Oracle® Application Integration Architecture - Foundation Pack 2.5: Concepts and Technologies Guide

Release 2.5

Part No. E15762-01

October 2009

ORACLE

Oracle Application Integration Architecture - Foundation Pack 2.5: Concepts and Technologies Guide

Part No. E15762-01

Contents

Preface v

Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide v

Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide vi

Oracle AIA PIP Implementation Guides vi

Additional Resources vii

Chapter 1: Getting Started with Oracle AIA Concepts and Technologies 9

Chapter 2: Understanding the Oracle AIA 11

Integrating Systems 11

Oracle AIA Capabilities 12

Solution Artifacts 13

Service-oriented architecture (SOA) and Enterprise Service Bus (ESB) 14

Support for Heterogeneous Integration Styles 15

The Canonical Data Model 15

Leveraging Oracle AIA 15

How Oracle AIA Differs from Point-to-Point Integrations 16

Use Case: Querying an Account Balance from a Billing Application 16

Chapter 3: Understanding EBOs and EBMS 17

EBOs 17

EBMs 20

EBM Architecture 21

EBM Headers 22

Chapter 4: Understanding EBSs 23

EBSs 23

EBS Operations 24

Verbs 24

EBS Types 24

Entity Services 25

Process Services 26

EBS Architecture 28

Enterprise Business Flow (EBF) Processes 31

EBS Implementation 33
EBS Message Exchange Patterns 35
Synchronous Request-Response Patterns in EBSs 36
Asynchronous Fire-and-Forget Patterns in EBSs 36
Asynchronous Request–Delayed Response Patterns in EBSs 38
Chapter 5: Understanding ABC Services 41
ABC Services 41
ABC Service Architecture 43
ABC Service Characteristics 45
Architectural Considerations 45
Participating Application's Service Granularity 45
Support for EBMs 46
Application Interfaces 46
Support for Logging and Monitoring 46
Support for Insulating the Service Provider 47
Support for Security 47
Validations 47
Support for Internationalization and Localization 48
Message Consolidation and Decomposition 48
Support for Multiple Application Instances 48
Implementing ABC Services 48
Requestor-Side ABC Services 49
Provider-Side ABC Services 52
Reviewing Implementation Technologies for ABC Services 55
ESB 55
BPEL 55
Extending or Customizing ABC Service Processing 56
Processing Multiple Instances 56
Participating Applications Invoking ABC Services 57
ABC Service Transformations 57
Transformation: Implementation Approach 57
Static Data Cross-Referencing 58
Dynamic Data Cross-Referencing 58
Structural Transformation 58
Chapter 6: Understanding Interaction Patterns 61

Patterns for Exchanging Messages 61
Request/Response 62
Synchronous Response 62
Fire-and-Forget 62
Message Routing 63
Message Splitting and Routing 64
Data Enrichment 64
Data Aggregation 65
Asynchronous Request – Delayed Response Pattern 66
Publish-and-Subscribe 66
Chapter 7: Understanding Extensibility 67
Extensibility 67
Schema Extensions 68
Customer Extensions 68
Industry-Specific Extensions 68
Schema in the Use Case 69
Transformation Extensions 69
Extensions in the Use Case 69
Transport/Flow Extensions 69
Process Extensions 70
Routing Extensions 70
Chapter 8: Understanding Versioning 71
Schema Versioning 71
Major and Minor Versions 71
Namespaces 72
Service Versioning 73
Naming Conventions 74
Participating Applications Versioning 74
Chapter 9: Understanding Batch Processing 75
Batch Processing 75
Chapter 10: Understanding Infrastructure Services 77
Logging and Error Handling 77
Composite Application Validation System (CAVS) 78
Business Service Registry (BSR) 79
Deployment 79

Internationalization and Localization Support 79
Chapter 11: Understanding Security 81
Security 81
Point-to-Point or End-to-End Security 82
Transport-Level Security 82
Message-Level Security 82
Securing ABC Services 83
Implementation Techniques for Enterprise Service Bus Security 83
Index 85

Preface

The *Oracle Application Integration Architecture - Foundation Pack: Concepts and Technologies Guide* provides definitions of fundamental Oracle Application Integration Architecture (AIA) Foundation Pack concepts and discusses:

- Oracle AIA.
- Enterprise business objects (EBOs) and enterprise business messages (EBMs).
- Enterprise business services (EBSs).
- Application business connector (ABC) services.
- Interaction patterns.
- Extensibility.
- Versioning.
- Business processes.
- Batch processing.
- Infrastructure services.
- Security.

The *Oracle Application Integration Architecture – Foundation Pack: Concepts and Technologies Guide* is a companion volume to the following guides and resources discussed in this preface:

- The *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide*
- The *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide*
- Process integration pack (PIP) implementation guides
- Additional resources

Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide

See the *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide* for information about how to:

- Work with the Composite Application Validation System (CAVS).
- Work with the Business Service Repository (BSR).
- Set up and use error handling and logging.

- Work with the diagnostics framework.
- Work with Oracle AIA developer tools.

Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide

See the *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide* for information about how to:

- Create an integration scenario.
- Define business service patterns.
- Design and develop EBSs.
- Design and develop enterprise business flows (EBFs).
- Design and construct ABC services.
- Work with message transformation, enrichment, and configuration.
- Develop custom XPath functions.
- Design and construct JMS Adapter services.
- Work with EBM headers.
- Work with message routing.
- Work with transactions.
- Develop Oracle AIA services to work with the CAVS.
- Configure Oracle AIA processes to be eligible for error handling and logging.
- Extend EBOs.
- Work with the Event Aggregation programming model.
- Work with the Publish-and-Subscribe programming model.

In addition, this book provides Oracle AIA naming standards.

Oracle AIA PIP Implementation Guides

A PIP is a pre-built set of integrated orchestration flows, application integration logic, and extensible EBOs and EBSs required to manage the state and execution of a defined set of activities or tasks between specific Oracle applications associated with a given process.

A PIP provides everything you need to deploy a selected integrated business process area. The PIP product offering is suited to those customers seeking to rapidly implement a discreet business process.

Implementation guides are available for each PIP.

Additional Resources

The following resources are also available:

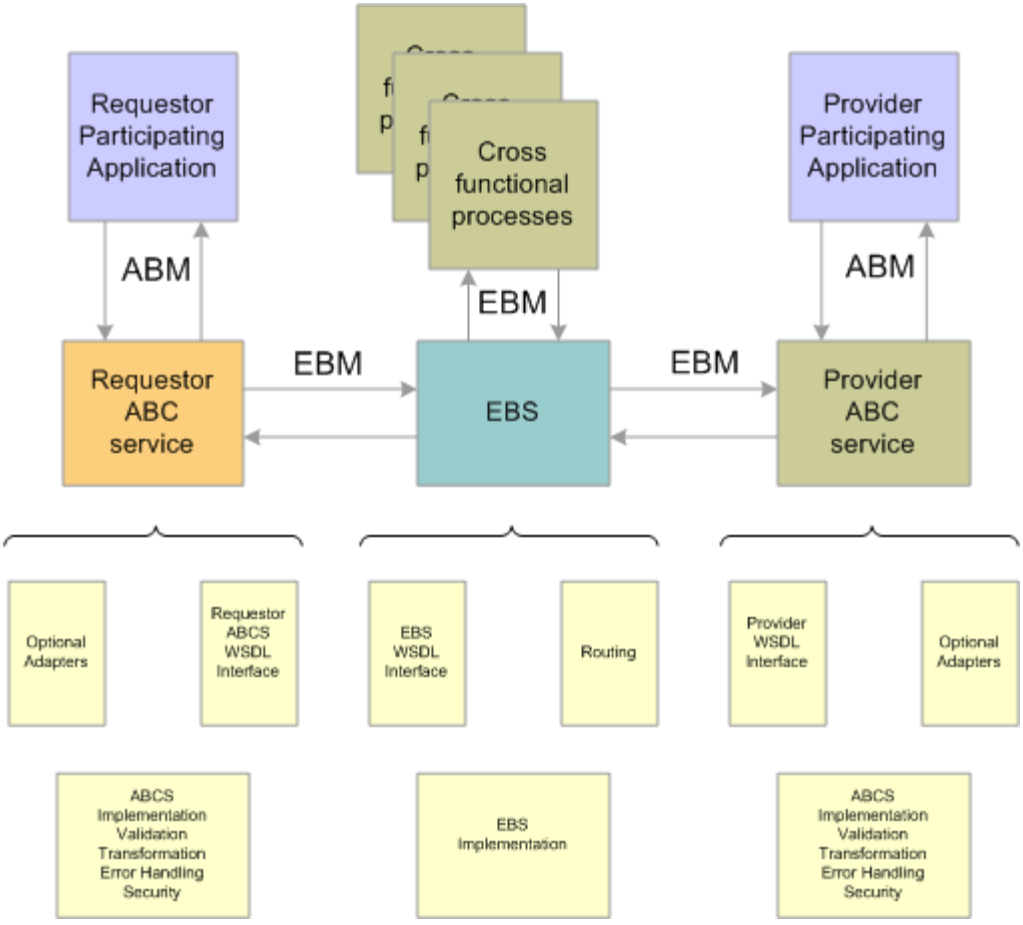
Resource	Location
Oracle Application Integration Architecture - Installation and Upgrade Guide	Metalink: https://metalink.oracle.com/
Known issues, workarounds, and most current list of patches	Metalink: https://metalink.oracle.com/
Release notes	Oracle Technology Network: http://www.oracle.com/technology/
Documentation updates	Metalink: https://metalink.oracle.com/

Chapter 1: Getting Started with Oracle AIA Concepts and Technologies

This chapter discusses the Oracle Application Integration Architecture (AIA).

Oracle AIA enables the development of compelling industry-specific composite business processes leveraging existing software assets that are available in your IT infrastructure. Prebuilt process integration packs (PIPs) from Oracle use the Oracle AIA to deliver composite industry processes for specific industries, using software assets from Oracle's portfolio of applications. Most of these solutions encompass orchestrated process flows, as well as prebuilt data integration scenarios that are meant to seamlessly connect the systems. To develop these solutions in a consistent manner, we have defined the Oracle AIA.

This diagram illustrates the components of Oracle AIA:



Oracle AIA Components

In this diagram, the following acronyms have been used:

- ABM = application business message
- EBM = enterprise business message

- EBS = enterprise business service
- EBF = enterprise business flow
- ABCS = application business connector service

Chapter 2: Understanding the Oracle AIA

This chapter introduces you to the Oracle Applications Integration Architecture (AIA) and talks about the various ways that systems that were not designed to work together can be integrated. It highlights the key capabilities of Oracle AIA, as well as the various artifacts that are delivered with it. The use case introduced at the end of this chapter is used throughout the book to explain the various constructs.

This chapter discusses:

- Integrating systems
- Oracle AIA capabilities
- Solution artifacts
- Service-oriented architecture (SOA) and Enterprise Service Bus (ESB)
- Support for heterogeneous integration styles
- Use case: Querying an Account Balance from a billing application

Integrating Systems

Oracle AIA provides solutions for connecting systems that were not designed to work together. The integrations can be categorized as:

- User-interface integration
- Data integration
- Functional integration
- Process integration

User-Interface Integration

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

Data Integration

Data integration connects the applications at the data level and makes the same data available to more than one application. This type of integration relies on the database technologies and is ideal when there is minimal amount of business logic to be reused and bulk amount of data transactions are involved across applications. This type of integration is suitable for batch data uploads or bulk data sync requirements.

Functional Integration

Functional integration connects the applications at the business logic layer. This type of integration is used when there is a need to reuse functionality such as business logic or a process. For example, submission of a Sales Order using a Front-Office application such as Siebel CRM will result in an invocation of service exposed by Order Fulfillment system to submit the order for fulfillment.

Functional integration can be accomplished either by exposing object interfaces that can be consumed by other systems, by using message oriented middleware systems to send the messages to the destinations, or by exposing web service interfaces that could be consumed by the clients.

Process Integration

Process integration involves integrating a set of activities that can span across applications or businesses to create a process flow. The process flow typically runs independently and it describes the individual steps that make up the business function. Each of the steps in a process flow is carried out by executing the appropriate application function. For example, the Order to Cash orchestration process has several activities such as checking credit, creating customer in billing system, and so on. For each of the activities, an appropriate business service will be invoked which is responsible for discerning the relevant application function that needs to be executed.

Process integration allows a clear separation of the process definition from the execution of the process and the implementations of individual functions in the applications. This allows application functions to be used in multiple process flows. Similarly, a single process flow can contain a step that can be fulfilled by more than one application having the same business capability.

Even though Oracle AIA addresses all types of integrations, the primary focus of this guide is to describe how functional and data integration between disparate systems can be accomplished.

Oracle AIA Capabilities

Oracle AIA capabilities include:

- Defines integration architecture by adopting a SOA.
- Leverages various existing assets found in Oracle's portfolio, as well as those of customers.
- Enables extensive access to web services provided by various applications.
- Provides a general infrastructure for consistent integrations that are also extensible and able to respond to requests from industry strategy.
- Facilitates the use of services in orchestrated process flows.
- Allows a customer to extend various artifacts of the delivered solution.
- Accommodates a loose coupling between systems. This includes the ability to:
    - Define loosely bound services that are invoked through communication protocols that stress location transparency and interoperability.
    - Define services that have implementation-independent interfaces.
    - Replace one service implementation with another with no impact to the client.
- Incorporates synchronous and asynchronous communication.
- Accommodates the following message interaction styles:

  - Synchronous request-response
  - Asynchronous request with delayed-response
  - Asynchronous fire-and-forget
  - Publish/subscribe
- Adopts an applications independent canonical data model to accomplish the decoupling of data format.
- Provides end-to-end security.
- Supports heterogeneous programming languages and platforms.
- Supports incremental adoption and implementation.
- Allows customers to upgrade participating applications and integration components according to timelines they are comfortable with.
- Handles high transaction rates and volumes that are normally associated with mission-critical applications.
- Preserves extensions and modifications to integrations during upgrades of source or target applications.

Solution Artifacts

The solutions that Oracle AIA delivers are composed of the following artifacts.

- Enterprise business objects (EBO)

  These are EBO schemas based on the application-independent canonical object model. These, along with enterprise business services (EBSs), act as the hub of our architecture.

  For more information, see EBOs.

- Enterprise business services (EBSs)

  These are application and implementation independent service definitions and business level interfaces. These service definitions are implemented by all applications that participate in integration.

  For more information, see EBSs.

- Application business connector services (ABC services)

  These services are responsible for exposing business functions available in Oracle or non-Oracle applications as services that are able to participate in the Oracle AIA ecosystem.

