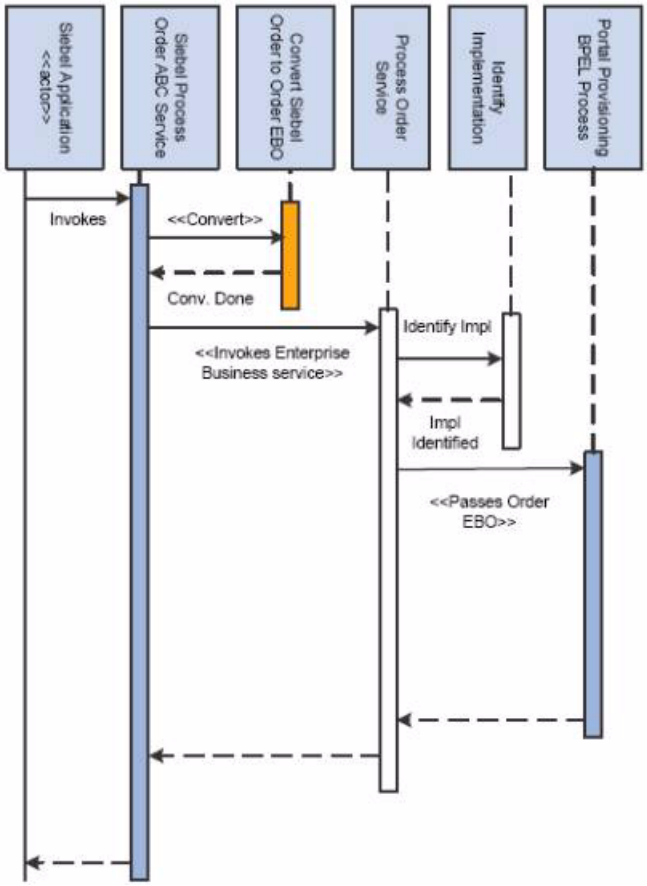


Figure 3–7 is a sequence diagram of events leading to a successful initiation of a BPEL process.

Figure 3–7 Invocation of Process Order



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

An application-independent BPEL process contains no steps specific to any particular participating application.

For more information about EBF, see "Designing and Constructing Enterprise Business Flows" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

3.7 EBS Implementation

Every EBO must have an EBS. The EBS defines and implements corresponding services for every action that can be done on the EBO. The EBS is a very lightweight Meidator routing service. Every service operation must have its own set of routing rules.

Example 3–1 shows the interface definition for each action that can be carried out on the Invoice EBO.

Example 3–1 Interface Definition

```
<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 a 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>
```

3.7.1 EBS Flow

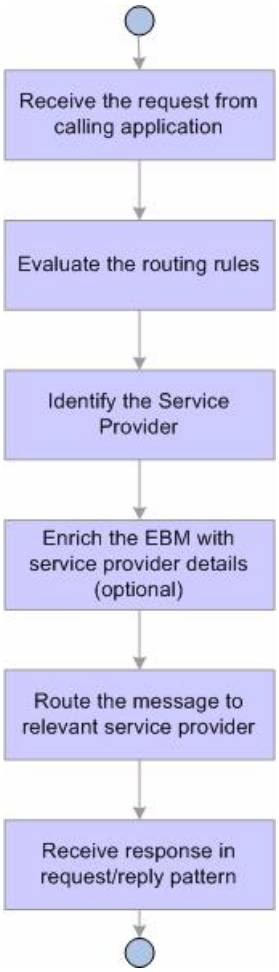
The EBS flow includes the following steps:

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 and 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 Oracle provides, 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 apply only to integration scenarios using the request-reply pattern. Figure 3–8 illustrates the steps an EBS performs:

Figure 3–8 EBS Flow



For example, invoking the Query Customer Party operation of the Customer Party EBS invokes the Query operation of the Customer Party EBO.

3.8 EBS Message Exchange Patterns

Business requirements drive the need for different message exchange patterns. This section defines message exchange patterns for various EBS operations.

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

An **asynchronous** operation does not wait for a response before continuing. This allows operations to occur in parallel. Thus, the operation does not block or wait for a response. Asynchronous operations open a communication channel between the parties, make the request, and close the channel before the response occurs. Message

correlation relates the inbound message to the outbound message. This is effective when large numbers of transactions take long periods to process. The asynchronous pattern is effective if the end user does not need immediate feedback. If the operations are short or must run in serial, synchronous operations may be more appropriate.

The EBS is designed to have multiple operations. Each operation executes the EBS for a particular business scenario requirement and is granular. Each operation can have the following interaction style or message exchange pattern:

- Synchronous request - response
- Fire-and-forget - no response
- Asynchronous request - delayed response

For more information about design patterns, see "Working with AIA Design Patterns" and "Establishing Resource Connectivity" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

3.8.1 Synchronous Request-Response Patterns in EBSs

A synchronous 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. Both the request and the response messages are independent.

The operations in the portType have input, output, and fault. Example 3–2 is from the SalesOrderEBS.wsdl and illustrates a synchronous request-response operation.

Example 3–2 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

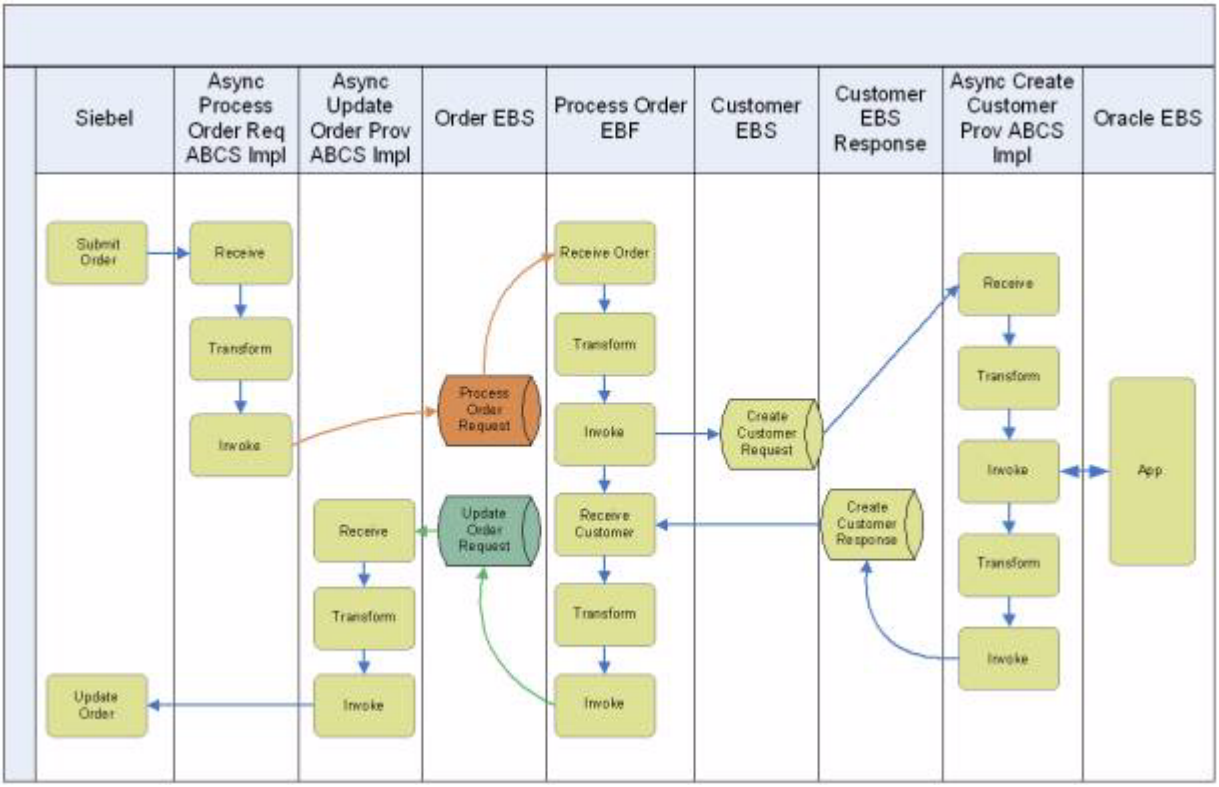
A fire-and-forget EBS operation pattern is asynchronous. TA request message is posted to an endpoint or placed in a channel/queue/topic. This pattern is also characterized as a one-way call.

