An Oracle White Paper
August 2009

Oracle Application Integration Architecture Enterprise Business Objects (EBO) Concepts – Concepts, Structure, Terminologies and Design Rules

ORACLE

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

[NOTE: This revenue recognition disclaimer is required for any white paper that addresses functionality or products that are not yet generally available. Most white papers will NOT need this disclaimer. To determine whether your paper requires this disclaimer, read the revenue recognition policy at http://files.oraclecorp.com/content/AllPublic/SharedFolders/GDMI-Public/REFG-P06.htm#ZZZ_TUT_0. If you have further questions about your content and the disclaimer requirements, e-mail REVREC_US@oracle.com.

To remove both the disclaimer and the page that it appears on, first display hidden characters (Tools>Options>View tab>Formatting marks>All). Notice that there is a section break displayed as a double-dotted line at the bottom of this page. Highlight all the text on this page and press the Delete key. Continue to press Delete until the page disappears and your cursor is on the Table of Contents page. Be sure not to remove the section break, or the formatting of the title page will be incorrect.]

Introduction

Objectives

This document outlines the concepts, standards, structure, and terminology involved in the process of Enterprise Business Objects (EBO) development. This document describes the building blocks of the EBOs, the technical prescriptions made during the process, the design choices, the structure of the Enterprise Object Library, and extensibility.

Target Audience

This document is for those who want to know about the EBO concepts and how the Foundation Packs help overcome the most common, yet critical, application integration challenges by providing a set of prebuilt enterprise objects and services, an application integration management infrastructure, and a proven application integration methodology. The document also outlines the design challenges and the standards that need to be followed.

This white paper is primarily targeted for those who have prior knowledge of Application Integration Architecture (AIA) so that they can build an EBO/EBM/EBS from scratch.

Application Integration Architecture (AIA) Introduction

Support for Standards

The Enterprise Business Object (EBO) structure and design is based primarily on the UN/CEFACT XML Naming and Design Rules. It is also based on the XML Naming and Design Rules provided by the Open Application Group and UN/CEFACT Core Component Technical Specification. This UN/CEFACT –Core Components Technical Specification is employed wherever business information is being shared or exchanged amongst and between enterprises, governmental agencies, and/or other organizations in an open and worldwide environment. While these specifications provide a base for design and structure of business objects, they have a larger scope in relation to mapping rules, rule-by-rule profiling, and more.

Industry Best Practices

One of the components of AIA is the Reference Process Models. These are industry specific process models that transcend both Oracle and partner applications. These models outline best practices in customers' industries and provide a model from which customers can start their process designs. In that sense these models are similar to Oracle Business Flows with the key differences that the models are:

- Industry specific
- Application agnostic

The Reference Process Models included in the AIA Foundation Pack follow industry best practices. These business processes are used to integrate two or more applications through the Enterprise Business Services/Objects layer.

Oracle AIA's Foundation Pack Extensions for Industries cater to various industry verticals including Communications, Insurance, and Utilities. The Oracle Application Integration Architecture **Foundation Pack Extension for Communications** enables the customers to simplify cross-application business process integrations using a productized integration solution designed for re-usability and configurability. With it, customers can create robust integrations on a standardized framework, leveraging their existing Oracle and non-Oracle application investments. To help customers accelerate their integration implementation, AIA Foundation Pack Extension for Communications includes various industry-specific EBOs. Communications industry-specific EBOs include:

- Communications FulfillmentOrder
  An order generated by the service provider for the goods or services purchased by a customer. The object is used to exchange order data between central fulfillment and participating applications other than provisioning.
- Communications ProvisioningOrder
  An order generated by the product or service provider for the goods or services purchased by a customer. The object is used to exchange order data between central fulfillment and provisioning applications.
- Communications CreditAlert
  A credit alert is raised to send updates on collection actions between CRM to BRM.
- Communications ServiceUsage

The Service Usage object is used for querying or sending billed usage information for a service.

- Communications TroubleTicket
  A Trouble Ticket is used for sending order failure information from the central fulfillment systems to the trouble ticketing application in the context of order fallout.

These EBOs are specifically designed for Communications processes and data structure in order to accelerate the customer's integration efforts. They include over 200 attributes allowing the customers to integrate their Oracle Order Management, Billing, and Customer Care systems to many different systems within their IT environment, such as a financials system, or network operations system.