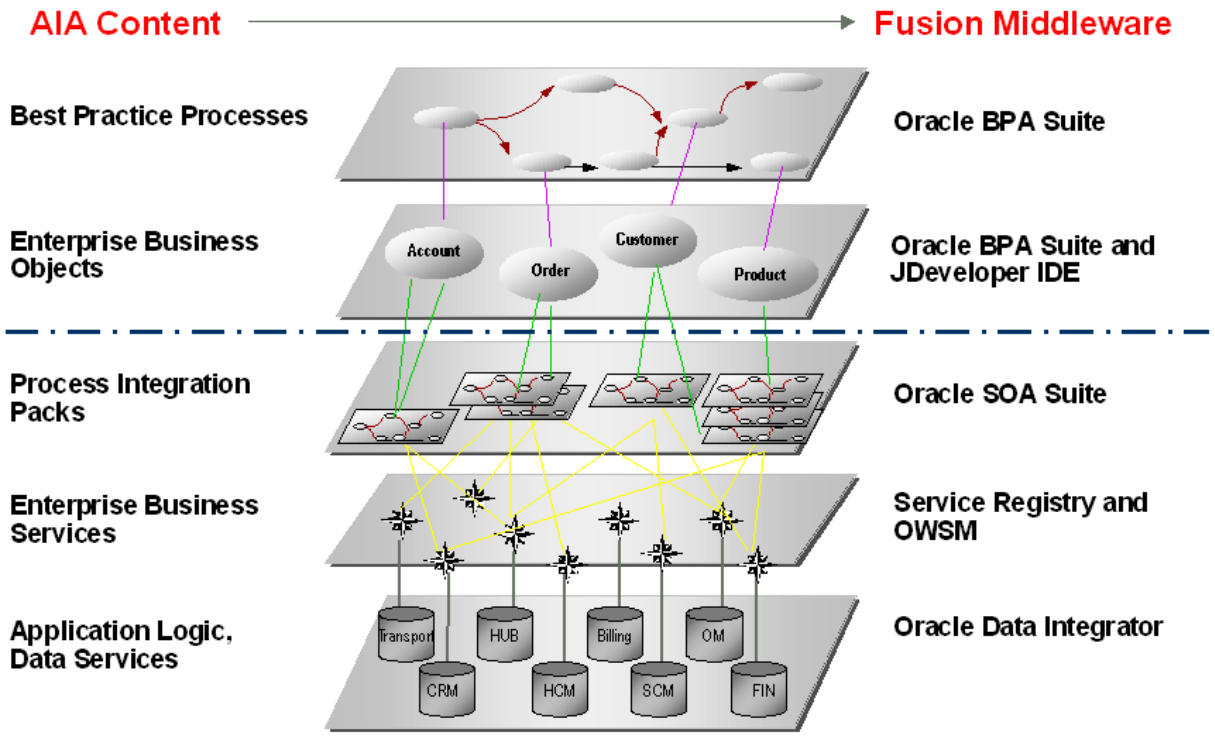
  For more information, see ABC Services.

- Orchestrated process flows, or enterprise business flows (EBFs)

These process flows are responsible for performing the end-to-end processes that involve activities spanning application and business boundaries.

For more information, see EBF Processes.

This diagram shows an overview of some artifacts that are delivered by Oracle AIA.



Oracle AIA Solution Artifacts

Service-oriented architecture (SOA) and Enterprise Service Bus (ESB)

Oracle AIA relies upon a SOA for integrations. In integrations where it makes sense, Oracle AIA also relies on ESB. SOA is an approach for defining an architecture based on the concept of a service. Service-oriented design promotes interoperability. To embrace SOA, the applications, as well as the underlying infrastructure, must support the SOA concepts.

SOA applications are integrated at the service specification point, not at the application interface. This allows application complexities to be separated from the services interface, and diminishes the impacts of back-end interface and application changes.

SOA-enabling applications involve the creation of service interfaces to existing or new business capabilities available in the applications.

SOAs consist of services that are defined by explicit, implementation-independent interfaces. They are loosely bound and invoked through communication protocols that stress location transparency and interoperability. The interaction model is document-oriented and leverages technology-neutral protocols. SOA helps in exposing the software assets found in the Oracle portfolio as high-level services.

Service-oriented integration leverages messages to communicate between consumers and providers and uses XML schemas and transports such as SOAP to transport messages.

An ESB is the underlying infrastructure for delivering SOA. Oracle AIA uses Oracle Enterprise Service Bus as the foundation for services using SOA. It enables interaction through services that encapsulate business functions and supports service routing and the substitution and translation of transport protocols.

Support for Heterogeneous Integration Styles

Oracle AIA provides complete support for the following integration styles:

- Synchronous request-response
- Fire-and-forget
- Asynchronous request with delayed-response
- Batch processing

These integration styles are supported by the ability of the ESB to support SOA, a message-driven architecture using its infrastructure. The SOA aids the interaction of applications through well-published and explicit implementation-independent interfaces. The message-driven architecture facilitates the transmission of messages through the ESB to the receiving applications. Oracle AIA, as well as the technologies used by the participating applications, helps applications to produce and consume messages without requiring that the applications be aware of one another.

For more information, see Chapter 6: Understanding Interaction Patterns.

The Canonical Data Model

Oracle AIA introduces a set of generic data structures called EBOs. An EBO represents a common object definition for business concepts such as Account, Sales Order, Item and so on. The business integration processes work only on messages that are either a complete EBO or a subset of an EBO. This approach enables the cross-industry application processes to be independent of participating applications. The EBO is defined by using inputs from various applications as well as from industry standards and eliminates the need to map data from different applications directly to each other. EBOs contain components that satisfy the requirements of business objects from the target application data models.

For more information, see EBOs.

Leveraging Oracle AIA

In application integration, the core functionalities of best-of-breed applications are leveraged to accomplish tasks by tying them into a business process. The functionality is exposed as a function. Events in applications trigger information interchange as a straight-through process consisting of multiple tasks spanning multiple applications. This can be real-time or in batch mode.

How Oracle AIA Differs from Point-to-Point Integrations

In traditional integrations where one system is directly connected to the other, the system making the request (requestor application) knows the internals of the system that is providing the service (provider application). It needs to know the shape of the data expected by the provider application. The requestor application also needs to know how to connect with the provider application. Exposing these implementation details to the requestor application constrains the provider application from making any significant changes to its implementation.

In the integrations that are built using Oracle AIA, the application making the request (requestor application) is not tightly coupled with the application providing the service (provider application). What that means is, the requestor application is completely unaware of who the ultimate service provider is. This greatly enhances the ability for making application changes to the provider application without introducing any significant impacts to the requestor application.

Use Case: Querying an Account Balance from a Billing Application

Throughout this guide the Get Account Balance use case will be used to explain the various constructs found in Oracle AIA. A walk-through of this use case will give you an overview of the architecture, as well as an introduction to the components that are involved.

The Customer Service Representative (CSR) is working with a client on the phone. The CSR opens the client's record in the Siebel CRM application but needs to have current account balance information to respond to the client's question. When the CSR clicks the Get Account Balance button, the Siebel CRM system retrieves the customer's account details (including the account balance information) from Oracle's Billing and Revenue Management application (BRM) and displays the information on the screen. Using this information, the CSR is able to answer the client's question.

The use case will highlight the process used to enable the Siebel CRM application to retrieve an account balance from the billing application:

- Define an application independent service called *Query Customer Party*. The service will have input and output parameters.
- Have the account querying capability available in Oracle's BRM exposed as a web service as per Query Customer Party definition.
- Modify the Siebel application to invoke the Query Customer Party service when the CSR clicks the Get Account Balance button. Retrieve the response returned by the service and render it on the screen.

Chapter 3: 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 the architecture and usages as well as how the context-specific views of the EBO can be created.

This chapter discusses:

- EBOs
- EBMs
- EBM architecture
- EBM headers

EBOs

The EBO is the standard business data object definition and reusable data components. The library of all EBOs makes up a data model. The EBO represents a layer of abstraction on top of the logical data model and is targeted for use by developers, business users, and system integrators. In the integrations developed using Oracle Application Integration Architecture (AIA), the EBO data model serves as a common data abstraction across systems. It supports the loose coupling of systems in Oracle AIA and eliminates the need for one-to-one mappings of the disparate data schemas between each set of systems.

The adoption of the EBO facilitates the mapping of each application data schema only once to the EBO data model. This significantly minimizes the manual coding for data transformation and validation since it eliminates the need to map data directly from one application to another.

EBOs have the following characteristics:

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

  Instead they provide the structure for exchanging data. XML provides the vocabulary for expressing business data. The XML schema is an XSD file that contains the application-independent data structure to describe the common object.

- Each EBO is represented in an XML schema (XSD) file format.

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

Components that are applicable to all EBOs are defined in a common components schema module. Business components that can be used across various context-specific definitions for a single EBO are defined within the EBO schema module.

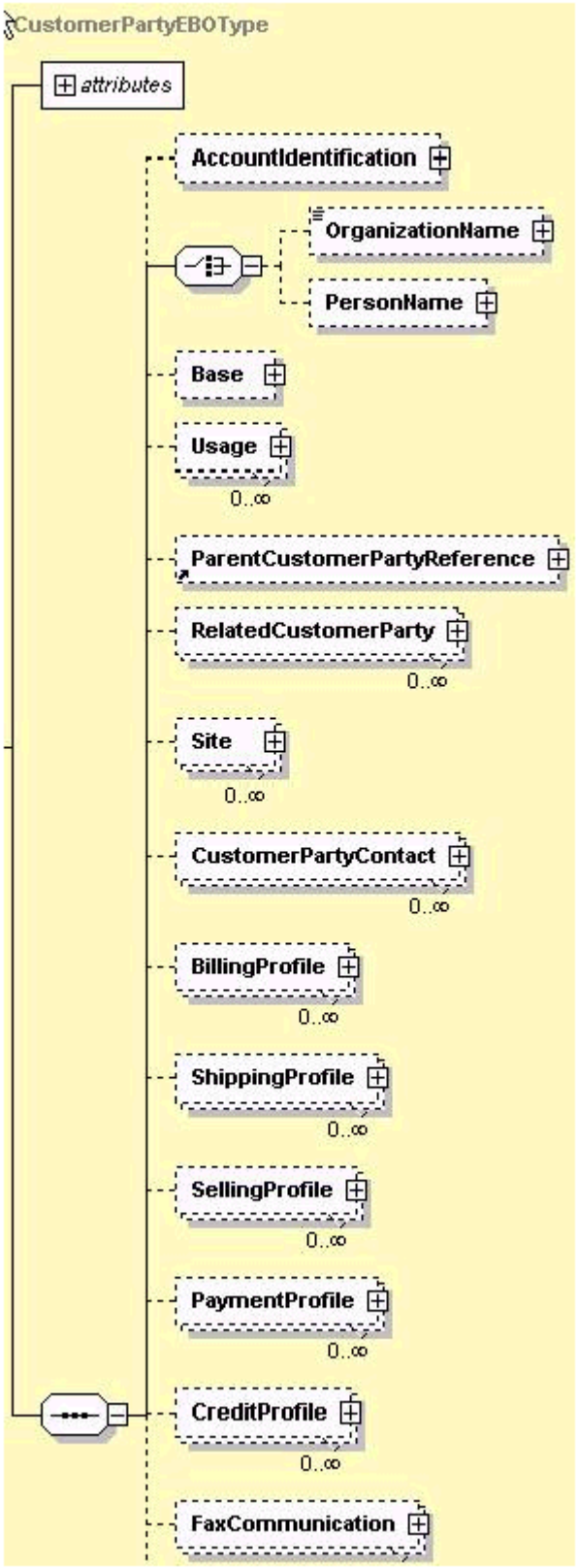
Wherever possible and practical, the EBO leverages widely adopted international standards for modeling and naming, such as the United Nations Centre for Trade Facilitation and Electronic Business (UN/CEFACT) Core Components Technical Specification (CCTS), UN/CEFACT XML Naming and Design Rules (NDR),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

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

EBOs and the Use Case

For the use case described in Use Case: Querying an Account Balance from a Billing Application, the *Customer Party* EBO is used to facilitate the integration between Siebel CRM (Requestor Application) and Oracle BRM (Provider Application). If the Customer Party EBO is not available, then the EBO must be created by first defining the content and then adhering to the guidelines for creating the EBO.

The XML schema snippet for Customer Party EBO shows the semantic clarity that is exhibited by each of the elements:



Structure of Customer Party EBO

EBMs

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

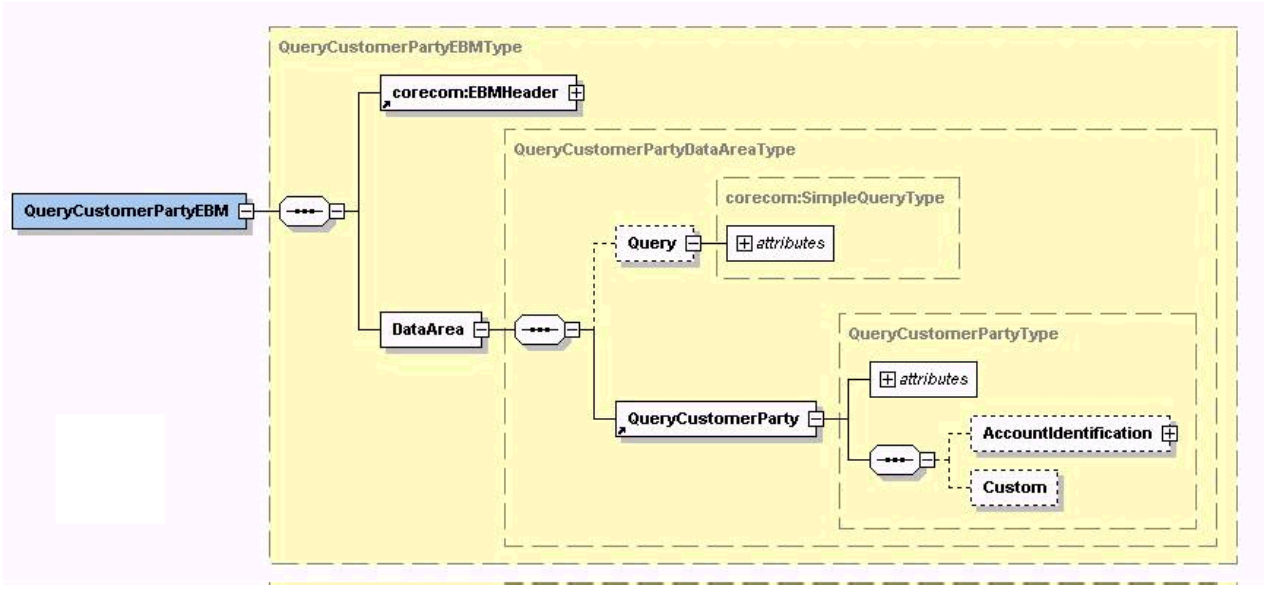
The context-specific EBM definitions are created by assembling a set of common components and EBO-specific business components. In some scenarios, the business components can be obtained from more than one EBO. These context-specific EBO definitions are then used in the appropriate context-specific EBMs.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

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

For more information, see Chapter 7: Understanding Extensibility.