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, not even an error. In the Order EBS WSDL, the portType defines the one-way request-only operation.

In a fire-and-forget pattern, a request presented to the provider cannot be re-presented. Errors in the provider service must be handled locally. This includes either retrying or terminating. In case the error cannot be handled locally, then compensation must be initiated, if required.

Figure 3–9, depicts the Order EBS fire-and-forget scenario. The Order EBS has two operations: Process Order Request and Update Order Request. Both of these fire-and-forget operations use one-way calls in the EBS WSDLs.

Figure 3–9 Order EBS Showing Fire-and-Forget Scenario



Example 3–3 is a sample from SalesOrderEBS.wsdl for a request-only operation.

Example 3–3 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 have input only. This contract requires the invoker to make one-way calls. For Entity EBS, the WSDLs are part of the Foundation Pack Enterprise Service Library. For Process EBS, the WSDLs are hand-coded based on provided template WSDLs.

3.8.3 Asynchronous Request-Delayed Response Patterns in EBSs

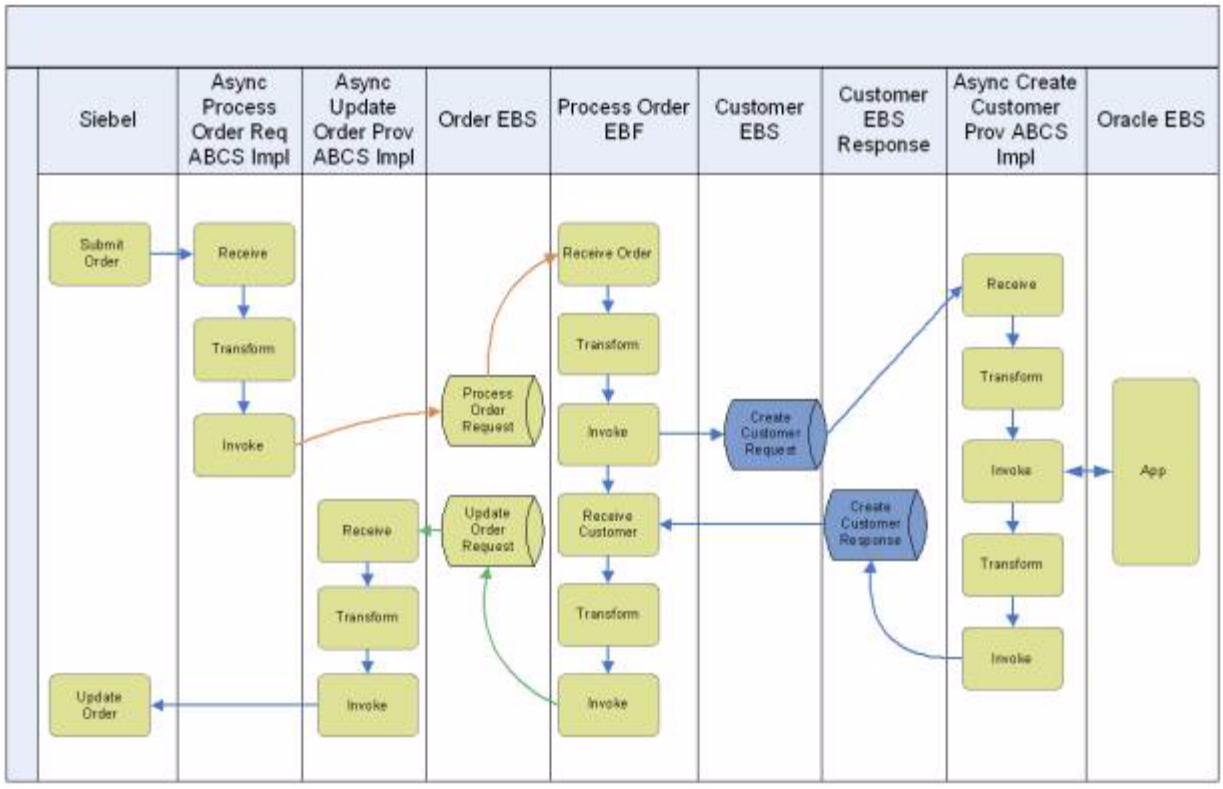
A request-delayed response EBS operation pattern is asynchronous. The requester sends the request message and sets up a callback for a response. The requester does not wait for the response after sending the request message. A separate thread listens for a response message. When the response message arrives, the response thread invokes the appropriate callback and processes the response. The EBS has a pair of operations-one for sending the request and another for receiving the response. Both operations are independent and atomic. A correlation mechanism establishes 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 includes fault information. In the Customer EBS WSDL, the portType defines the one-way request and response operations. The one-way operation meant for response is defined in the portType having all the operations, which are Response.

In Figure 3–10, 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 one-way calls in the EBS WSDL.

This pattern allows more flexibility in error handling. In case of an error, the provider sends a response to the requester service, which re-presents or resubmits the request. In some situations, the provider service can handle errors locally too, similar to the fire-and-forget pattern. The business use case scenario dictates the methodology. Figure 3–10 illustrates a one-way request and one-way response scenario.

Figure 3–10 One-way Request and One-way Response



To increase the reliability of message delivery, use persistence at strategic asynchronous points.

For more information about guaranteed message delivery designs, see "Configuring Oracle AIA Processes for Error Handling and Trace Logging" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

Example 3–4 is a sample from SalesOrderEBS.wsdl for an asynchronous request-response operation.

Example 3–4 Asynchronous 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:InitiatorInterface>

<svcdoc:InitiatorOperation>CreateSalesOrderRequest</svcdoc:InitiatorOperation>
              </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 are hand-coded based on provided template WSDLs.

4

Understanding Application Business Connector Services

This chapter introduces Application Business Connector Services (ABCS) and provides an overview of ABCS architecture and characteristics, architectural considerations for implementing ABCS, extensions and customizations for ABCS, processing multiple instances, and ABCS transformations.

This chapter includes the following sections:

For more information about ABCS, see "Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4.1 Introduction to ABCS

The ABCS exposes a participating application's business functions in a representation compatible with a service interface.

In canonical integration style, an Enterprise Business Service (EBS) exposes the common service interface. The common service interface allows participating applications to invoke EBSs.

In non-canonical 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 use the service interface defined by the provider to invoke the EBS.

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 requester ABCS enables the participating application to invoke the EBS to access data or perform a transactional task. The client side ABM is the payload passed by the requester application to the requester ABCS.

The requester application must define the requester-specific ABCS. The requester application could be Siebel CRM, PeopleSoft Enterprise CRM, or Oracle eBusiness Suite CRM. The requester application-specific ABCS must take the requester application-specific ABM as input and provide the requester application-specific ABM as output.

The **provider ABCS** exposes the business capabilities available in the provider application according to the EBS specification. In certain non-canonical integration styles, the provider defines the EBS interface. In other situations, the EBS artifact acts as an abstraction layer. The service-provider-side ABCS accepts the request either from the EBS through the Enterprise Business Message (EBM) or directly from either Requester Application or Requester ABCS and sends the response using the same format. The ABCS transforms different application object representations to the canonical definition, making communication between applications possible.

The ABCS coordinates various steps provided by several services, including:

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

For every activity of an Enterprise Business Object (EBO), the participating requester application and the service provider application must each define an ABCS.

4.2 ABCS Architecture

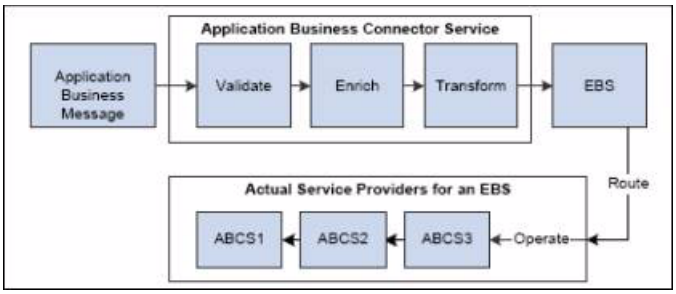
The Oracle Application Integration Architecture (AIA) requires messages sent from applications providing business functions to follow either EBS or provider ABCS (in non-canonical integration styles) standards. Applications meeting these standards can participate in cross-application business processes and prebuilt data integrations.