The Oracle AIA **Foundation Pack Extension for Insurance** enables insurance companies to accelerate application integration, especially between claims and financial systems in order to process claim payments, set up claim reserves, and pursue outstanding debt. It also facilitates integration with policy administration and other systems, whether it's Oracle or a third-party. Oracle AIA Foundation Pack Extension for Insurance helps dramatically minimize costs and risks for insurance companies by speeding time-to-value. The pre-built insurance claims business objects and services and insurance-specific enhancements to horizontal objects and services, enable companies to develop more flexible integrations that increase business and IT efficiencies, and accelerate innovation. With Oracle AIA Foundation Pack Extension for Insurance, insurance organizations can easily adapt to changing business needs through optimized business operations using documented insurance business processes and simplified upgrades via common objects and services.

Utilities customers can easily leverage Oracle AIA **Foundation Pack Extension for Utilities** to facilitate integration design and enable composite business processes including Concept to Launch, Order to Bill, Meter to Cash, and Customer Self Service. These processes span a variety of applications, including CRM for sales, marketing, and service; customer care and contact center; customer eBilling, ePayment and eSupport; meter data management; rating, billing and collections; and financials ERP.

The industry best practices are taken into consideration so that there is compliance to both common and industry specific standards which is required for interoperability between heterogeneous applications. The Enterprise Business Objects (EBOs) in the Foundation Pack Industry Extensions can either be industry specific (e.g. only for Utilities) or can be industry extensions of Core EBOs. The architecture allows for incorporating industry specific attributes as overlays. An industry specific object can be created by assembling together a set of business components available at the core with a set of industry specific components.

AIA in Business Flows

AIA is a complete integration solution for orchestrating agile, user-centric business processes across enterprise applications. Applications Integration Architecture is responsible for providing industry-specific solutions for composite business processes leveraging the various software assets that are available in the Oracle portfolio. Most of these solutions encompass orchestrated process flows, as well as pre-built data integration scenarios that are meant to seamlessly connect the systems which were not designed to work together.

AIA offers prebuilt solutions at the data, process, and UI level delivering a more complete process solution to business end users. All AIA components are designed to work together in a mix and match fashion and are built for configurability, ultimately lowering the cost and burden on IT of building, extending, and maintaining integrations. AIA products include:

- **Foundation Pack:** Business process composition framework that provides the architectural and programming model, best practices, utilities, and application independent Enterprise Business Objects and Services on which AIA users can develop standardized, cross-application process integrations.
- **Process Integration Packs (PIPs):** Pre-built, packaged integrations to accelerate the delivery of composite business processes across both Oracle and non-Oracle applications.
- **Direct Integrations:** Are used for bulk processing with AIA. Direct Integrations are used where it is appropriate to augment and support SOA integrations, such as ODI for high volume batch data

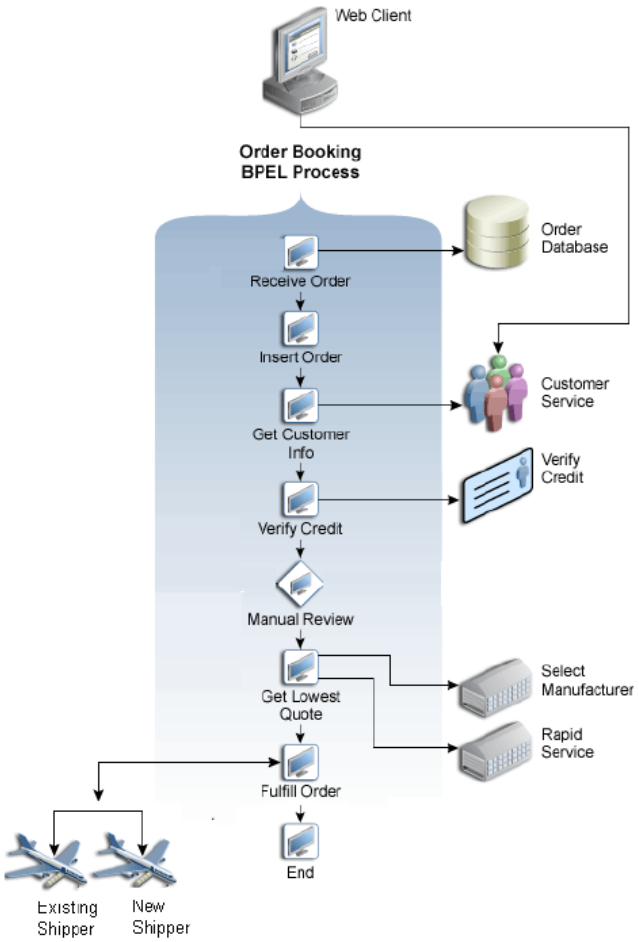
There are numerous features/advantages of Oracle Application Integration Architecture. Some of them are listed below:

- Defines integration architecture by adopting a service-oriented architecture.
- 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 in multi-interaction styles.
- Adopts an applications independent canonical data model to accomplish the decoupling of data format.