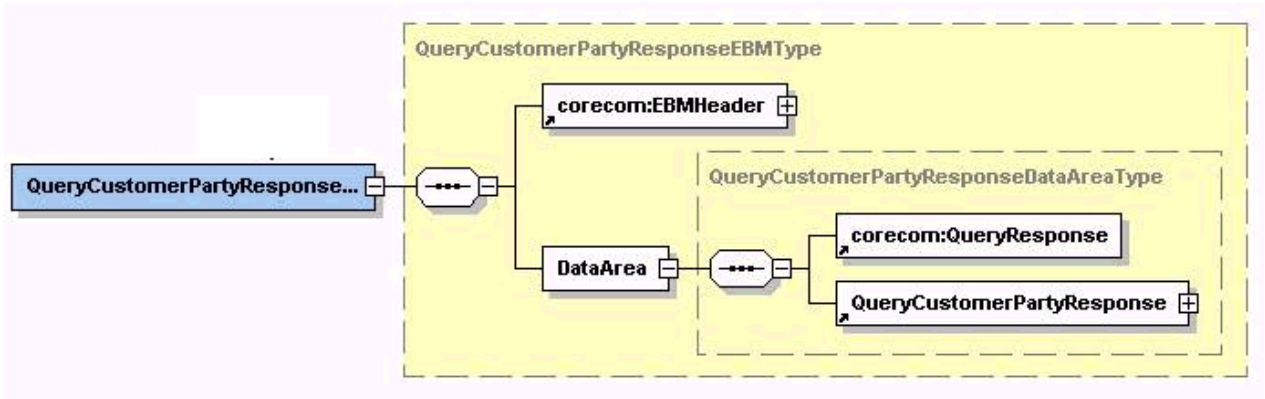
Using EBMs in the Use Case

Now that you have an understanding of what an EBM is, you can see what needs to be done with respect to the use case. You need to define the shape of the message to list out what needs to be sent by the requestor application for invoking the service *Query Customer Party*. Both the request and response messages will have content from Customer Party EBO. The request message will be *QueryCustomerPartyEBM*.



Query Customer Party EBM

You also need to define the shape of the message to list out what needs to be sent by provider application in response to the request, and the response message will be *QueryCustomerPartyResponseEBM*.



QueryCustomerPartyResponseEBM

EBM Architecture

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

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

When using EBMs, consider the following:

- Application interdependencies.
- Any application invoking the enterprise business services will have to generate the EBM in order to pass the EBM as a payload to the enterprise business service (EBS).
- The action in the EBM identifies the action that the sender or the requester application wants the receiver/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 or not. The Query action may carry information about the document header section of the original EBM that resulted in the execution of this action.

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

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

- An EBM can be defined to carry multiple instances. Only the actions that support bulk processing will use EBMs that support multiple instances.
- The information 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.

EBM Headers

The EBM header is an integral part of every EBM. The EBM header can be considered as a wrapper or an envelope around transactional data messages. It comprises representations of functional data such as Document Identification, Involved Parties (Sender, Provider, intermediary services_, Security, & Transaction Rules (Transaction State & Exceptions).

The EBM header provides the ability to:

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

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

For more information, see *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide,* "Working with EBM Headers."

Chapter 4: Understanding EBSs

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

This chapter discusses:

- EBSs
- EBS operations
- Verbs
- EBS types
- EBS architecture
- Enterprise business flow (EBF) processes
- EBS implementation
- EBS message exchange patterns

EBSs

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

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

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

For more information, see EBM Architecture.

EBS components do not presuppose a specific backend implementation of that service. They simply provide an integration layer for the customer's choice of a backend. For example, the *Query Customer Party* Service that is discussed in the use case does not mandate that customers use the Oracle BRM-based implementation of the Query Customer Party service that might be delivered out of the box. Customers might want to use either a third-party billing or homegrown billing implementation of Query Customer Party. Regardless of the choice, customers can still achieve the seamless interaction experience in the pre-built integrations that are delivered by Oracle AIA. Any backend implementations that can support the interface standards defined by an EBS can be automatically considered as service providers.

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

For more information, see Verbs.

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

Verbs

Every operation has a “Verb” to identify the action to be performed by the operation. The concept of a “Verb” was originally introduced by OAGIS in their BOD definitions and has been adopted with some modifications in the EBO definition.

Strictly speaking, the significance of a “Verb” to identify the action to be performed by an operation is not applicable in a service-oriented Web Services world, as the operation definition of a Web Service assumes this responsibility. However, not all integrations are using web services, and there are still message-oriented integration scenarios that may require the processing action to be identified within the message.

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

EBS Types

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

Entity Services

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

For the use case, the EBS *Customer Party* will be used. This EBS will contain all standard business activities that can be acted upon a customer.

The standard activities in an entity-based EBS are:

- Create

  To create an object instance

- Update

  To update an object instance with only the changes that occurred

- Delete

  To delete an object instance

- Query

  To retrieve details about an object

- Sync

  To send a current snap shot

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

For the use case, the EBS Customer Party will be used. This EBS contains all standard activities that can be acted upon a customer. To retrieve details about the customer (including account balance), Query Customer Party is used. In web services parlance, it will be called as a *service operation*. The Query Customer Party service operation will use Query Customer Party EBM as the input and Query Customer Party Response EBM as the output.

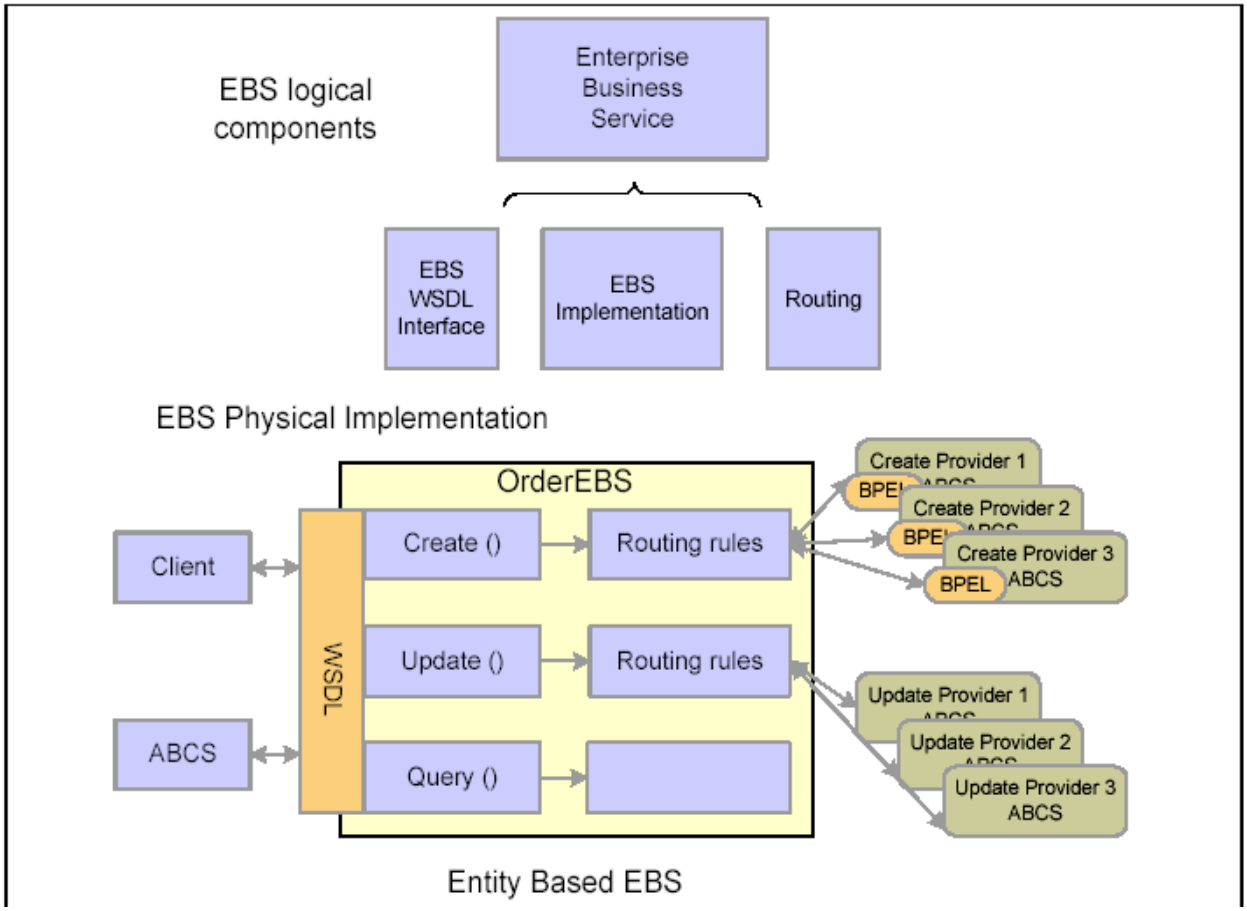
Entity services expose operations that act on a specific EBO.

Entity services have the following characteristics:

- Each of the business object entities has a corresponding enterprise business service; all of the actions that can be performed on this object are exposed as service operations and are part of this EBS.
- The entity operations consist of standard CRUD operations that have the same look and feel across services as well as any specialized operations specific to the EBO.
- Entity services are implemented as enterprise service bus (ESB) routing services.
- Entity services receive and return messages in the form of EBMs.
- Entity services leverage the context-based routing rules to delegate the request to the appropriate application-specific application business connector (ABC) service containing the actual

implementation.

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



Entity Services

Process Services

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

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

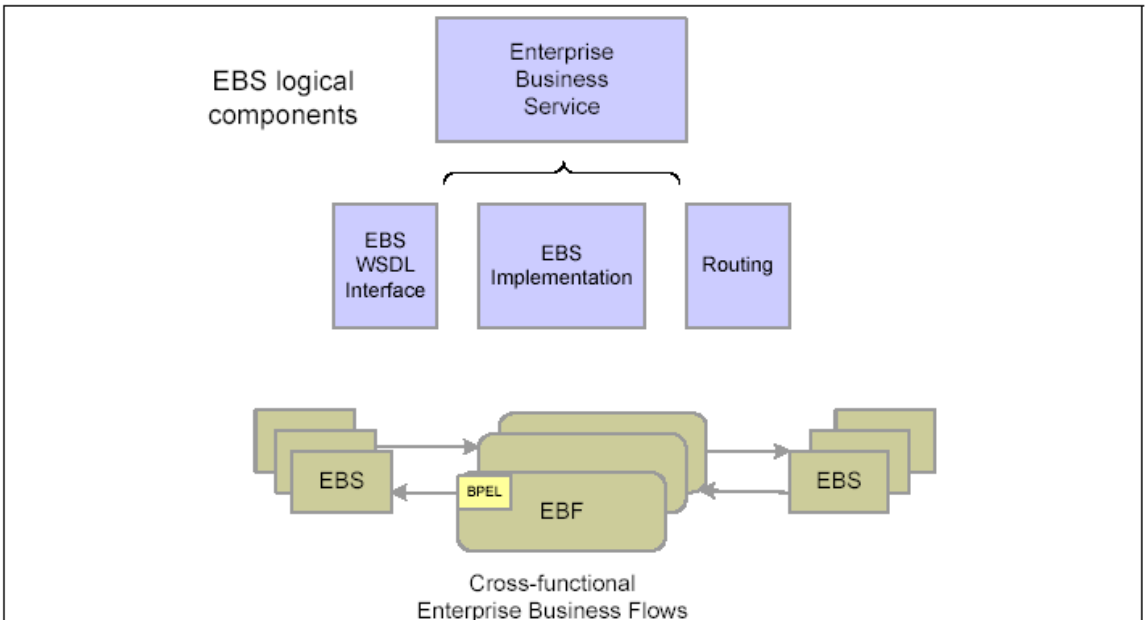
Process services expose operations related to an enterprise business process.

Process services have the following characteristics:

- Each EBF has a corresponding EBS; all of the actions that can be performed on this flow are exposed as service operations and are part of this EBS.

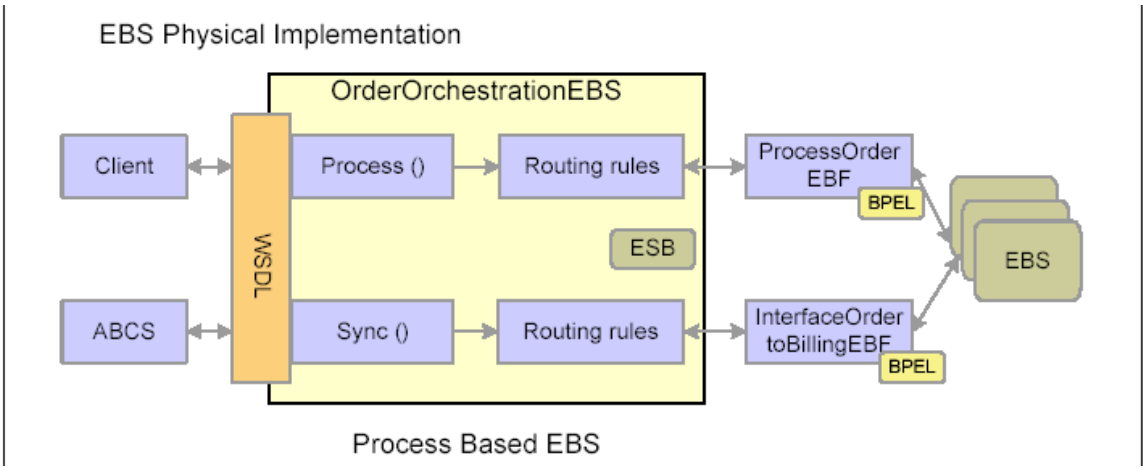
- Process services are implemented as ESB routing services.
- Process services receive and return messages in the form of EBMs.
- Operations exposed on the EBS will initiate cross-functional enterprise business flows that coordinate complex long-lived flows that span multiple services. These flows can only interact with EBS services. This way both the Process EBS and the related business flows are completely application agnostic.
- Unlike Entity Services, a Process Service may act on more than one EBO.

This diagram illustrates the EBS logical components for process services.



EBS Logical Components for Process Services

This diagram illustrates the EBS physical implementation of process services.



EBS Physical Implementation of Process Services

EBS Architecture

The EBS architecture enables:

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

Reuse of Available Assets

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

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

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

The virtualization layer enables Oracle AIA to achieve one or many of the following:

- The physical implementation of the target service needs to be abstracted and its location needs to be hidden so that the target service (Service Provider) can be eventually replaced by some other service (endpoint virtualization).
- If the shapes of data and operations are different, data transformations and operation-to-operation mappings are needed. This is common when old systems in existing infrastructures need to be replaced with no interruption or change to the consumers.
- The virtual service may be composed of more than one physical service, as in aggregation of services or rule-based routing. For example, a request from CA goes to the CA-Warehouse.
- You may not want to expose all operations of the actual service to the external world (partial service).
- Target service might be supporting a different transport protocol than supported by the service consumer. For example, a header transformation needs to happen, from JMS to SOAP.
- If complex, content-based validations against XML are required. For example, the order price must reflect the sum of prices of each order line. In this case, Schematron should be used inside the mediator.
- If the service consumer expects an asynchronous message exchange pattern and the service provider only allows for synchronous calls. In such cases, the client application invokes a virtual or proxy service (in Oracle AIA, it is an EBS) representing the target service(s).