Similarly, applications receiving messages from an EBS must be able to understand them. For applications not designed to interact with an EBS, the ABCS exposes the application-specific business functions in the Mediator.

The ABCS uses a variant of the industry-standard integration pattern called VETO: Validate, Enrich, Transform, and Operate. The VETO pattern and its variations ensure that consistent, validated data is routed throughout the Mediator. A common variation is the VETRO pattern, which adds a Route step.

The ABCS architecture adds an extra Operate step to the VETRO pattern, resulting in the VETORO pattern illustrated in Figure 4–1.

Figure 4–1 VETORO Pattern



The VETORO pattern consists of the following steps:

- Validate: This step ensures that the content of the incoming message can be transformed for the target system. In certain cases, the validate step can check whether the incoming message contains a well-formed XML document in conformance with the schema of the receiving application. This is often one of the first steps in the EBS. It allows 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 translates an application-specific object representation into an EBO, or translates an EBO-specific representation into an application-specific object representation.
- Operate: This step invokes the target service or an interaction with the target application. In Oracle AIA, it could invoke the EBS, a specific ABCS, or an interaction with the participating application using the provider ABCS.
- Route: The Route step deciphers the appropriate service provider responsible for performing the service. In Oracle AIA, the Route step is implemented in the EBS using content-based routing technologies available in Oracle Mediator.

In some cases, one ABCS implements the validate, enrich, and transform steps. Also, a Mediator routing service may use XSL-based validation and content-based routing directly in the service itself, rather than using a separate routing service.

The ABCS does not handle transport protocol abstraction itself. The ABCS has inbound and outbound interactions using only the SOAP/Mediator protocol. A transport adapter integrates the participating application-specific native protocols with the ABCS-specific standard. This facilitates reuse of the same ABCS for multiple transport adapters and vice versa.

4.3 ABCS Characteristics

The ABCS has the following characteristics:

- For every activity of an EBO (canonical pattern-based integration style), the participating requester and provider applications must each provide an ABCS.
- In a non-canonical pattern-based integration style, the provider application must provide an ABCS.
- The requester ABCS has participating application-specific ABMs as input and output.
- The service accepts requester application-specific ABMs as input and provides requester application-specific ABMs as output.

- The provider ABCS has EBMs (representing EBO-specific content needed for performing the operation) as input and output.
- The service accepts EBMs as input and returns EBMs as output.
- Although a single ABCS can 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 must have the activity information accepted as a part of the input. The ABCS can then decipher the action to be performed and the appropriate transformations and invocations. In addition, allowing a single requester ABCS to handle multiple activities means that a single requester application-specific ABM encompasses 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 and the design paradigms you 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 exactly matches that of an application-agnostic or provider application-specific interface.

If an exact match exists, your main task is to transform the application-specific ABM into the enterprise EBM and vice versa. Mediator is an implementation technology candidate for building the ABCS.

If the granularity of the application's web service does not match that of the EBS, enhance or modify the application if possible. If you cannot modify the application, the ABCS must handle transformations and aggregate and disaggregate services.

For example, suppose 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 probably has a chatty conversation with the provider application. The provider ABCS is also responsible for state management.

You must implement this type of ABCS through BPEL technologies, not through Mediator services.

Although Oracle AIA allows this type of ABCS, Oracle highly recommends encapsulating much of this application logic in native applications instead of in the ABCS.

4.4.2 Support for EBMs

Because the EBS operates only on EBMs, you must determine whether the applications that implement the services provide support for EBMs. If application-provided services support EBMs, transforming the EBO into an EBM is easier. If the applications that implement the services do not support EBMs, you must determine whether their services can support EBMs.

If these applications cannot support EBMs, the ABCS must perform transformation-related work.

4.4.3 Application Interfaces

Interfaces determine how participating applications allow business logic to interact with the Mediator. Some applications support web service interfaces. The WSDL defines the interface that communicates directly with the application business logic. In this preferred scenario, the ABCS uses the web service interface to invoke the application business logic.

In packaged applications such as Siebel, PeopleSoft, J.D. Edwards, and SAP, the respective packaged-application adapters are preferred. These adapters can be deployed as J2CA resource adapters. This is a better solution than using the conventional SOAP interface. When the participating applications do not expose their business logic as web services, database adapters, advanced queuing (AQ) adapters, or other adapters are needed.

Investigate whether services exposed by the participating applications support proprietary message formats, technologies, and standards. If the applications do not support standards and technologies such as XML, SOAP, and JMS, then the ABCS must handle transformations.

For example, suppose the application receives and sends messages only through files and recognizes only EDI format. In this case, the ABCS integrates with the application using a file adapter, translates EDI-based messages into XML format, and exposes messages as SOAP messages.

For more information, see "Establishing Resource Connectivity" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4.4.4 Support for Logging and Monitoring

The ABCS facilitates logging and monitoring. The ABCS invokes convenience services for logging and auditing.

For more information, see *Oracle Fusion Middleware Infrastructure Components and Utilities User's Guide for Oracle Application Integration Architecture Foundation Pack.*

4.4.5 Support for Insulating the Service Provider

The granularity of the service provider's service may 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 EBS interface expects the service provider to return complete details about a bill. In this case, the ABCS protects the application implementing the service by doing workload buffering. The ABCS makes 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 largely determines ABCS responsibilities. Support for a point-to-point security model requires only that the ABCS authorize the service

requests. Support for an end-to-end security model necessitates the transmission of requester credentials to the service provider.

The latter scenario can use Web Services Security if all participating systems support it. Otherwise, security credentials can be transmitted as application data or in the SOAP header. In either case, the ABCS must be coded accordingly for authentication.

4.4.7 Validations

As a message travels between participating applications, validations must ensure that accurate data reaches the target system. Validation specificity varies significantly depending on distance from from the target system.

Participating applications providing services impose validation constraints. ABMs generated from EBMs must obey these constraints. In addition, the target system's validation rules may reject a message unless a transformation step alters the incoming data.

Because the EBM cannot enforce all participating application-specific constraints, the specific ABCS must enforce them. 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 must ensure that the ABM generated from the EBM includes the product description. Thus the ABM passed to the PeopleSoft CRM Create Product service complies with service constraints.

The validation for a constraint very specific to the PeopleSoft CRM application must 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.

Because validation during capture creates a better user experience in asynchronous interaction styles (request-only), the requester application should perform the validation at another integration point before submitting the message for processing. For example, in an Order Capture application, the requester application makes a synchronous call to validate the content before submitting the order for fulfillment. A Validate Order or Validate Billing Information EBS might be available for various service providers. If the response from the Validate Order service is favorable, the requester application then makes the request to process the order.

4.4.8 Support for Internationalization and Localization

The ABCSs that pass the EBMs to the actual service providers translate the documents into locale-specific ABMs. Similarly, the ABCS translates the locale-specific ABM into the locale-independent EBM.

The ABCS deciphers the locale from the user preferences for the relevant participating application. This data specifies how the locale-specific ABM must be constructed, and how the locale-specific ABM must be interpreted.

The ABCS specifies the locale of the service provider's response. For example, the ABCS for the Get Product Details EBS specifies the locale in which the Oracle eBusiness Suite application provides product details. If the requester wants the product details in Spanish, the Get Product Details EBS must instruct the real service provider accordingly.

4.4.9 Message Consolidation and Decomposition

You may need to combine responses to a request that originates from multiple sources. For example, in convergent billing in the telecommunication solution, the ABCS for the getBillDetails EBS might have to retrieve details from multiple participating applications. This ABCS also consolidates them into a single response.

4.4.10 Support for Multiple Application Instances

You may have multiple instances of a packaged application with the same business capabilities in your company's ecosystem. The EBS routing rules decipher the right application instance to receive the request. Regardless of the number of application instances for a packaged application, only one ABCS exists for that packaged application to perform a specific task.

4.5 Implementing ABCS

Each application implementing a business activity or task has its own ABCS.

An ABCS for a particular participating application should implement a single action. The action is the same on both sides of a synchronous request-response pattern. In a delayed response pattern, one ABCS implements the request action and another implements the response.

The ABCS can be implemented in two ways:

- Use only Oracle Fusion Middleware components.