Sample Use Case

It is quite easy to discuss terms in abstract, but things become simpler when a use case is used to discuss how a solution is designed and deployed. We will use the following sample use case throughout this paper to illustrate how you can use the Oracle Application Integration Architecture Foundation Pack to build your own custom integrations to bring together a number of disparate applications, both internal and external, in a service-oriented way.

In this sample application, Global Company is a retail storefront for ordering electronic devices through a web-based client application. This diagram outlines the company's business process for booking and fulfilling orders:



This flow diagram illustrates the process when a new order is placed:

1. The web client sends a message to a JMS Queue which then routes the message to any service that has registered an interest in these messages. In this case, it is the order booking process.
2. This process sets the order to *Pending*, and writes the order to the order database tables. The process then calls the customer service system to retrieve customer ID, name, address, and credit card information, and checks the identified customer against the credit service to verify if the customer's credit card is valid. The credit service returns the relevant rating for the customer.
3. If credit is not approved, the process cancels the order.
4. If the order is approved, it is sent to two suppliers for their price quotes. The process collects the quotes and selects the supplier quoting the lowest price. The process then invokes the fulfillment service which selects the appropriate fulfillment provider based on a business rule.
5. After the order is fulfilled, the BPEL process sets the order to *Complete*, and starts a notification service, which sends the customer an email with the purchase order information.

In the sample use case, the online portal application only has the responsibility to post the sales order message in the common AIA Enterprise Business Object format without worrying about how the order is represented in any of the other end systems. Some of the business objects that come in play in the use case are Sales Order, Purchase Order, Item, Invoice, etc.

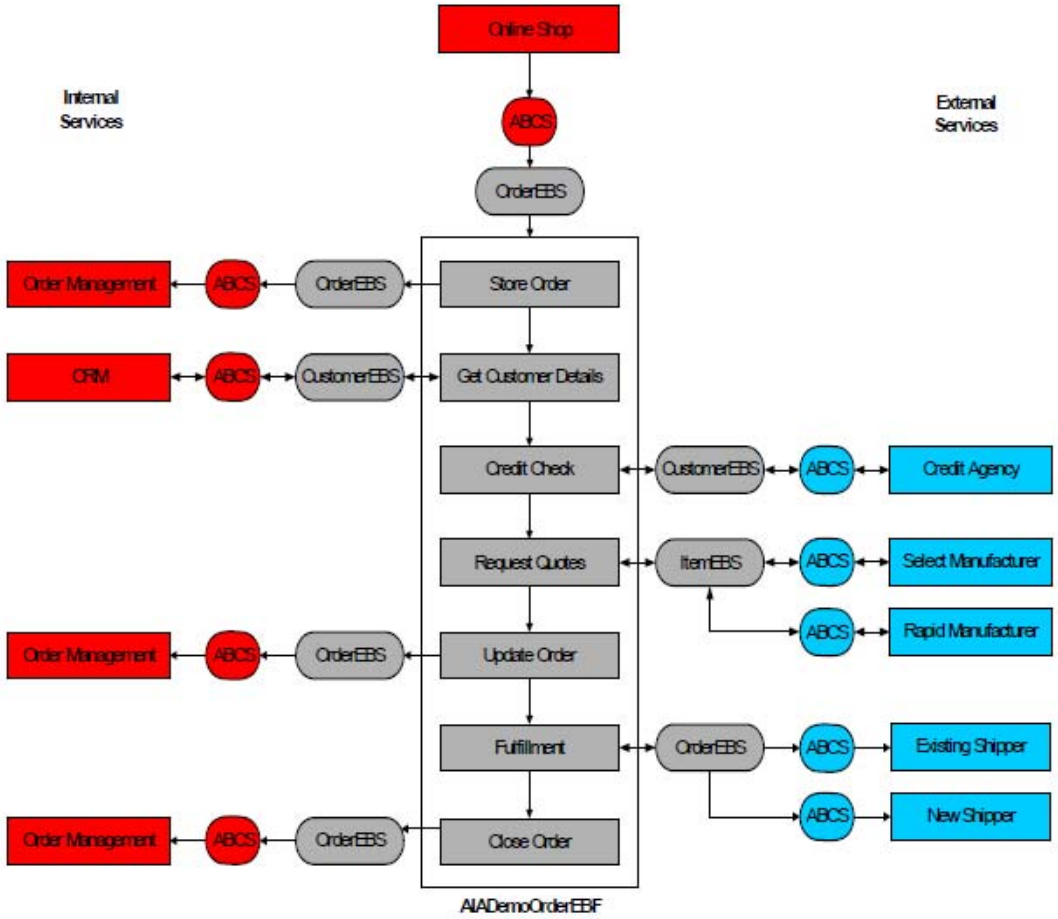
To understand the value of Application Integration Architecture, it's important to look at the key principles associated with Service Oriented Architecture (SOA). The following are the advantages of going to SOA:

- Standards – Compliance to both common and industry specific standards is required for interoperability between heterogeneous applications.
- Abstraction – Reusable, modular coarse-grained business services provide rapid composite application development and easier maintenance.
- Loosely coupled – Minimal dependencies with other applications offer durability and agility.

AIA is designed and developed on these first principles of SOA. Let's take our order booking sample use case and see how that business process translates into SOA based AIA.

Application Integration Architecture Schematic of Sample Use Case

This diagram shows a technical representation of the order booking sample use case implemented using the AIA Foundation Pack:



The use case will be referred to in the creation of EBO, EBM and EBS along with extensibility.

EBO Concepts

EBO, EBS and EBM Basics

The term Enterprise Business Object (EBO) refers to a data model consisting of standard business data object definitions and reusable components representing a business object such as Sales Order, Party, Item, Customer etc. It is the best-in-class definition of the business data, rationalized across Oracle's entire application portfolio and industry standards. The Enterprise Business Objects are delivered as XML Schema Definition (XSD) files.

Enterprise Business Messages (EBMs) are the messages that are exchanged between two applications or devices. For every EBO, there will be two schema modules – one containing the definition of the EBO and another containing the definition of the operations that need to be performed on that EBO (e.g. Read or Update operation).
Enterprise Business Services (EBSs) are basically coarse-grained business service interfaces for performing a business task.

EBS Features

- Application agnostic interfaces
- Services implemented by applications or cross-functional business processes
- Standardized Enterprise Business Message as payload
    - Noun – Enterprise Business Object
    - Verb - service operation
    - Message Header – message meta data
- Transport agnostic
- SOAP, HTTP, HTTPS, JMS
- Allows for service substitution

Business Objects

Enterprise Business Objects

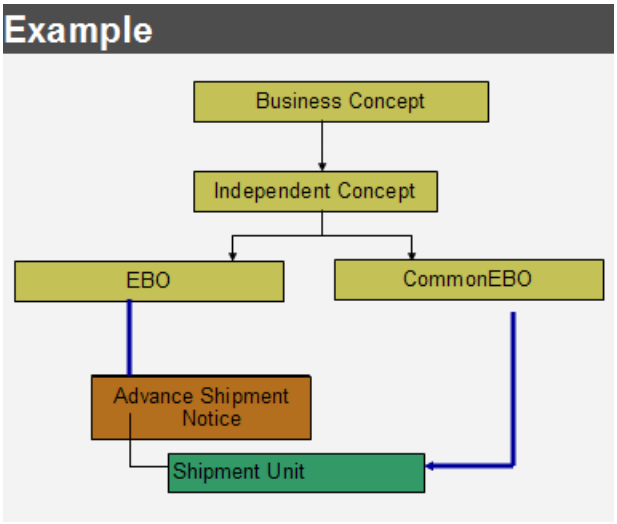
EBOs are enterprise business objects based on the canonical object model. These, along with Enterprise Business Services (EBSs), act as the cornerstones of AIA architecture.
The EBO defines a set of generic data structures that support cross-application business processes and data integration packs, independent of the source or target applications. The model is a composite of Application Business Objects (ABOs) and industry standards and eliminates the need to map data directly from one application to another. EBOs contain reusable components that satisfy the requirements of business objects from the target application data models.

Reusable EBO (Common EBO)

The term EBO refers to a data model consisting of standard business data object definitions and reusable data components representing a business object such as Sales Order, Party, Item, Customer etc.

The Common EBO is an independent business concept very much like an EBO but it may be contained (in its entirety) within the definition of another EBO and can be shared across multiple EBOs. For example, Item is a Common EBO because it is part of the ItemComposition EBO.