For more information, see Chapter 5: Understanding ABC Services.

Substituting One Service Provider with Another

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

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

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

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

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

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

Content-Based Selection of the Service Provider

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

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

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

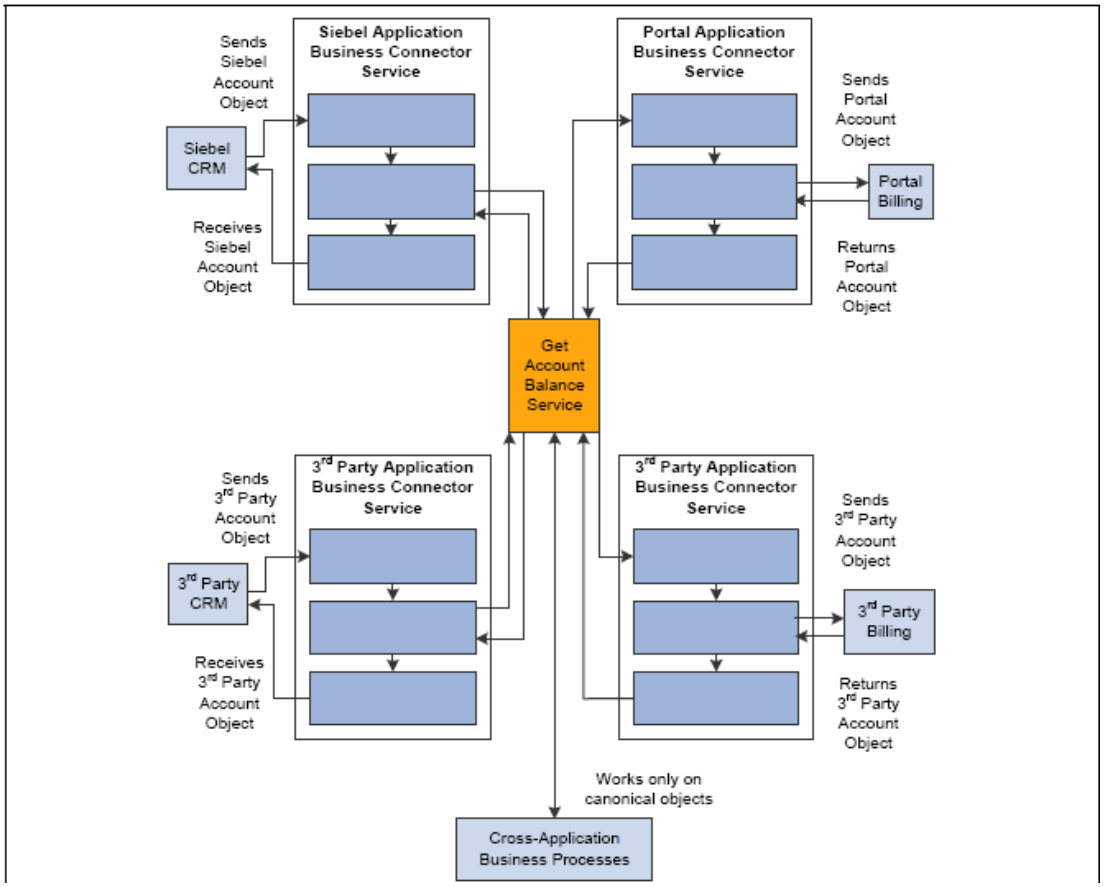
However, at design time, the requestor will explicitly specify the name of the enterprise business service operation, which in turn is responsible for identifying the actual service provider. The data formats are also specified at design time and regardless of the actual service providers, the data formats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

EBS Purpose

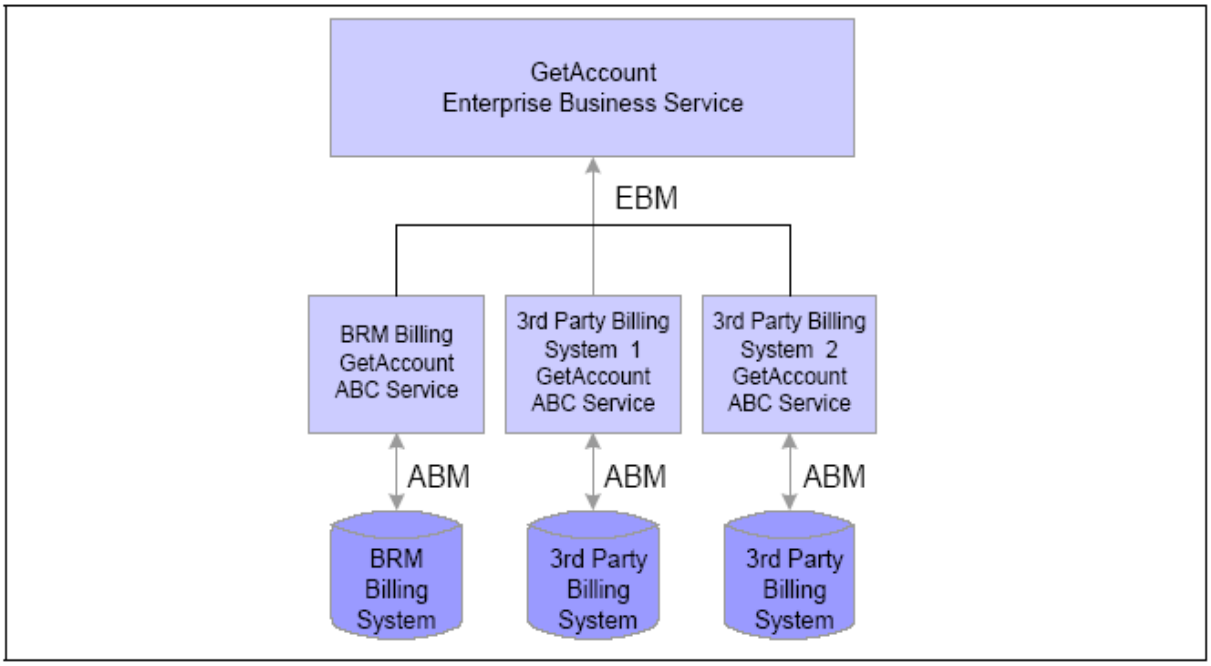
The purpose of the EBS can be summarized as follows:

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

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



EBS Enables Loose Coupling of Requestors with Service Providers



Example of GetAccount EBS

Enterprise Business Flow (EBF) Processes

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

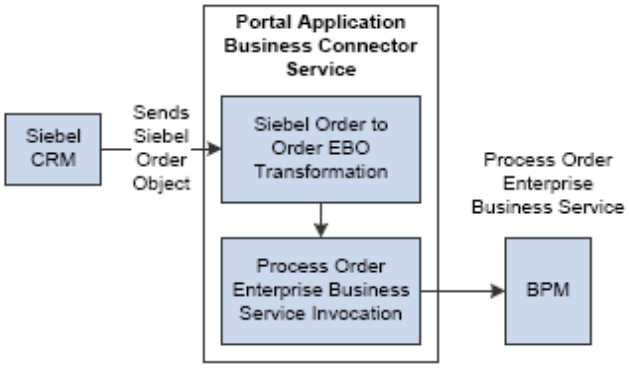
Business processes are defined independently of the underlying applications, simplifying the process of integrating applications from multiple vendors. Business processes will always use the services of the EBS.

For more information, see EBSs.

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

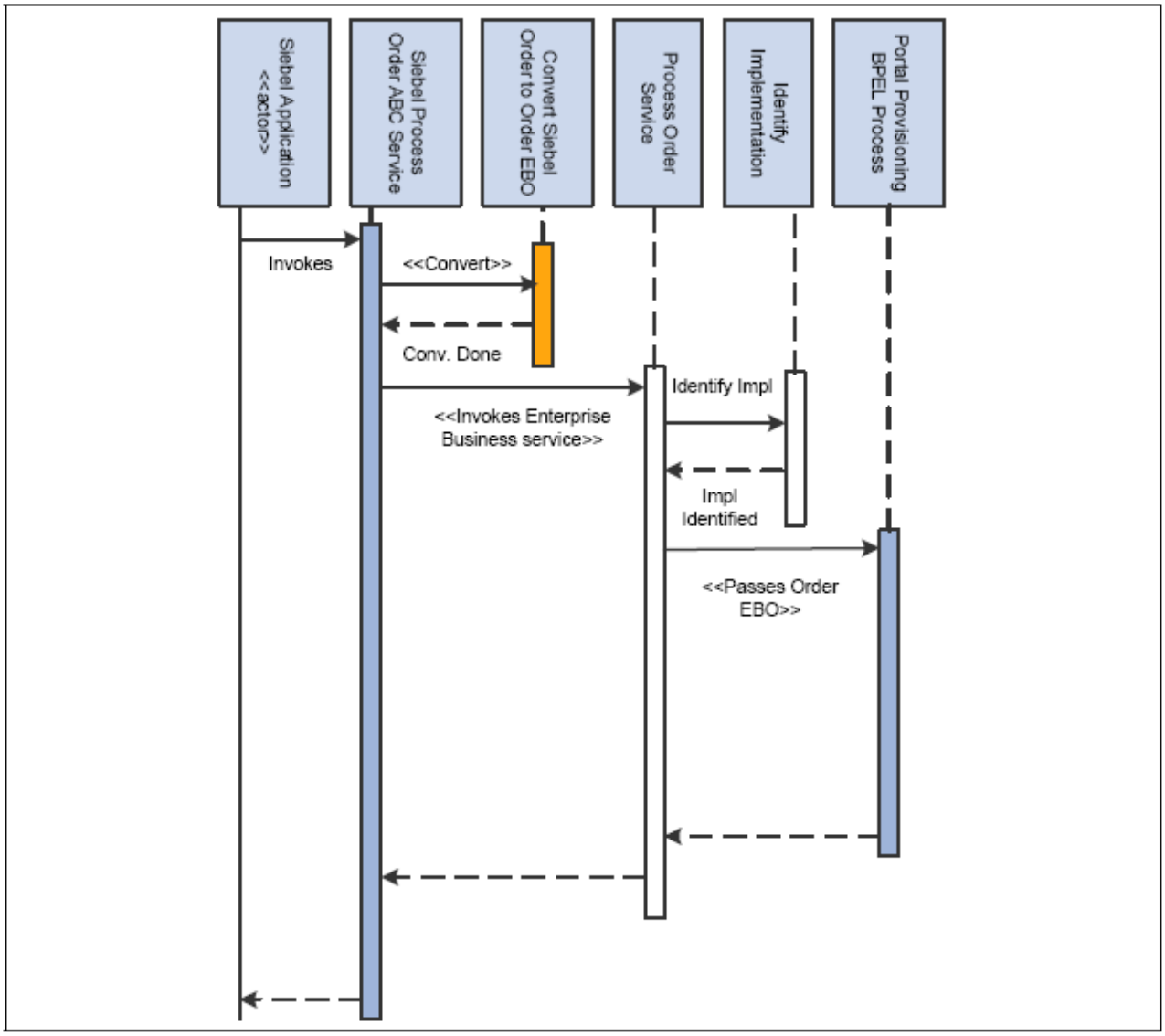
Within a cross-application business process, the EBO is used as the structure that holds the in-memory representation of the message that is sent back and forth between applications.

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



Participating Application Request Initiates a Business Process

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



Invocation of Process Order

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

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

EBS Implementation

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

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

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

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

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

```
            </svcdoc:Operation> </documentation>
        <input message="sordsvc:CreateSalesOrderMsg"/>
    </operation>
```

EBS Responsibilities

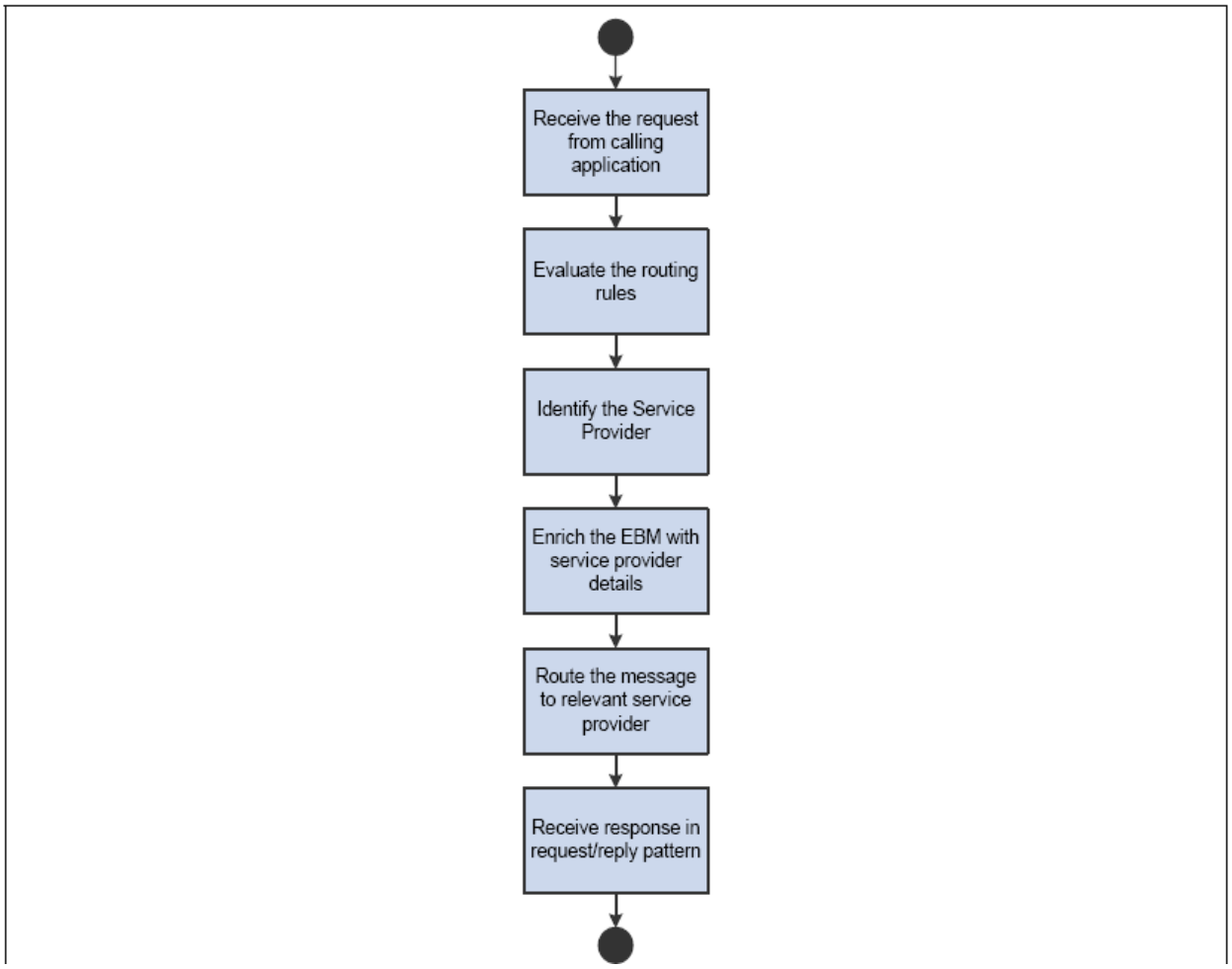
Here is an overview of the responsibilities of the EBS:

1. The EBS passes the enterprise business message (EBM) to the routing rules.
2. The routing rules evaluator applies the relevant rules to the EBM to decipher the ABC service it should invoke as well as the end-point details.
3. In scenarios where there are multiple instances for a single application, the EBS service enriches the EBM header section of the document with details about the service provider and the end-point location.