  A Mediator service or BPEL process performs the tasks listed in the following sections.

- Build the transformation services mainly in the participating application.

  Use this approach when the participating application's technology stack can perform the transformations. However, a lightweight ABCS still must perform the translations related to cross-reference details.

For more information about ABCS, see "Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

Regardless of how an ABCS is implemented, it is still a service, so an interface is exposed as a WSDL. The client applications use this interface to invoke the ABCS.

XSL scripts implement client-side and server-side ABCS transformations.

The ABCS itself must be independent of deployment. Only one ABCS instance exists even for multiple deployments of a single client or provider application. For example, suppose two BRM Billing application deployments existed, one for North American customers and another for all others. A single ABCS would invoke the application business service in the appropriate deployment.

In many situations, you cannot map a single action to a single API or operation in the provider application. The provider application might have very fine-grained operations or require calling applications to handle transaction management. In this case, the provider-side ABCS probably has a chatty conversation with the server application. The provider-side ABCS also handles state management.

You must implement this type of ABCS using only BPEL technologies, not Mediator services.

Although Oracle AIA allows this type of ABCS, Oracle highly recommends encapsulating application logic within native applications instead of in the ABCS.

The service populates the EBM message header section with values. The service deciphers some values by itself, and receives other values from participating applications through an ABM.

For more information, see Section 2.2.1, "EBM Architecture".

4.5.1 Requester-Side ABCS

The requester-side ABCS has the following core responsibilities:

- Enrichment or augmentation of the ABM.

  The ABM from the participating application may lack required content. For example, a CRM application might pass only the order identifier in the ABM. The ProcessOrder EBS interface might require the entire order object. In this situation, the ABCS must request all required information from the participating application 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 Section 2.2.1, "EBM Architecture".

- Invocation of any extension handler that you may have registered.

  The extension handler can perform additional transformations. The extension handler is passed to the EBM and the transformed EBM is passed back as the response. You can perform additional transformations on the EBO before the EBS is invoked.

- Invocation of the EBS.

  The EBS uses an EBM to provide the response.

- Transformation of an EBM into an ABM.
- Invocation of any extension handler that you 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. You can perform additional transformations on the ABM before the ABM is passed back to the calling application.

- Validation of the ABM passed to the participating client application to ensure compliance with constraints enforced by the participating application.
- Return of the response to the calling application.

Figure 4–2 illustrates the high-level flow of activities in a requester-specific ABCS. The EBS and ABCS interact in a request-response style. The steps for running the extension to do additional transformations are optional.

Figure 4–2 Requester-Specific ABCS Implementing the Request-Response Style Interaction Pattern

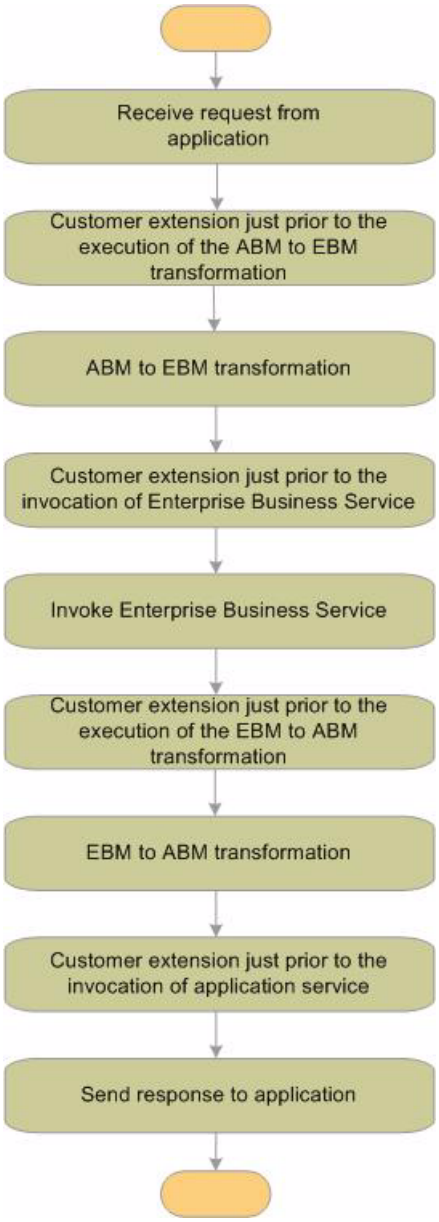
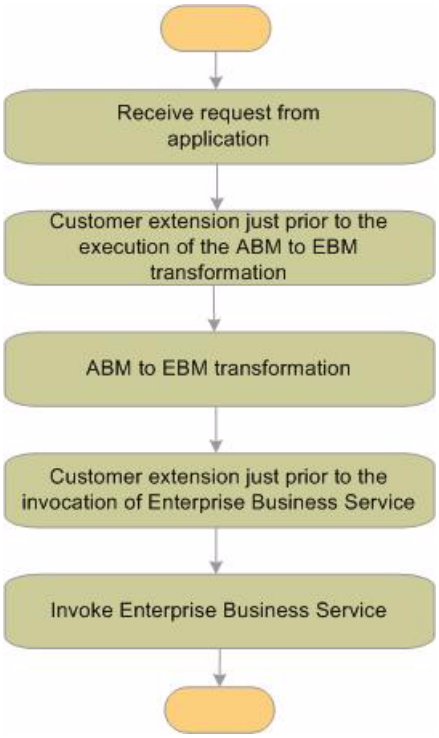


Figure 4–3 depicts the high-level flow of activities in a requester-specific ABCS. The EBS and ABCS interact in a fire-and-forget style. The steps for running the extension to do additional transformations are optional.

Figure 4–3 Requester-Specific ABCS Implementing the Fire-and-Forget Style Interaction Pattern



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 you may have registered.

  The extension handler can perform additional transformations. The extension handler is passed to the ABM and the transformed ABM is passed back as the response. You can perform additional transformations on the ABM before the ABM is passed to the provider application business service.
- Validation of the ABM passed to the participating provider application to ensure compliance with constraints enforced by the participating application.
- Invocation of the provider application business service.

  Invocation can consist of one or multiple calls.

  For more information, see Section 4.4, "Architectural Considerations."
- Transformation of an ABM into an EBM. The response message sent by the provider application must be returned to the caller application. In this case, it would be the EBS. Because the EBS expects only the EBM as output, the ABM must be transformed 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 Section 2.2.1, "EBM Architecture".
- Invocation of any extension handler that you may have registered.

The extension handler can perform additional transformations. The EBM is passed to the extension handler and the transformed EBM is passed back as the response. You can perform additional transformations on the EBM before the document is passed to the EBS.

- Return of the document to the EBS.

Figure 4–4 depicts the high-level flow of activities in a provider-specific ABCS. The EBS and ABCS interact in a request-response style.

Figure 4–4 Provider-Specific ABCS implementing Request-Response Style Interaction Pattern

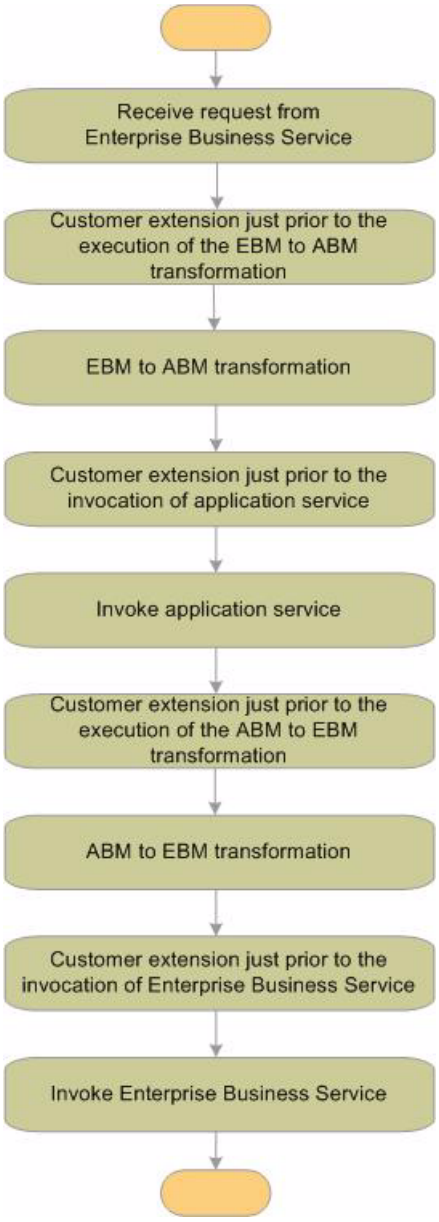
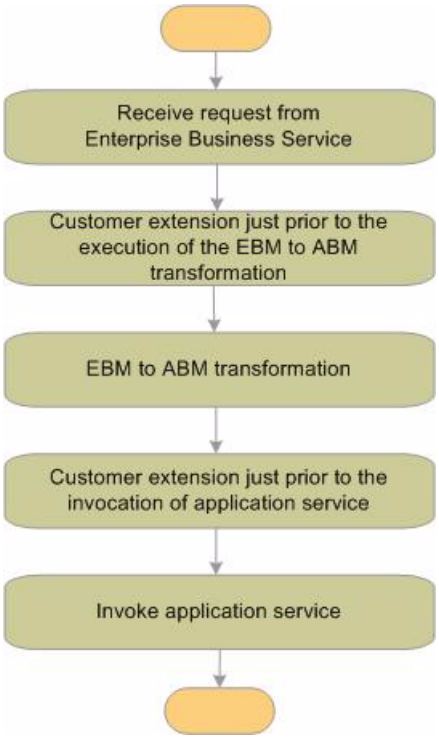