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

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

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



EBS Steps

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

EBS Message Exchange Patterns

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

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

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

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

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

Synchronous Request-Response Patterns in EBSs

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

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

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

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

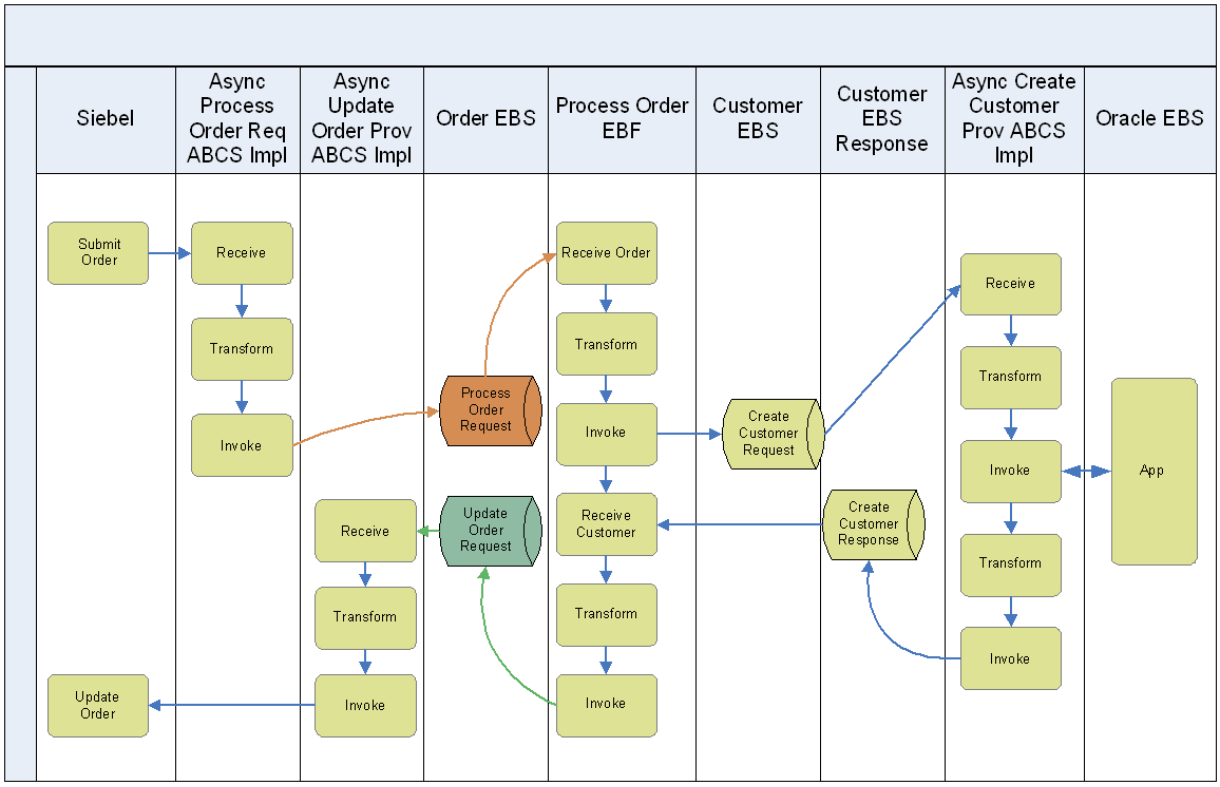
Asynchronous Fire-and-Forget Patterns in EBSs

A fire-and-forget EBS operation pattern is asynchronous in nature. This is an event that leads to a request message being posted to an endpoint or placed in a channel/queue/topic. This pattern is also characterized as 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. There is no concept of a response even in the case of an error. In the Order EBS WSDL, the one-way operation meant for request is defined in the portType having all the operations, which are either request – response or request only.

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

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



Order EBS Showing Fire-and-Forget Scenario

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

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

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

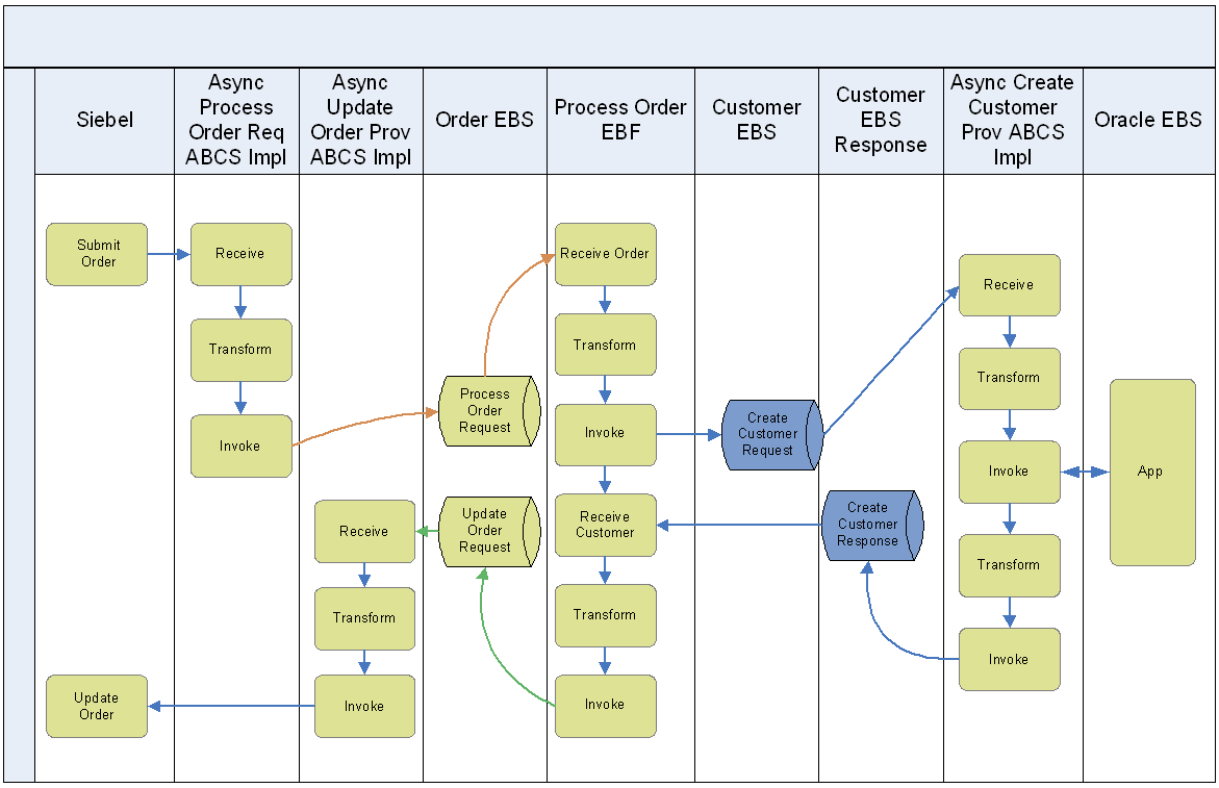
Asynchronous Request–Delayed Response Patterns in EBSs

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

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

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

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



One-way Request and One-way Response

To increase the reliability of message delivery, there will be a need to resort to persistence at strategic asynchronous points. This is discussed in more detail in the chapter on Guaranteed Message Delivery Designs.

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

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

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

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

   <svcdoc:InitiatorInterface>CreateSalesOrderRequestEBM</svcdoc:Initiato
rInterface>

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

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

Chapter 5: Understanding ABC Services

This chapter provides an overview of application business connector (ABC) services and discusses:

- ABC service architecture
- ABC service characteristics
- Architectural considerations
- Implementing ABC services
- Requestor-side ABC services
- Provider-side ABC services
- Reviewing implementation technologies for ABC services
- Extending or customizing ABC service processing
- Processing multiple instances
- Participating applications invoking ABC services
- ABC service transformations

ABC Services

The role of the ABC service is to expose the business functions provided by the participating application in a representation that is agreeable to enterprise business services (EBSs). It also serves as a glue to allow the participating application to invoke the EBSs.

An ABC service can be requestor-specific or provider-specific. A requestor ABC service accepts the request from the client application through a client-specific application business message (ABM) and returns the response to the client application through a client-specific ABM. The role of the requestor ABC service is to act as a vehicle to allow the participating application to invoke the EBS either to access data or to perform a transactional task. The client side ABM will be the payload that is passed by the requestor application to the requestor ABC service.

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

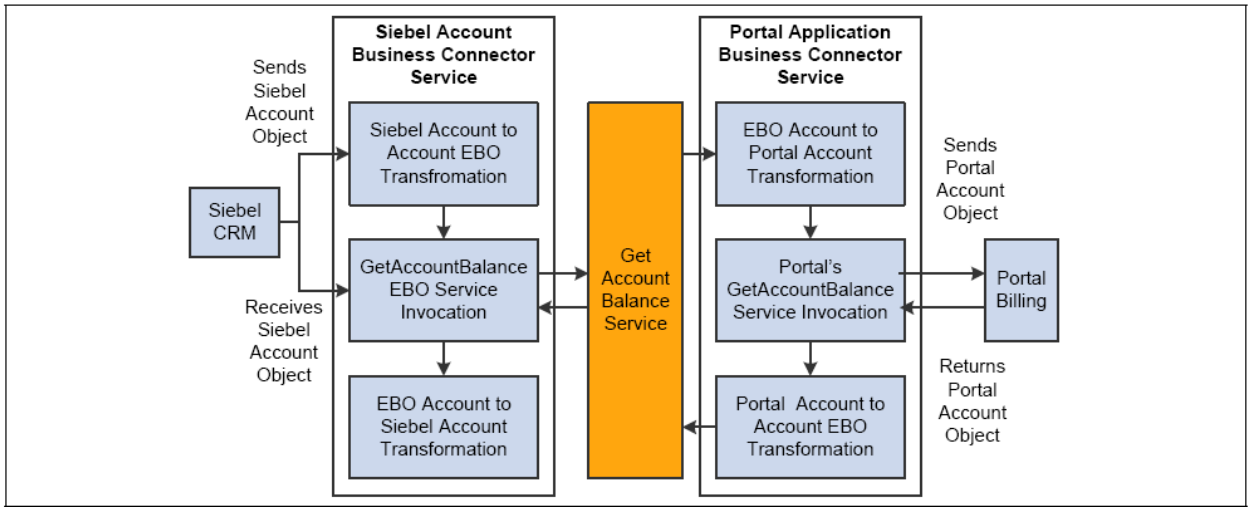
The role of the provider ABC service is to expose the business capabilities that are available in the provider application according to the EBS specification. The service-provider-side ABC service will accept the request from the EBS through the enterprise business message (EBM) and will send the response using the same format. Application business connector services are needed because every application has a different representation of objects, and any communication between applications necessitates the transformation of these objects to the canonical definition.

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

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

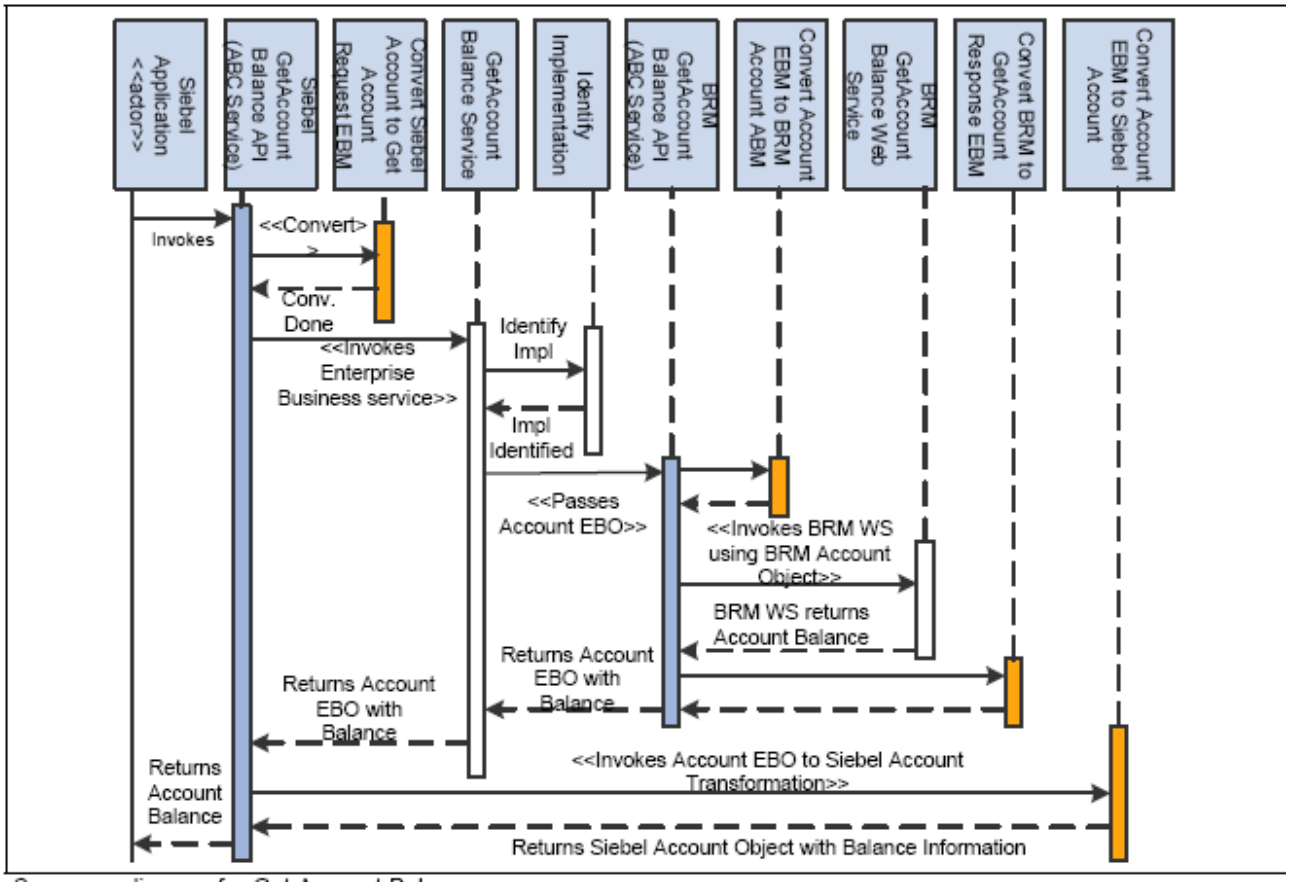
For each of the activities that can be performed with an enterprise business object (EBO), an ABC service must be defined by the participating requestor application and another ABC service must be defined by the service provider application. In the use case, *Query Customer Party* could be an action for the Customer Party EBO. In this case, a Query Customer Party provider ABC service must be created by the service provider application (in the use case, Oracle BRM) and a Query Customer Party requestor ABC service must be created by the service requester application (for the use case, Siebel CRM).

The following data flow diagram depicts the end-to-end flow of events between the requestor and the provider. Notice that the Siebel GetAccount ABC service invokes only the *Get Account Balance Service* and is unaware of the events occurring within that service.



End-to-end Flow of Events between Requestor and Provider

This diagram illustrates the end-to-end flow of the *Get Account Balance* process.



Example of Get Account Balance Process

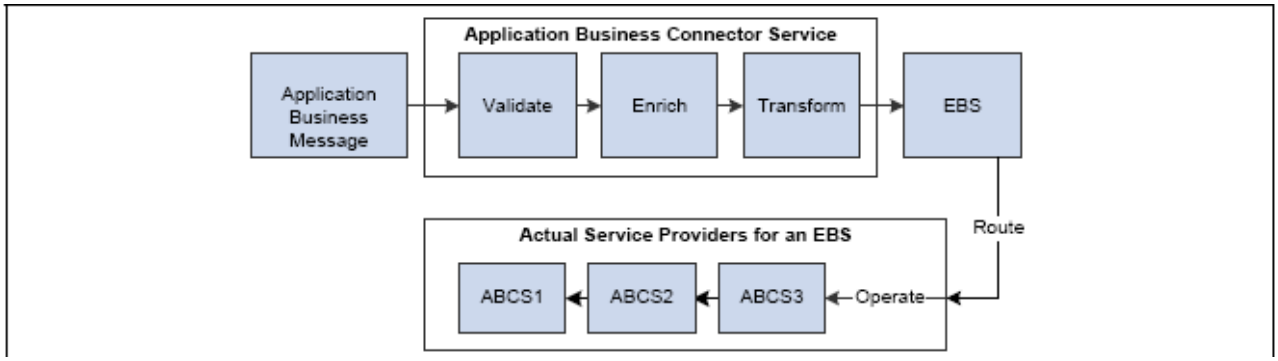
ABC Service Architecture

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

Similarly, the application that receives messages from EBS must be able to understand the message types. Very few applications are built to readily interact with EBS. For applications that are not able to readily interact with EBS, ABC services act as conduits to expose the application-specific business functions in the enterprise service bus (ESB).

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

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



VETORO Pattern

The VETORO pattern is composed of the following steps:

- Validate

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

- Enrich

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

- Transform

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

- Operate

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

- Route

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

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

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

ABC Service Characteristics

The ABC service has the following characteristics:

- For each of the activities that can be done on an EBO, an ABC service must be provided by each of the participating requestor and provider applications.

  For the use case, one ABC service would be provided by Siebel CRM and another ABC service provided by the BRM Billing application. If the PeopleSoft CRM system wants to play a client role in this process, it needs to provide a requestor ABC service.

- The requestor ABC service has participating application-specific ABMs as input as well as output.

  The service accepts requestor application-specific ABMs as input and provides requestor application-specific ABMs as output.

- The provider ABC service has EBMs (representing specific content of EBO needed for performing the operation) as input as well as output.

  The service accepts EBMs as input and returns EBMs as output.

- Although a single ABC service can be used to handle multiple activities, *Oracle highly recommends allowing only a single ABC service per action*. This approach greatly reduces the complexity of designing a generic ABC service.

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

Architectural Considerations

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

Participating Application's Service Granularity

Perform some analysis to identify how the participating applications intend to expose their business functionality to the ESB. If the applications expose their functionality using a web service interface, verify that the granularity of functionality matches exactly that of what has to be exposed as an EBS.

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

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

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

This type of ABC service can be implemented only through BPEL technologies and not through ESB services.

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

Support for EBMs

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

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

Application Interfaces

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

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

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

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

Support for Logging and Monitoring

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

For more information, see Logging and Error Handling.

Support for Insulating the Service Provider

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

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

Support for Security

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

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

Validations

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

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

Because not all participating application-specific constraints can be enforced at the EBM level, these validations must be enforced by the specific ABC service. 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 ABC service will be responsible for ensuring that the ABM generated from the EBM includes the product description. This validation will ensure that the ABM passed to the PeopleSoft CRM Create Product service is compliant with the constraints imposed by the service.

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

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

Support for Internationalization and Localization

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

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

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

Message Consolidation and Decomposition

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

Support for Multiple Application Instances

Situations will occur where multiple instances of a packaged application having same business capability existing in a customer's ecosystem. The routing rules defined in the EBS will be responsible for deciphering the right application instance to which the request needs to be routed. Regardless of number of application instances for a packaged application, there will be only one ABC service for that packaged application to perform a specific business task.

Implementing ABC Services

Each of the participating applications implementing a business activity or task for an EBO will have its own ABC service.

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

The ABC service can be implemented in two ways:

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

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

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

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

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

For example, in the use case, the Siebel CRM application will invoke the Siebel application-specific ABC service for Query Customer Party to get the balance. Similarly, the EBS for Query Customer Party will invoke the BRM application-specific ABC service for Query Customer Party.

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

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

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

This type of ABC service can be implemented using only BPEL technologies and not through ESB services.

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

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

For more information, see EBM Architecture.

Requestor-Side ABC Services

The requestor-side ABC service has the following core responsibilities:

- Enrichment or augmentation of the ABM.

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

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

For more information, see EBM Architecture.

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

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

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

- Invocation of the EBS.

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

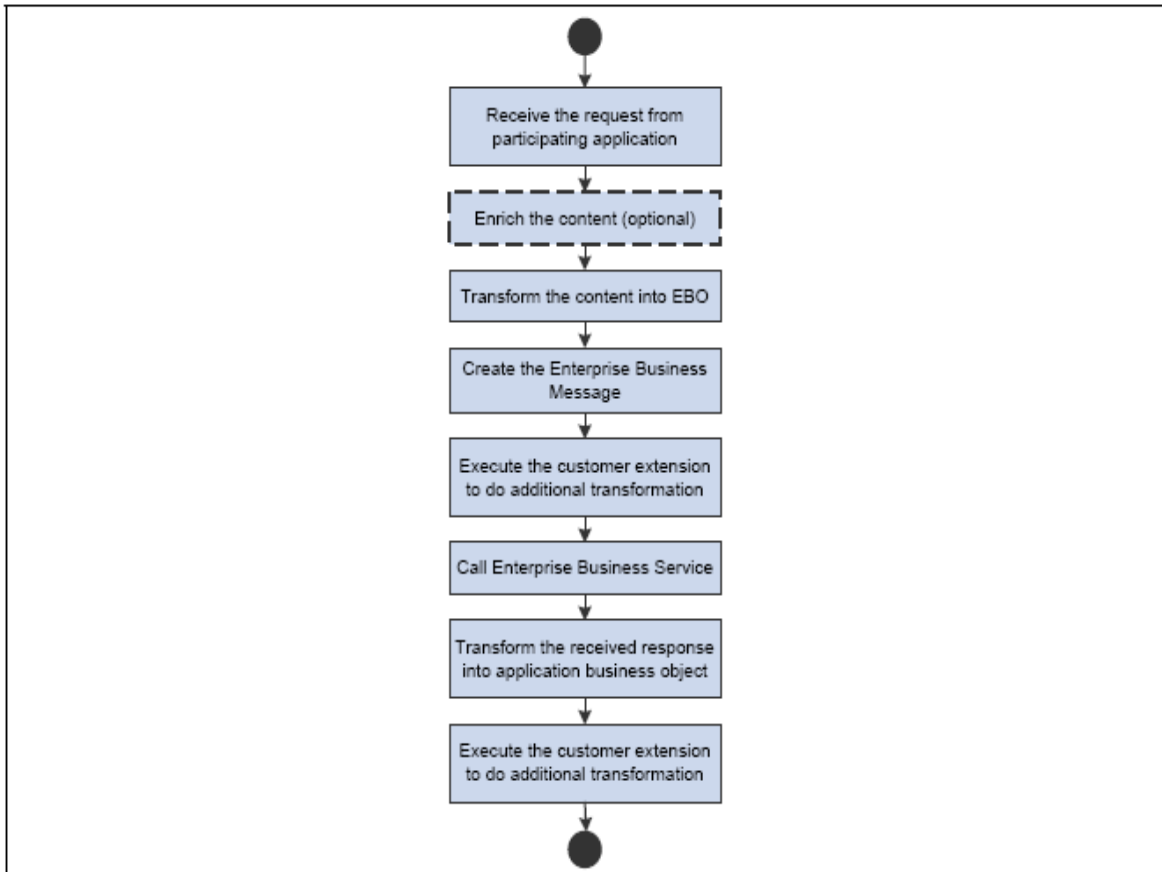
- Transformation of an EBM into an ABM.
- Invocation of any extension handler that the customer may have registered.

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

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

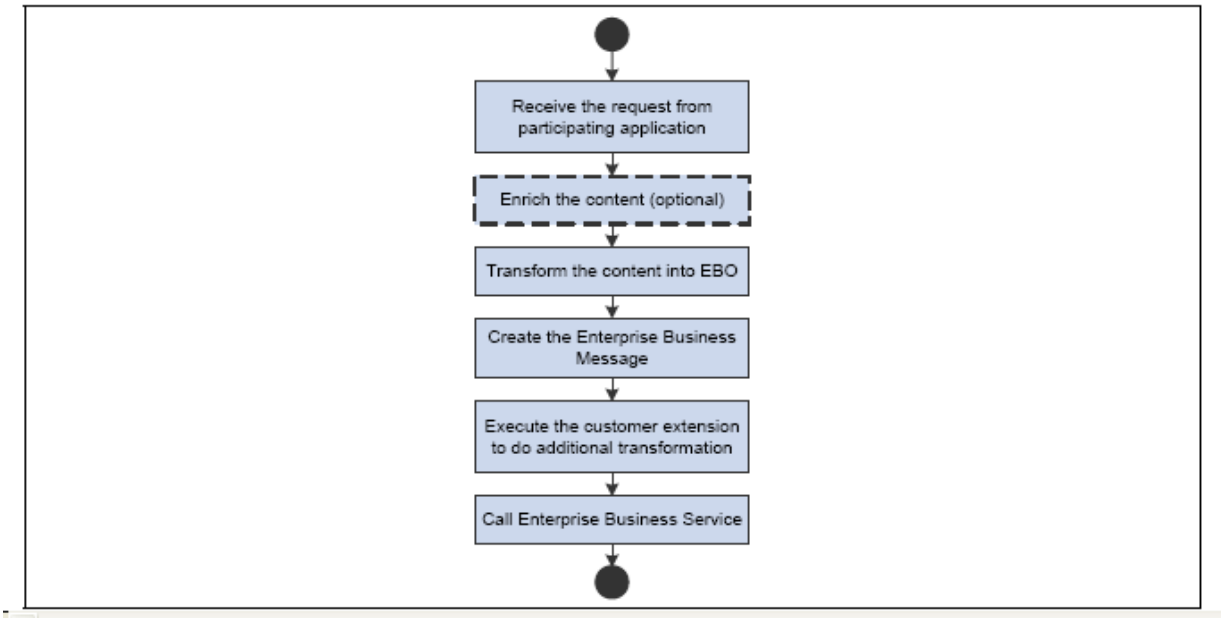
- Perform any necessary validations to ensure that the ABM that needs to be passed to the participating client application complies with the constraints enforced by the participating application.
- Return the response to the calling application.

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



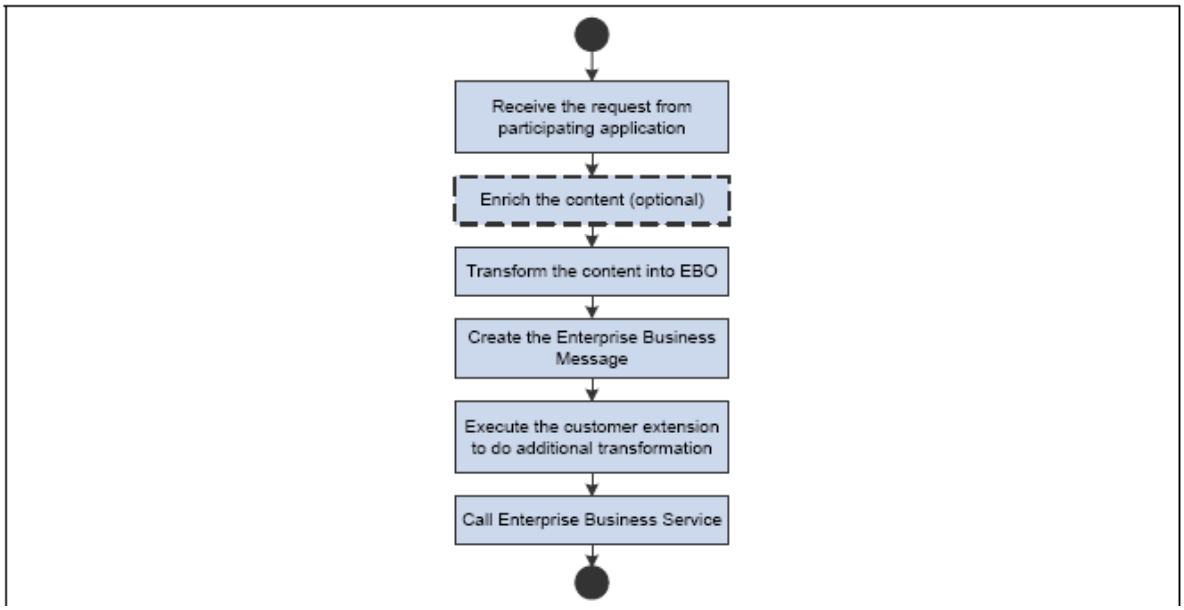
Requestor specific ABC Service Interacting with Request/response Interaction Style

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



Requestor-specific ABC service interacting with request/response interaction style

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



Requestor-specific ABC service interacting with fire-and-forget interaction style

Provider-Side ABC Services

The provider-side application business connector service has the following core responsibilities:

- Transformation of an EBM into a provider application-specific ABM.

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

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

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

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

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

  For more information, see Architectural Considerations.

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

  For more information, see EBM Architecture.