Figure 4–5 depicts the high-level flow of activities in a provider-specific ABCS. The EBS and ABCS interact in a fire-and-forget style.

Figure 4–5 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

Use the Oracle Mediator blueprint when you do not need additional enrichment or content validation. In this model, the ABCSs are implemented as Mediator services.

To configure a different transport, add transport adapters before the client-side ABCS or after the provider-side ABCS.

For more information about using Oracle Mediator, see "Designing and Developing Enterprise Business Services" and "Tuning Integration Flows" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

4.6.2 BPEL

Use BPEL when the ABCS must augment or validate content. In most situations, the ABCS must call one or more participating applications to enrich the content. It may also have to handle state management.

BPEL can also extend the connector. You can implement the ABCS using procedural object-oriented languages such as Java or C++.

Use BPEL when a Mediator service cannot implement the ABCS due to technology constraints or complexities.

For more information about using BPEL, see "Designing Application Business Connector Services," "Constructing the ABCS," and "Completing ABCS Development"

in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4.7 Extending or Customizing ABCS Processing

With Oracle AIA, you can use different transports without modifying 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 that invokes the participating application's services without modifying the ABCS.

A client-side or provider-side ABCS implementing the request-response pattern provides four extensibility points. A requester ABCS provides two extensibility points before EBS invocation and two after the EBS responds. A provider ABCS provides two extensibility points before application-specific service invocation and two after the application service responds. An ABCS implementing the fire-and-forget pattern has only two extensibility points. You can inject additional behavior, such as custom validation or transformations, at extensibility points. You can use transformations for additional elements introduced at the implementation site. You can introduce additional processing without having to customize the delivered ABCS. The extension service is passed to either the EBM or the ABM, depending on the situation. The content is passed as context to the extension service, which returns the content after performing the alterations.

For more information about ABCS, see "Completing ABCS Development" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack.*

4.8 Processing Multiple Instances

Because an EBM can carry multiple instances, the provider ABCS must iterate through the instances and process them. Not all EBMs support carrying multiple instances. EBMs created specifically for supporting bulk processing can support multiple instances.

An ABCS can decide whether its application can 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 cannot 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 invocation, the requester application passes the ABM to the ABCS. The requester application can either pass everything an ABCS needs or pass the bare minimum. In the latter approach, the ABCS fetches from the participating application the details relevant to composing an EBM. Although the architecture supports both approaches, Oracle highly recommends that the participating applications use the first approach to minimize overhead.

Because the ABCS encloses an EBO in an EBM, the participating application must pass much of the EBM header content. For this reason, in addition to passing the business content, the participating application is also passing data related to the source environment. This information associates an EBM with the originator.

For more information, see Section 2.2.1, "EBM Architecture".

The participating application also specifies the locale used to construct the ABM. This locale interprets 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 ABCS performs transformations and invokes the participating application's services. The transformations result replace application-specific fields with some generic fields and vice versa. Transformations also replace static and non-static application-specific identifiers with a common identifier. Non-static identifier-related transformations use the Mediator cross-reference facility. The design patterns used and the type of participating application determine the types and number of transformations done in a single ABCS. A single ABCS can interact with either a client-side or provider-side participating application.

For more information about ABCS, see "Working with Message Transformations" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

4.10.1 Transformation: Implementation Approach

You can implement ABCS transformations in two ways. First, an ABCS for a particular action can invoke an XSL file that performs all the transformations.

Second, the architecture allows you to provide transformations for additional elements introduced as part of a customer-specific implementation. Customer-specific transformations do not modify Oracle AIA artifacts and are unaffected by Oracle AIA upgrades.

The ABCS transformations handle:

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

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

- 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, a Country common component object may use the full country name, such as United States of America, while an ERP application may

use a two-letter code, such as US. 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 available in the Mediator facilitates 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 data objects are the same means 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 is that 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 Mediator and BPEL services that create specific integration scenarios between named participating applications. The services use diverse interactive styles and patterns for exchanging messages. This chapter lists various patterns, highlights their features, and presents guidelines for choosing them based on integration scenarios.

This chapter includes the following sections:

For more information, see "Working with AIA Design Patterns" in *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

5.1 Introduction to Patterns for Exchanging Messages

The business requirements of an integration scenario drive the need for different message exchange patterns between participating applications.

An application integration leverages the core functionalities of the best applications and links tasks 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. Triggering events can invoke interactions that are synchronous, asynchronous, or a combination.

Synchronous operations wait for a response before continuing. This forces operations to occur in a serial order. 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 is effective unless many channels are left open for long periods. 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 open a communication channel between the parties, make the request, and close the channel before the response occurs. Message correlation relates the inbound message to the outbound message. This is effective when many transactions take long periods of time to process. If the operations are short or must run in serial, synchronous operations may be more appropriate. The asynchronous pattern is effective if the end user does not need immediate feedback.

The following sections describe basic message exchange patterns and variations.

5.2 Request-Response

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

5.2.1 Synchronous Response

Suppose an application that requests information from an external system has to wait for the response.

Use case

A CRM application gets account details from a billing application. The service representative clicks a button in the CRM application, which sends an Application Business Message (ABM) to the Siebel Get Account Details Application Business Connector Service (ABCS), which invokes the Get Account Details Enterprise Business Service (EBS). The CRM application waits for the Siebel Get Account Details ABCS to return the ABM before rendering the information on the screen.

Synchronous request-response

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

5.3 Fire-and-Forget

Suppose a customer integrating two applications does not want the sender application to be affected if the receiving application is unavailable.

Use case

A customer wants to convert opportunities in the CRM On Demand application to quotes in the CRM On Premises application in real time or near real time. However, non-availability of the CRM On Premises application must not impact the CRM On Demand application. Most important, work done with CRM On Demand cannot be lost.

Asynchronous transaction using queue/topic

Oracle AIA uses queues for asynchronous and reliable message delivery. When a business event occurs, CRM On Demand 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 enters the message in a queue. CRM On Demand considers a message sent when it is dropped into a queue. CRM On Demand can continue sending new

messages regardless of CRM On Premises availability. A JMS consumer (a Mediator/BPEL service with a JMS adapter) dequeues each message and invokes the CRM On Demand ABCS. This transaction ensures that a message is removed from the queue only after the CRM On Premises application successfully completes its task.

5.3.1 Message Routing

Suppose a customer wants to route requests to different applications providing the same business function based on certain criteria.

Use case

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 does not want the CRM system that gets bill details for customers to also route requests to the appropriate billing system.

Message routing/mediation

Oracle AIA architecture can decouple the requester and provider. For this use case, *Get Bill Details* is the EBS operation. Each billing application provides an 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 to invoke. For EMEA customers, vendor A's ABCS is invoked. For North American customers, vendor B's ABCS is invoked. Each vendor's ABCS gets bill details from their applications, converts the details to Enterprise Business Message (EBM) format, and sends the EBM to the EBS.