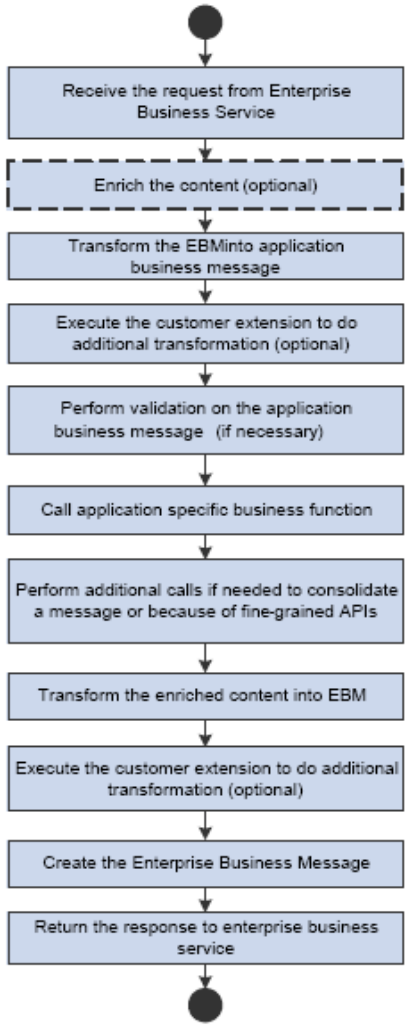
- Invocation of any extension handler that a customer may have registered.

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

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

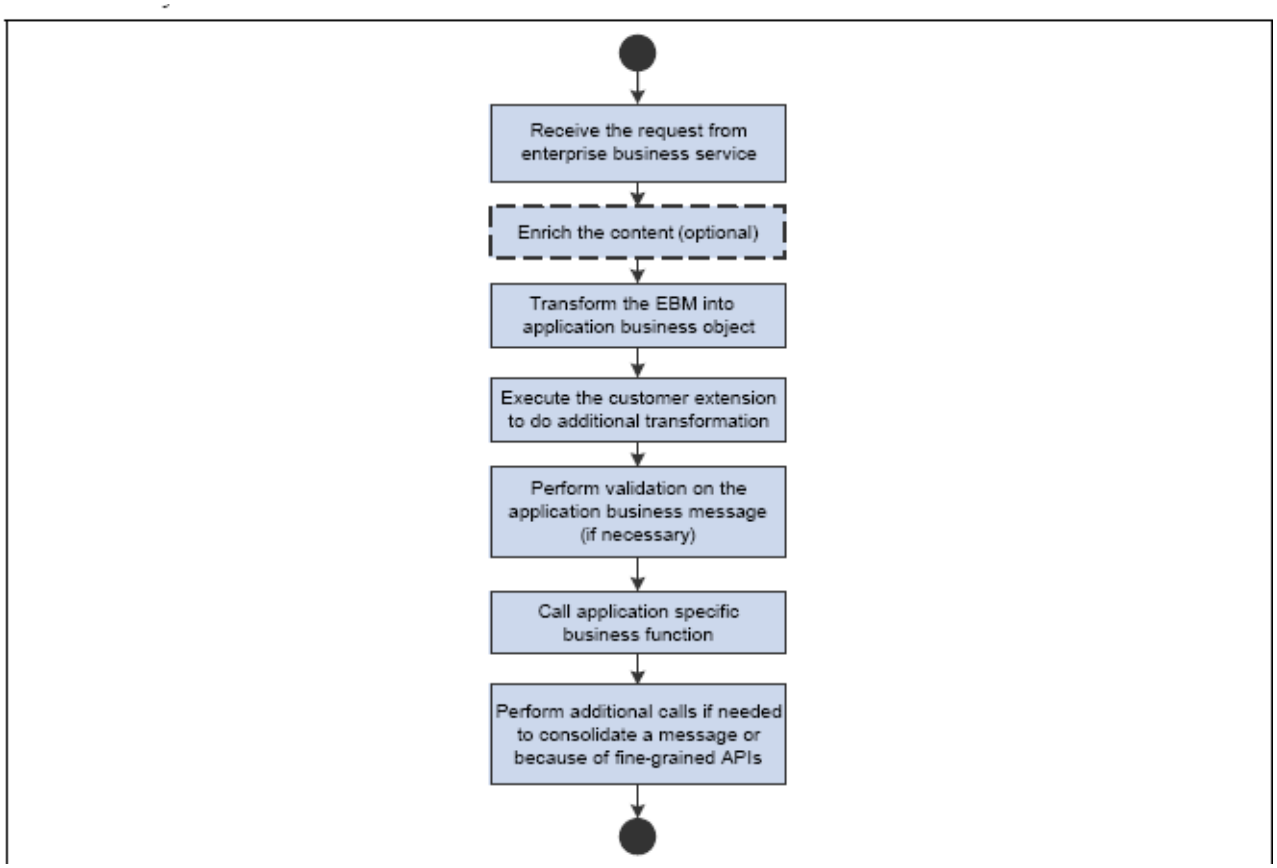
- Return of the document to the EBS.

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



Provider-specific ABC service interacting with request/response interaction style

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



Provider-specific ABC service interacting with fire-and-forget interaction style

Reviewing Implementation Technologies for ABC Services

Oracle AIA provides two blueprints for implementing an ABC service: the ESB and BPEL.

ESB

The ESB blueprint can be applied in situations where you do not need the ABC service to do additional enrichment and validation of the content. In this model, the ABC services are implemented as ESB services.

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

BPEL

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

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

BPEL will also be used when an ESB service can't be used to implement the ABC service either due to the constraints in ESB technology or due to the complexities.

Extending or Customizing ABC Service Processing

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

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

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

Processing Multiple Instances

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

The ABC service can decide whether its application has the ability to process the instances in bulk form. If the application has the ability, then the ABC service 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 ABC service transforms the content of each instance back into EBM format, consolidates them in an EBM, and returns the message to the EBS.

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

Participating Applications Invoking ABC Services

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

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

For more information, see EBM Architecture.

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

ABC Service Transformations

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

Transformation: Implementation Approach

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

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

The ABC service transformations handle:

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

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

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

Static Data Cross-Referencing

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

Dynamic Data Cross-Referencing

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

Structural Transformation

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

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

ABC Services in the Use Case

In the use case, the requestor application (Siebel CRM) must have an ABC service developed for Query Customer Party specific to Siebel. This service will be invoked by the Siebel application when you click the Get Account Balance button. The service will accept Siebel ABM as the input.

This ABM will contain the information such as customer ID (existing in the Siebel application). The Siebel Query Customer Party ABC service will be responsible for transforming the message into the EBM. It will leverage the cross-referencing information to transform the Siebel customer identifier into an application-independent customer identifier. After construction of the EBM, the service will be responsible for invoking Query Customer Party service operation. The message returned by this service will have complete details about the customer (along with the balance that this use case is about). The ABC service will do another transformation—in this case, in the opposite direction. The generated ABM will then be returned to Siebel CRM.

Similarly, an ABC service exists on the provider side, Query Customer Party for the BRM application. This service takes the EBM as the input and returns the EBM as the response. After taking the Query Customer Party EBM, the ABC service transforms the message into ABM and all application-independent values are transformed into BRM-specific values. For example, the application-independent customer identifier will be transformed into a BRM customer identifier. The ABC service will then invoke BRM services using the BRM adapter (J2CA) and get the customer details. It does another transform, this time in the opposite direction, and transforms the ABM into and EBM. It then returns the EBM to EBS.

Chapter 6: Understanding Interaction Patterns

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

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

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

Patterns for Exchanging Messages

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

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

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

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

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

Request/Response

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

Synchronous Response

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

- Use case

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

- Synchronous requestor/response

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

Fire-and-Forget

Business problem: A customer integrating two applications doesn't want the sender application to be affected due to non-availability of the receiving application.

- Use case

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

- Asynchronous transaction using queue/topic

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

Message Routing

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

- Use case

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

- Message routing/mediation

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

For more information, see Chapter 4: Understanding EBSs.

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

- Use case

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

- Message routing/mediation

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

For more information, see Chapter 4: Understanding EBSs.

Message Splitting and Routing

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

- Use case

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

- Message splitting and routing

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

Data Enrichment

Business problem: The requestor application does not send all of the required data that is necessary for invoking the EBS.

- Use case

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

- Data enrichment of the ABM

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

Data Aggregation

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

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

- Use case

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

- Data aggregation of the ABM

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

Business problem: Information needed for providing the response is spread across multiple applications.

- Use case

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

- Data aggregation of the ABM

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

Asynchronous Request – Delayed Response Pattern

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

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

Publish-and-Subscribe

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

For more information about the publish-and-subscribe interaction pattern and implementing the Publish-and-Subscribe programming model, see *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide,* "Programming Models."

Chapter 7: Understanding Extensibility

This chapter provides an overview of extensibility and discusses:

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

Extensibility

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

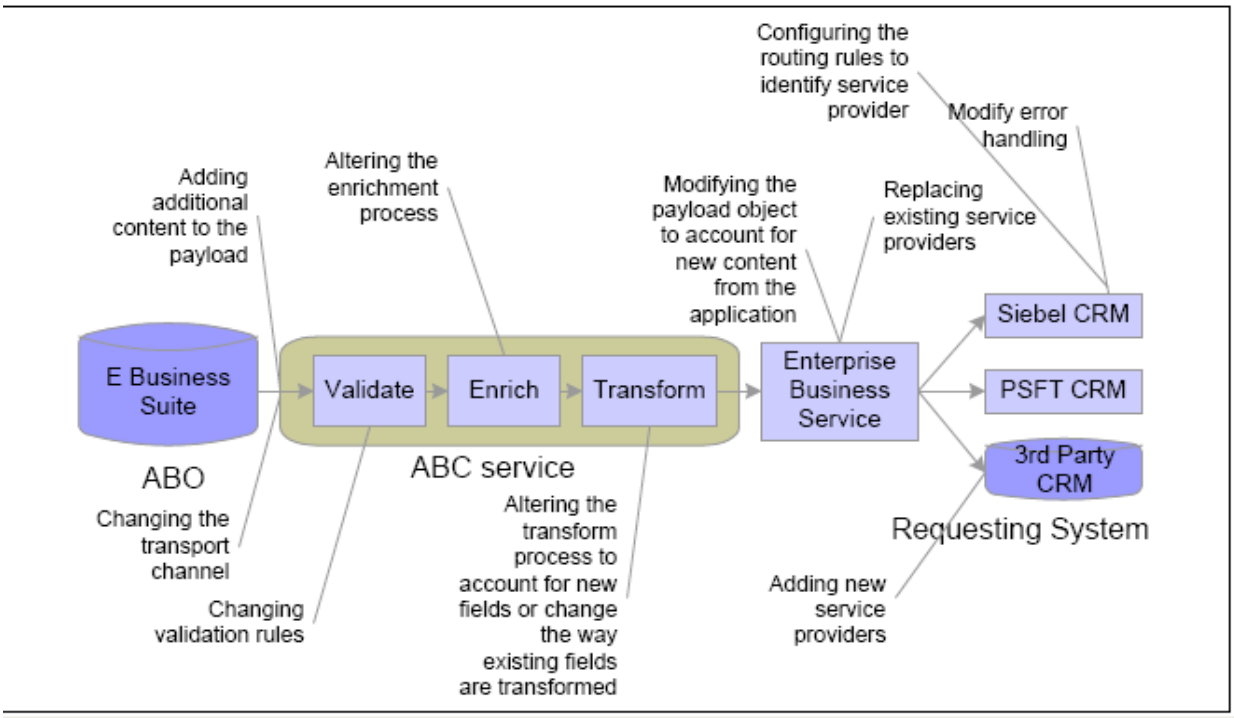


Illustration of Customer Extensibility Points

Schema Extensions

Oracle AIA facilitates the extension of delivered enterprise business objects (EBOs) to accommodate the introduction of industry-specific and customer-specific needs.

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

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

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

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

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

Two approaches are available for implementing extensibility:

- Customer-specific extensions
- Industry-specific extensions

Customer Extensions

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

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

Industry-Specific Extensions

The architecture allows for incorporating industry-specific attributes as overlays. An industry-specific object can be created by assembling a set of business components that are available at the core with a set of industry-specific components.

Schema in the Use Case

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

Transformation Extensions

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

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

Extensions in the Use Case

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

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

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

Transport/Flow Extensions

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

Process Extensions

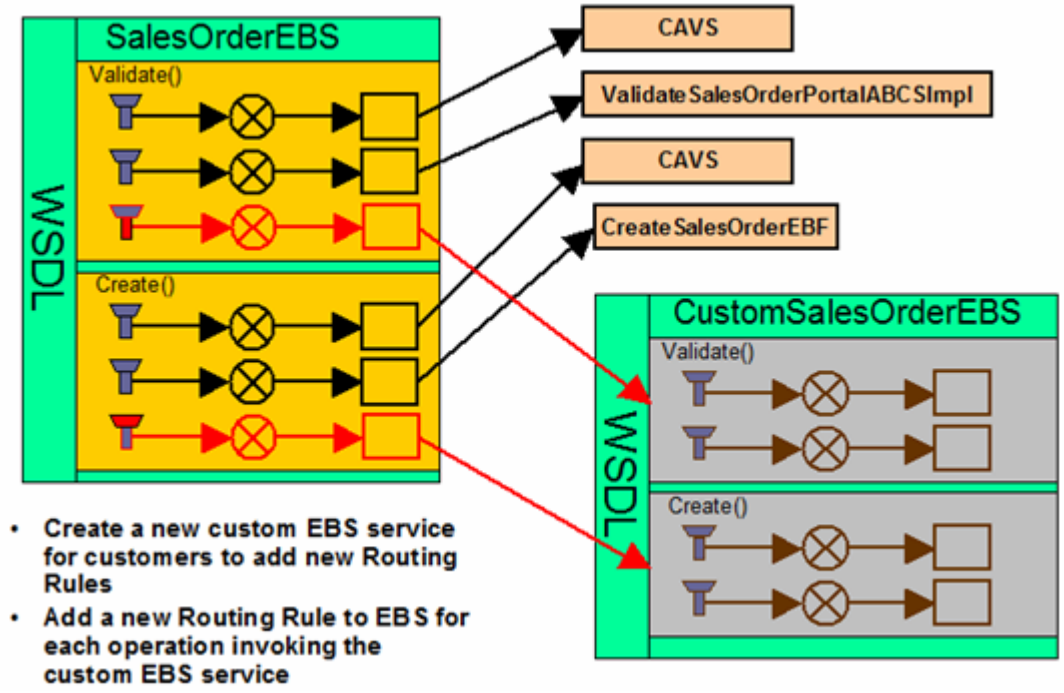
The architecture provides recommendations on how the ABC services as well as the orchestration processes can be designed to allow the customers to introduce the extensions to augment the functionality.

Each of the BPEL processes can have its own set of extensibility points based on the functional needs. You can implement the interface that will be defined for each of the extensibility points to either augment or override the behavior. ABC services are recommended to have a minimum of 4 extensibility points, in case of request/response pattern, to allow customers to inject additional behavior. In case of fire-and-forget pattern, the services are expected to have a minimum of 2 extensibility points. The orchestration processes might decide not to have any extensibility points.

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

Routing Extensions

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



Routing Extensions

For more information about whether a Process Integration Pack supports routing extensions, see the respective Process Integration Pack implementation guide.