For more information, see Section 1.8.2, "What is an Enterprise Business Service?".

Suppose a customer wants to route requests to different instances of the same application based on certain criteria.

Use case

The customer has two Oracle BRM application instances: A for servicing EMEA customers and B for servicing North American customers. The customer does not want the CRM system that gets bill details for customers to also route requests to the appropriate BRM instance.

Message routing/mediation

Oracle AIA architecture can decouple the requester and provider. For this use case, *Get Bill Details* is the EBS operation, which is implemented as a Mediator routing service. The BRM application has only one ABCS, which is integrated only with the Get Bill Details EBS. The customer must define a transformation service that adds target system information to the EBM header, which the BRM ABCS uses to route the request to the right BRM instance.

For more information, see Section 1.8.2, "What is an Enterprise Business Service?".

5.3.2 Message Splitting and Routing

Suppose a business document with multiple line entries needs each line to be handled differently.

Use case

The customer uses vendor A's billing system for broadband customers and vendor B's billing system for wireless customers. A CRM system sends a Process Order Request EBM that can contain requests for both services. Vendor A's billing system receives the

broadband-related portion of the order and vendor B's billing system receives the wireless product-related portion.

Message splitting and routing

The CRM system (CRM ABCS) invokes the Process Order EBS operation. The implementation is a BPEL process that splits the order business document into multiple business documents, each having data for only one order line. Each order line business document is passed to another EBS, such as Activate Service, which uses the routing rules to decipher the billing system to use.

5.4 Data Enrichment

Suppose a requester application does not send all the data required 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.

Data enrichment of the ABM

Data enrichment may be required when the ABM received from the participating application lacks required content. The ABCS must use web services (or JCA-based adapters, if available) to get the information required to enrich the ABM from the participating application. The participating application must expose the needed business capabilities as web services.

5.5 Data Aggregation

Suppose a provider application does not return all the data the EBS operation requires.

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 (information for all of the services in one bill) for a customer. It passes the customer identifier and the month as input parameters to the service provider. It expects the complete bill details from the provider. The billing application has a web service that provides only a bill summary. The application does not have a single service that provides complete details. However, it does have services to get details for every part of the bill.

Data aggregation of the ABM

You may need multiple interactions with the provider application to get all the needed content. Then you must combine the content to return a single response to the EBS. The billing application ABCS interacts with the provider application using three web services to retrieve the bill header, bill summary, and bill details. After retrieving all of the content, the ABCS combines the three ABMs and produces a single EBM.

Suppose information needed for providing a 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 customer. It passes the customer identifier and the month as input parameters to the service provider. It expects the complete bill details

from the provider. However, the customer has three billing applications, each with information about only one service: wireless, broadband, and land line.

Data aggregation of the ABM

The ABCS for the Get Bill Details EBS must retrieve details from multiple billing applications. It interacts with the applications using the services they provide. After retrieving all of the content, the ABCS combines the three ABMs and produces a single EBM.

5.6 Asynchronous Request - Delayed Response Pattern

A request-delayed response pattern is asynchronous. The requester sends a message and sets up a callback for a response, but does not wait for it. When the response arrives, a separate response thread invokes the appropriate callback and processes the response. The EBS has two operations; one for sending the request and another for receiving the response. The operations are independent and atomic. A correlation mechanism establishes 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 that waits for the asynchronous response. If an error occurs in the providing service, the response includes fault information. In the Customer EBS WSDL, the portType having all the Request-Response or Request only operations defines the one-way request operation. The portType having all the Response operations defines the one-way response operation.

5.7 Publish-and-Subscribe

In many integration scenarios, participating applications publish events and messages to which multiple participating applications subscribe. This pattern is transactional: it changes 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 "Establishing Resource Connectivity" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

6

Understanding Extensibility

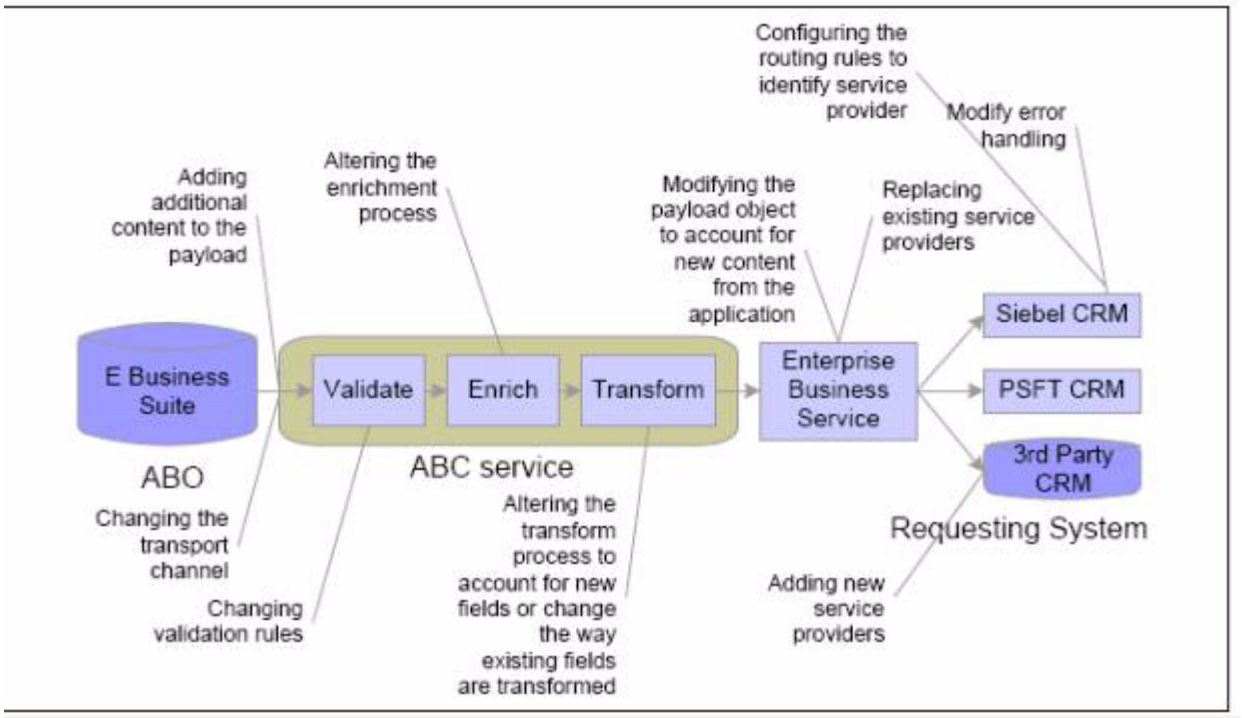
This chapter provides an overview of extensibility and discusses considerations for schema extensions, transformation extensions, transport/flow extensions, process extensions and routing extensions.

This chapter includes the following sections:

6.1 Introduction to Extensibility

The Oracle Application Integration Architecture (AIA) allows customers to extend various artifacts of prebuilt integrations. These extensions are protected during upgrades, although postupgrade configurations may be needed to point to the artifacts. The artifacts are designed and constructed with native extensibility support. Figure 6–1 shows various extensibility points in the integration flow and provides examples of how the flow might be extended at each point.

Figure 6–1 Integration Flow — Extensibility Points



6.2 Schema Extensions

Oracle AIA allows Enterprise Business Objects (EBOs) to be extended to meet industry-specific and customer-specific needs.

The EBO structure that Oracle AIA delivers and provides upgrades for is separate from and does not affect any extensions.

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 such identification, naming conflicts might occur between different extensions or between extensions and future additions to the core specification.

- A straightforward path should exist to identifying details about the extension.

Two approaches are available for implementing extensibility:

- Customer-specific extensions
- Industry-specific extensions

6.2.1 Customer Extensions

Every EBO component has an element at the end to accommodate applicable customer-specific attributes. For example, a customer may complete the data type definitions for data elements of interest.

For information about extension-enabling an EBO and adding customer-specific attributes, see the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

6.2.2 Industry-Specific Extensions

The architecture allows for industry-specific attributes as overlays. An industry-specific object combines business components available at the core with industry-specific components. It's not clear what this means. Here's my best guess: Only Oracle can create industry-specific EBOs. Customers can add industry-specific content using the approach described in Section 6.2.1.