Chapter 8: Understanding Versioning

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

This chapter discusses:

- Schema versioning
- Service versioning
- Participating applications versioning

Schema Versioning

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

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

Major and Minor Versions

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

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

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

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

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

- Adding optional components, such as elements, attributes, or both.
- Adding optional content, such as extending an enumeration.

- Adding, changing, or removing default initializations; changing the order of elements in a choice; and so forth.

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

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

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

A release that updates the major version number of an EBO contains changes that might sometimes make it incompatible with the prior major release. This means that consumer applications that depend on an earlier major release might need to be modified to work with the new release. On the other hand, a release that updates the minor version number of an EBO is a backward-compatible change. This means that an application written against version 1.0 will work when targeted against versions 1.1 and 1 .2, but may fail if moved to version 2.0 of the EBO.

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

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

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

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

Namespaces

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

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

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

Here is a sample namespace:

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

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

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

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

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

Service Versioning

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

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

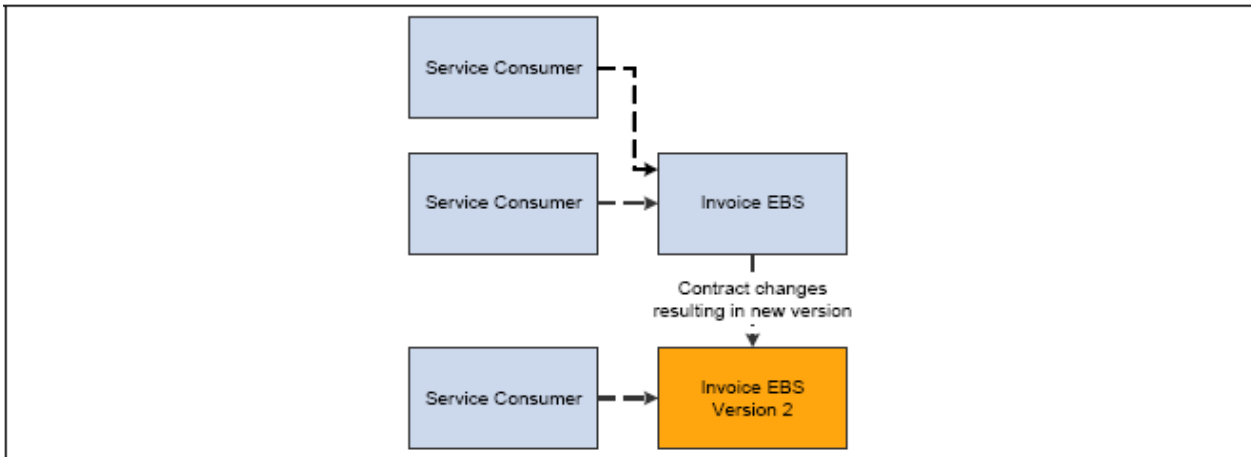


Illustration of Versioning

Naming Conventions

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

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

Participating Applications Versioning

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

Chapter 9: Understanding Batch Processing

This chapter discusses batch processing.

Batch Processing

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

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

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

Chapter 10: Understanding Infrastructure Services

This chapter discusses:

- Logging and error handling
- Composite Application Validation System (CAVS)
- Business Service Registry (BSR)
- Deployment
- Internationalization and localization support

Logging and Error Handling

Error handling and logging support the needs of integration services operating in an Oracle Application Integration Architecture (AIA) ecosystem.

Error handling provides the following key functions for the integration services operating in an Oracle AIA ecosystem:

- Error logging.

  Logs error messages in a consistent schema in a nonintrusive manner. Error notifications are sent to suitable actor roles, such as integration administrators, and FYI roles, such as customer service representatives.

- Error actions.

  Automatically acts upon the errored object. For example, error actions provide an automated retry action. The Oracle AIA Common Error Handler is invoked whenever an error occurs in the Oracle AIA ecosystem. This module initiates error logging and error notifications in a consistent manner regardless of where the error occurred, whether in BPEL or enterprise service bus (ESB).

Oracle AIA enables you to generate trace and error log files that provide a detailed view of the services running in your Oracle AIA ecosystem. These logs can be especially informative when you are troubleshooting service processing issues. Logging is implemented using the Oracle Diagnostic Logging (ODL) framework and logs are created using the ODL log schema.

Oracle AIA generates two types of log files through ODL:

- Trace

  Trace logs capture chronological recordings of general activities of a service. The trace log is created by configuring the service to make an explicit call using the trace logging custom XPath or Java API.

- Error

Error logs capture a recording of errors that occur during the activities of a service. No specific configurations are required to make BPEL and ESB services eligible for error logging. The Error Handling Framework is designed to trigger an error logging event for errors occurring in any of the Oracle AIA services, whether they are BPEL-based or ESB-based. The Error Handling Framework does this logging non-intrusively.

These log files can be managed and viewed using the Oracle Enterprise Manager (OEM) user interface in much the same way that standard log files generated by various components of the Oracle SOA Suite can be handled in OEM. Using OEM as the user interface for the logs enables searching, sorting, and filtering of logs.

Composite Application Validation System (CAVS)

The CAVS enables you to test integration web services without the deployed participating applications in place. Using a framework that includes initiators to simulate calls to participating applications and simulators to simulate responses from participating applications, CAVS provides a system that can test integrations while also eliminating the need to set up deployments of all participating applications that are involved in the integration. The CAVS initiators and simulators enable a layered testing approach, allowing you to test requester and provider services separately, without needing to link them together. Each component in an integration can be thoroughly tested without having to account for the presence of participating applications and other dependencies. Consequently, when you build an integration, you can add new components to an already-tested subset, allowing any errors to be constrained to the new component or the interface between the new component and the existing component.

The CAVS provides a test repository, an execution engine, and a user interface. Tests can be configured to run in a single-threaded batch. Each execution of a test generates a log, which can be used for troubleshooting. While CAVS can test any web service, in the context of Oracle AIA, CAVS is used to validate the delivered process integration packs (PIPs) by testing the individual application business connector (ABC) services that comprise the PIP. That is, your PIP has been delivered with CAVS test definitions, test simulators, and test groups that have been developed to specifically test and validate the PIP out-of-the-box.

The CAVS provides value as a testing tool throughout the integration development life cycle, including:

- Unit test

  Integration developers can build integration components and easily test them throughout the development cycle, without the presence of participating applications and other dependencies.

- System test

  Quality assurance engineers can test integrations without the presence of participating applications and other dependencies. The reusability of test definitions, simulators, and test groups is a time saver.

- Validation

  Use the CAVS content that is delivered with a PIP to validate that the delivered deployment of the PIP works.

Business Service Registry (BSR)

The BSR stores and provides information about the objects, messages, and services that compose the integration scenarios in your Oracle AIA ecosystem. An integration scenario refers to the end-to-end (requester participating application-to-provider participating application) flow of a message.

The BSR is secure and UDDI v3-compliant, and provides a foundation for the life cycle management of services in your Oracle AIA ecosystem. In this way, the BSR is a key component in defining and enforcing service-oriented architecture (SOA) governance for your Oracle AIA ecosystem.

The primary objectives of the BSR are to provide:

- A mechanism for publishing content into the BSR.

  The BSR includes a command-line utility that enables you to publish single or multiple WSDL, XML, and XSD files into the BSR.

- A user interface (UI) for discovering and learning about the integration scenarios in your Oracle AIA ecosystem.

The BSR presents the components of your integration scenarios in a centrally managed and searchable repository, enabling it to become the system of record for your business services.

Oracle AIA Setup UI

The BSR provides a UI for managing information about the applications that are participating in your Oracle AIA ecosystem. The BSR also provides a UI for managing mappings between actor and FYI roles and their participating applications for use during Oracle AIA error notifications.

Potential BSR users are active across the span of the SOA artifact life cycle and include functional and business analysts, architects, developers, system integrators, and system administrators. For example, a business analyst working on requirements for building out a particular business process can use the BSR integration scenario UI to determine which services are available in a particular integration area and be able to determine which additional services may need to be built. System administrators can use the BSR setup UI to manage the application registry of the Oracle AIA ecosystem.

For more information, see *Oracle Application Integration Architecture – Foundation Pack: Core Infrastructure Components Guide,* “Understanding the BSR.”

Deployment

The infrastructure framework provides deployment tools for automating the migration of artifacts from development to test to deployment environments. Process and service dependency analysis identifies all artifacts that are required for a given deployment.

Internationalization and Localization Support

To accommodate heterogeneous locale-specific participating applications in the Oracle AIA ecosystem, the architecture will have comprehensive support for internationalization and localization. It will house language and country variants. It will also support the major categories of locale-dependent processing.

Internationalization support will ensure that locale-specific content is represented in locale-agnostic form in enterprise business objects (EBOs). For example, the date will be represented in ISO format. In situations in which the translation is not possible, the EBOs will carry the content as is, but will also specify the locale to which it is applicable. For example, the product description element will have the language attribute that indicates the locale. The ABC service will be responsible for translating the locale-specific application business object (ABO) into the locale-agnostic EBO. For this to happen, the ABC service needs to know the locale of the user who was used to create the ABO that was handed over to the service. The locale-sensitive participating application could either communicate locale settings using the ABO, or the ABC service could have a mechanism to derive the locale.

In addition to internationalization support, the architecture needs to provide support for localization to ensure that EBOs that are handed over to the participating application contain content represented in locale-specific form. The ABC service that passes the EBO to the actual service provider will be responsible for translating the document into a locale-specific ABO.

ABC services will have the capacity to handle different locales that are set per client. This denotes that locale-specific processing for each client context can occur in an ABC service. The ABC service deciphers the locale of the user who is used to interact with the participating application. This data will provide information about how the locale-specific ABO needs to be constructed and how the locale-specific ABO needs to be interpreted.

The Oracle AIA Developer's Guide contains detailed best practices regarding the way in which locale-specific computing for each client context can be carried out in an ABC service.

For more information, see *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide*

Chapter 11: Understanding Security

This chapter discusses security.

Security

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

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

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

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

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

- Decrypt the incoming XML message
- Extract the user's credentials
- Perform an authentication for this user
- Perform an authorization check for this user and this web service
- Write a log record of the above information
- Pass the message to the intended web service, if all steps are successful

- Return an error and write an exception record, if all of the steps are not successful

To apply the security policy, OWSM intercepts every incoming request to a web service and applies any one of the policy items listed here. 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.

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

Because atypical interaction in the Oracle AIA framework will be part of multiple discrete interactions involving a service requestor, client-specific application business connector (ABC) service, enterprise business service (EBS), server-specific ABC service and the service provider, choosing a security model plays a critical role.

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

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

- Can an entire transaction be considered as secured as long as the individual discrete transactions are secured?
- Decisions also have to be made regarding whether the *trusted model* expressed above can be agreed to in principle, or if it must be enforced using certificates provided by a certificate authority for the discrete interactions.
- Can the communication-level security methods such as SSL encryption be used to secure the individual discrete transactions within a trusted model?
- Adoption of the industry-standard WS-Security security model is possible, provided that all participating applications in the transactions provide inherent support.

Transport-Level Security

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

Message-Level Security

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

Securing ABC Services

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

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

Implementation Techniques for Enterprise Service Bus Security

Consider whether or not the client identity and password need to be transmitted from one end to another using the enterprise business message (EBM) header.

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

Index

ABC service implementation technologies.....55
BPEL.....55
ESB.....55
ABC service transformations.....57
dynamic data cross-referencing.....58
static data cross-referencing.....58
structural transformation.....58
transformation - implementation approach.57
ABC services
application interfaces.....46
architecture.....43, 45
characteristics.....45
extending processing.....56
implementation.....48
internationalization.....48
localization.....48
message consolidation and decomposition48
participating applications invoking.....57
participating application's service granularity 45
provider.....41, 52
requestor.....41, 49
responsibilities.....41
security.....83
support for EBM.....46
support for insulating the service provider..47
support for logging and monitoring.....46
support for multiple application instances..48
support for security.....47
validations.....47
VETORO pattern.....44
application business connector services.....*See* ABC service
asynchronous fire and forget pattern.....36
asynchronous operations.....61
asynchronous request – delayed response pattern.....66
asynchronous request–delayed response pattern.....38
backward compatibility.....72
batch processing.....75
BSR
Oracle AIA setup UI.....79
overview.....79
Business Service Repository.....*See* BSR
canonical data model.....15
CAVS
overview.....78
system test.....78
unit test.....78
validation.....78
Composite Application Validation System....*See* CAVS
customer extensions.....68
data aggregation.....65
data enrichment.....64
data integration.....11
dynamic data cross-referencing.....58
EBF
processes.....31
EBM
architecture.....21
considerations.....21

header....22
EBO
characteristics....17
in the Use Case....18
EBS
architecture....28
asynchronous request-delayed response pattern....38
content based selection of the service provider....29
implementation....33
message exchange patterns....35
operations....24
purpose....30
responsibilities....34
reusing available assets....28
substituting one service provider with another....29
synchronous request-response pattern....36
types....24
verbs....24
enterprise business messages....*See EBM*
enterprise business objects....*See EBO*
enterprise business services....*See EBS*
Enterprise Service Bus....*See* ESB
entity services....25
characteristics....25
standard activities....25
error actions....77
error logging....77
error logs....77
ESB....14
extending ABC service processing....56
extensibility....67
fire and forget....62
forward compatibility....72
functional integration....11
integrating systems....11
integration styles....15
interaction patterns
asynchronous request-delayed response....66
data aggregation....65
data enrichment....64
fire and forget....62
message routing....63
message splitting and routing....64
publish-and-subscribe....66
request/response....62
synchronous response....62
internationalization....79
localization....79
log files....77
error....77
trace....77
logging and error handling....77
major versions....71
message exchange patterns....35
asynchronous request-delayed response pattern....38
synchronous request-response pattern....36
message routing....63
message splitting and routing....64
message-level security....82
minor versions....71
namespaces....72
naming conventions for versioning....74
operations....24
Oracle AIA capabilities....12
participating applications invoking ABC service....57
process extensions....70

process integration 11
process services 26
  characteristics 26
processing multiple instances 56
provider ABC service 41
provider ABC service 52
publish-and-subscribe 66
request/response 62
requestor ABC service 41
requestor ABC service 49
routing extensions 70
schema extensions 68
schema versioning 71
security 81
  ABC service 83
  message-level 82
  point-to-point 82
  transport-level 82
Service Oriented Architecture ..... *See* SOA
service versioning 73
SOA 14
solution artifacts 13
static data cross-referencing 58
structural transformation 58
synchronous operations 61
synchronous request-response patterns 36
synchronous response 62
trace logs 77
transformation extensions 69
transformations
  ABC service 57
  implementation approach 57
transport/flow extensions 69
transport-level security 82
use case
  ABC services 58
  extensions 69
  introduction 16
user interface integration 11
verbs 24
version number 71
versions
  backward compatability 72
  forward compatability 72
  naming conventions 74
  participating applications 74
VETORO 44