6.2.3 Schema in the Use Case

You can extend the Get Account Balance use case to retrieve customer usage details from the billing system. Suppose the BRM web service exposes this information. You can render the usage details on a Siebel screen along with other information from BRM. Extend the Enterprise Business Message (EBM), specifically the Query Customer Party Response EBM.

6.3 Transformation Extensions

The transformation scripts in prebuilt integrations are extension-ready. This ensures that customer-defined transformations are nonintrusive and customer-specific transformation-related extensions are durable.

Every component in the EBM, including the EBM header, contains an extensibility point for transformations. Oracle provides a transformation script exclusively dedicated to housing customer extensions. You can add code to the templates in this script to specify transformation rules for the new content.

In this release, the XSLT extensibility programming model allows customers to add maps only to their new elements in the EBM. You cannot override existing maps for certain elements. Similarly, you cannot add maps to existing elements having no maps in the Pre-Built Integrations.

6.3.1 Extensions in the Use Case

The Get Account Balance use case illustrates how to send the extensions to the Siebel screen. The Oracle BRM Application Business Connector Service (ABCS) for Query Customer Party transforms the Application Business Message (ABM) into an EBM. Suppose that the BRM web service has retrieved the details and made them available in the ABM. Now you must ensure that the ABM content is also in the EBM.

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

The Enterprise Business Service (EBS) operation Query Customer Party returns the Customer Party Response EBM to the Siebel Query Customer Party ABCS. The script in the ABCS transforms the EBM into an ABM, which is then sent to the Siebel application. Now the Siebel ABM gets the ABM from the ABCS and can display the Usage Details on the screen.

6.4 Transport/Flow Extensions

You can change the transport channel in a nonintrusive manner for messages traveling between the participating application and the connector services. For example, the prebuilt integration might use SOAP/HTTP to transport the message between Siebel CRM and the connector service. At implementation time, you might decide to ship the

data using a file instead. You can reconfigure the transport without customizing the delivered artifacts.

6.5 Process Extensions

Oracle recommends ways for customers to extend the architecture of the ABCS and the orchestration processes. These orchestration processes are Composite Business Processes (CBPs) and Enterprise Business Flows (EBFs).

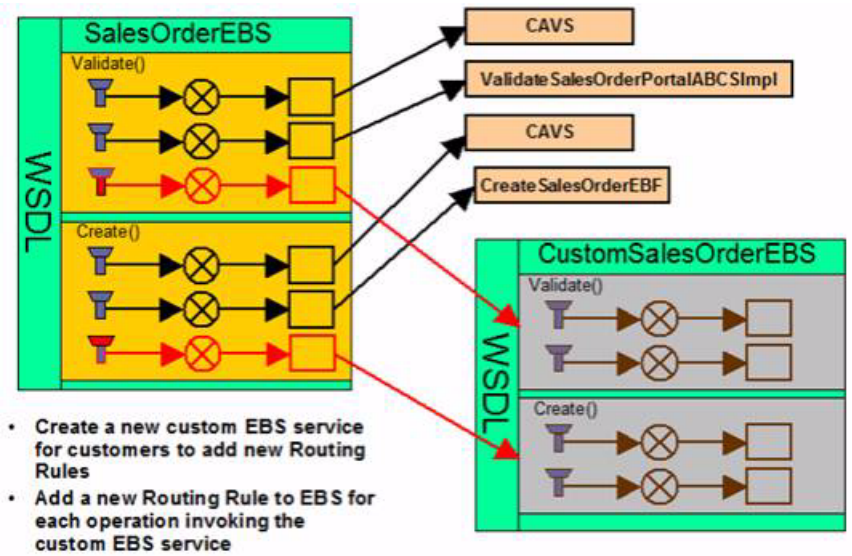
Each BPEL process can have its own extensibility points based on functional needs. You can implement the interface for each extensibility point to augment or override the behavior. Oracle recommends that an ABCS for a request-response pattern have a minimum of four extensibility points. An ABCS for a fire-and-forget pattern is expected to have a minimum of two extensibility points. The orchestration processes might not have any extensibility points.

Refer to the documentation for the appropriate Pre-Built Integrations to check whether they to support process extensions.

6.6 Routing Extensions

Figure 6–2 shows how to add custom routing rules using the custom extension points in the custom EBS. You can route a message to any homegrown applications or services plugged into Oracle AIA to extend service provisioning for any service request.

Figure 6–2 Routing Extensions



For information about whether a Pre-Built Integration supports routing extensions, see the relevant implementation guide.

7

Understanding Versioning

This chapter describes 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 includes the following sections:

7.1 Schema Versioning

EBOs evolve over multiple generations as you add, remove, or change the semantics or characteristics of 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 EBO in the library has its own release life cycle, so each object has its own version number. EBO version numbers are independent of participating application release numbers. New releases of participating applications do not necessarily result in new EBO versions.

The version number consists of the major and minor version numbers.

The following changes, which may break EBO backward compatibility, require a new major version number:

- 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

The following changes require a new minor version number:

- 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 type of change, not the size, determines whether a new version is major or minor. A small change warrants a major version change if it breaks backward compatibility. Similarly, enormous changes may result in a minor version change if backward compatibility is not broken.

The Enterprise Object Library contains all major versions of every EBO, but only the latest minor versions within the major versions. Hence, the schema file in the folder for the major version always contains the latest minor version.

Consumer applications that depend on an earlier major release might need to be modified to work with the new release. For example, an application written against version 1.0 works when targeted against versions 1.1 and 1.2, but may fail if moved to version 2.0 of the EBO unless modified.

Because Oracle AIA uses a service-oriented architecture for request/response interaction, backward and forward compatibility surface at the same time. An upgraded provider application must be backward compatible to understand requests from older requesters. Similarly, requesters must be forward compatible to recognize the responses of the provider application. Compatibility in both directions, at least among minor versions, ensures independence of providers and requesters.

The architecture does not mandate (in most situations) concurrent upgrade of the requester and provider, so additional XML message transformations between older and newer major versions are required. In some cases, these transformations may not be technically feasible or may not make functional sense. In these situations, the applications receiving the messages cause an irrecoverable error.

The **schema declaration version attribute** in the XML schema and a required custom attribute in the XML instance document identify the schema version. An XML instance specifies its namespace and minor version. 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. For example:

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

7.1.2 Namespaces

Each EBO has its own namespace. This minimizes duplication of names and allows each business object to 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 is the same across multiple minor and major versions and is changed only when a schemas undergoes major architectural changes. Introducing backward-incompatible changes alone does not warrant namespace changes.

Here is a sample namespace:

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

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

Second-layer namespaces import the innermost layer namespaces. Each second-layer namespace has declarations that are specific to a business process or functional area. For example, the PurchaseOrder namespace contains all documents used for placing a purchase order.

In addition, customer-specific namespaces house customer-specific extensions.

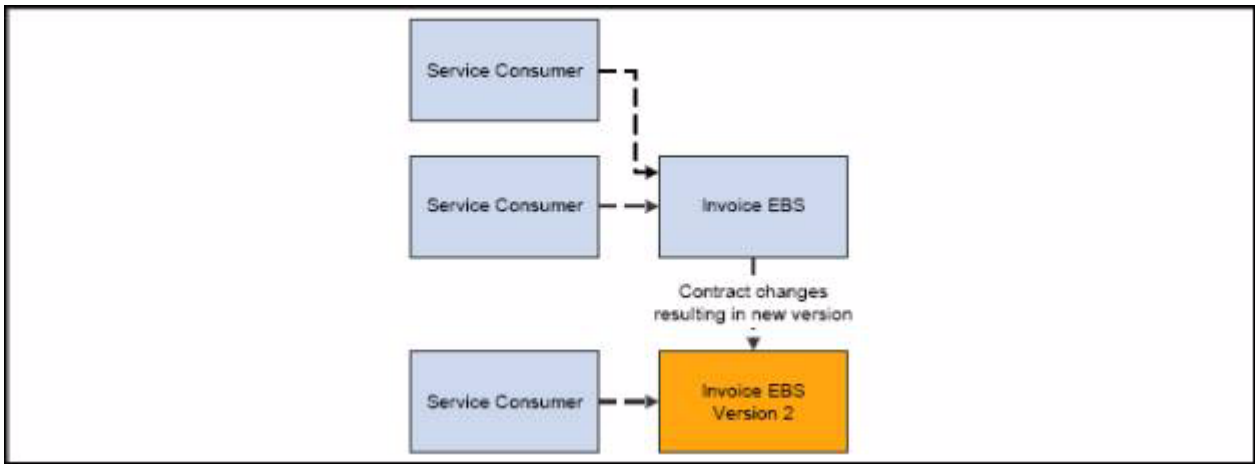
When the innermost layer namespaces are versioned, the next layer namespaces are also versioned if they use the new common constructs. However, the reverse is not true: versioning the second-layer namespaces does not require versioning the innermost layer namespaces. The innermost layer does not import the functional layer-specific namespaces.

7.2 Service Versioning

Enterprise Business Services (EBSs) evolve over multiple generations as you change the interface definition or the implementation. When a change affects the contract the consumer relies upon, it leads to the creation of a new service version.

Oracle AIA facilitates the co-location of multiple EBS implementations with identifiable versions. This allows consumers to use the service version they need. A new version does not force consumers to switch to the latest version immediately. Figure 7–1 is an illustrates versioning.

Figure 7–1 Illustration of Versioning



7.2.1 Naming Conventions

This section discusses naming conventions only with respect to versioning.

Similar to EBOs, each EBS in the library has its own release life cycle and each service its own version number. The first service version does not have a number. The default value is 1.0. Subsequent versions have numbers affixed to the name of the service. EBS version numbers are independent of participating application release numbers. New releases of participating applications do not necessarily result in new EBS versions.

7.3 Participating Applications Versioning

Participating applications in both requester and provider roles also evolve. New application versions can introduce enhancements to underlying connecting technologies or web services standards. Changing the connectivity/transport protocol, web service definitions, or payloads of Application Business Connector Services (ABCSs) specific to participating applications does not affect integrations.

8

Understanding Batch Processing

Oracle Application Integration Architecture (AIA) uses an extract, transform, and load (ETL) tool called Oracle Data Integrator (Oracle ODI) for high-performance movement and transformation of very large volumes of data between heterogeneous systems. Batch, real time, synchronous, and asynchronous modes are supported. Based on the relevant RDBMS engines and SQL, Oracle AIA can transform data on the target server at a set-based level. This chapter discusses use cases for batch processing and discusses bulk data integration patterns.

For more information about batch processing, see "Using Oracle Data Integrator for Bulk Processing" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

This chapter includes the following sections:

8.1 Batch Processing Use Cases

Oracle AIA uses batch processing technology for these 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 use the Cross Reference and Domain Value Map when the data transferred runs services from an Integration Scenario.

8.2 Bulk Data Integration Patterns

Oracle AIA supports the following bulk data integration patterns:

- Initial data loads
- High volume transactions with XREF table
- High volume transactions without XREF
- Intermittent high volume transactions

8.2.1 Initial Data Loads and High Volume Transactions with XREF Table

When you implement a new integration, you may need to synchronize some records. Using Oracle ODI, you can handle a large data-set and maintain the cross-reference table if necessary.

8.2.2 High Volume Transactions Without XREF

If synchronized records require no further DML operations, maintaining a cross-reference (XREF) record is unnecessary. For example, if retail stores transmit daily sales transactions to HQ, and if HQ requires no DML operations (as in a data-warehouse), then maintaining the cross-reference table is unnecessary.

8.2.3 Intermittent High Volume Transactions

Use the Intermittent High Volume Transactions pattern when both Oracle ODI and AIA EBS manage data integration. AIA EBS handles normal instance-based synchronization. Oracle ODI handles synchronization for batch transactions. Oracle ODI can also handle intermittent transactions containing very large objects, such as orders with many lines.

9

Understanding Security

This chapter describes Oracle Application Integration Architecture (AIA) support for all security-related functions.

This chapter includes the following sections:

- Section 9.1, "Introduction to Security"
- Section 9.2, "Oracle AIA Methodology"
- Section 9.3, "Introduction to Application Security Context"

For more information about security, see "Working with Security" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

9.1 Introduction to 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

Service-oriented architecture (SOA) separates interfaces and business logic. The security architect has several security deployment choices for SOA and web services.

For example, you can host a SOAP web service interface such as CreateSalesOrder as a proxy instead of hosting the real endpoint of the business logic implementation. The gateway handles communication to and from the web service and performs security functions on behalf of the service endpoint. The actual endpoint is virtualized. The client behaves as if it communicates with the service provider, when it actually communicates through the proxy.

Web service administration tools such as Oracle Web Services Manager (OWSM) ensure the validity and safety of XML messages exchanged between services. Integration architects and developers focus on integration logic, while security architects and administrators focus on security and management. Enforcing security policies through a centralized tool ensures corporate rule compliance and applies

policy changes centrally instead of in each web service. Administrators create security and management policies using a browser-based tool as needed during deployment. Security degrades performance, so administrators should use it only when needed. Typical web service security can have multiple policies attached. These policies can:

- Decrypt the incoming XML message
- Extract the user's credentials
- Authenticate this user
- Authorize this user and web service
- Log 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 the relevant 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.

9.1.1 Point-to-Point or End-to-End Security

A typical interaction in Oracle AIA involves discrete interactions between a service requester, client-specific Application Business Connector Service (ABCS), Enterprise Business Service (EBS), server-specific ABCS, and service provider. Choosing a security model plays a critical role in these interactions.

Oracle AIA supports point-to-point and end-to-end security models. You can choose either at implementation time for each transaction.

To choose a specific security model and implementation technique, consider:

- Is an entire transaction secured if the individual discrete transactions are secured?
- Is the *trusted model* adequate, or do the discrete interactions require certificates from a certificate authority?
- Can communication-level security methods such as SSL encryption secure individual discrete transactions within a trusted model?

You can adopt the industry-standard WS-Security model if all participating applications in the transactions provide inherent support.

9.1.2 Transport-Level Security

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 encrypted data exchange. In Oracle's Web Services Security model, SSL can provide point-to-point security, data integrity, and data confidentiality.

9.1.3 Message-Level Security

Oracle AIA strongly emphasizes on message-level security. XML encryption secures web service data exchanges. In comparison, SSL has limitations. For example, suppose a document visits several web services before reaching its eventual endpoint. XML

encryption can process the document while at rest or in transport. It also allows encryption of portions of a document instead of the whole.

9.2 Oracle AIA Methodology

AIA strongly recommends securing all services. Applying policies globally is preferable to constraining policy attachment only for individual services.

However, you must override globally attached client and service policies with directly attached local policies in these cases:

- Global authentication policies delivered:
  - Global Service Policy applied - oracle/wss_saml_or_username_token_service_policy
  - Global Service Client Policy applied- oracle/wss10_saml_token_client_policy
- Services in Pre-Built Integrations may be further hardened based on business needs.
- Applications invoking secured AIA Web Services must send credentials.
- The Global Service Client Policy handles inter-composite communication.
- AIA services invoking non-protected and protected application services must have the relevant client policy applied to the reference components present in the service.

Applying default global policies secures all Oracle AIA services and propagates security identity across all locally invoked services. Service developers can override default functionality as needed by disabling or replacing a policy for a locally invoked service.

9.3 Introduction to Application Security Context

The application security context model integrates participating applications with different security representations in a standard way by eliminating point-to-point security.

Participating applications are often developed at different times with different concepts and implementations of authentication and authorization. When applications are integrated, you must pass authentication and authorization information between them. The application security context standardizes authentication and authorization between applications so any application can be integrated with any other.

App Context includes any message processing information sent between the provider and requester applications. This includes but is not limited to authentication and authorization information. Oracle AIA supports authorization in app context. It also supports adding other context information.

Oracle recommends XACML Context Request as the best authorization information standard. XACML is an OASIS standard for managing access control policy. Released in 2003 and based on XML, XACML is designed to become a universal standard for describing who has access to which resources. XACML includes a policy language and a query language that results in a Permit, Deny, Intermediate (error in query), or Not Applicable response.

For more information about security, see "Working with Security" in the *Oracle Fusion Middleware Developer's Guide for Oracle Application Integration Architecture Foundation Pack*.

Index

A

B

C

D

E

F

I

M

N

O

P

R

S

T

U

V

W