The following diagram illustrates a Common EBO:



The figure illustrates that the AdvanceShipmentNotice EBO uses the ShipmentUnit Common EBO in its definition.

Common Components

Shared Components

Shared components are used for containing components and business data types that are applicable to all EBOs and EBMs. Examples of shared components are:

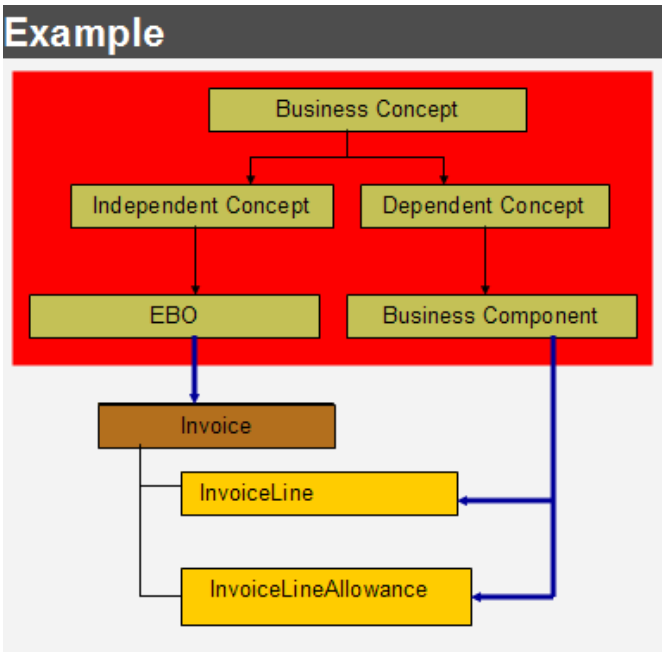
- Address
- Note
- UnitPrice
- Attachment
- CreditCard

The benefits of using shared components are that they promote reusability and that the common schema module is imported into each of the EBO schema modules.

Business Components

Business Components are business concepts that are specific to a business object but have no independent existence.

A Business Component is always a dependent concept that exists in the context of an EBO. It does not have any existence independently.



In this example, Invoice is an EBO which can exist independently whereas InvoiceLine and InvoiceLineAllowance are dependent concepts (Business Components).

Reference Components

Reference Components act as a foreign key. However, more commonly used groups of attributes could be part of the reference structure. The key features of Reference Components are:

- Reference to other EBOs
- Equivalent to a foreign key
- Group of attributes could become part of the reference structure depending on the business need
- Share the namespace with Common Components

The benefit of using Reference Components is that they reduce overhead because they can be easily associated to other EBOs.

Infrastructure Components

Various data types are used for designing the EBOs and common components. These data types could be simple types that are used to represent the "field level" data content such as DateType, NumericType, StringType etc., or complex types for groups of fields such as AmountType, QuantityType etc. The data types are compliant with CCTS guidelines.

The Infrastructure Components are comprised of meta data and code lists. Meta data are elements used in EBM header and body and code lists are metadata code lists which normally have a list of values e.g., for language, query operator, debug level etc.
Wherever possible, the EBO is designed with reusable objects by using object types that are inherited by the different common objects in which they are referenced. Using Address definition typeAs an example, if the 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 will be defined in a Common

Components schema module. Business components that can be used across various context-specific definitions for a single EBO will be defined within the EBO schema module.

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. For example, an invoice might be used in three contexts – process, cancel, and update. In this situation, the context for processing, or creating, the invoice might warrant the presence of almost all of the elements present in the EBO. However, the context for canceling the invoice might need only the information necessary to identify the invoice instance that needs to be cancelled.
The context-specific EBO definitions are created by assembling a set of common components and EBO-specific business components. These context-specific EBO definitions alone will be used in the appropriate context-specific Enterprise Business Messages (EBMs). In this scenario, the process-specific invoice definition will be a part of the 'ProcessInvoice' EBM and the cancel-specific invoice definition will be a part of the 'CancelInvoice' EBM.

EBO in the Sample Use Case

Oracle Application Integration Architecture provides a prebuilt Enterprise Object Library that defines the best-in-class representation of business entities such as Sales Order, Purchase Order, Item, Invoice, etc. These definitions are rationalized across the entire Oracle application portfolio and industry standards.

This canonical definition of the object brings together disparate applications using a common language and drastically reduces the number of transformations required to integrate between different applications. In our sample use case, the online portal application now only has the responsibility to post the sales order message in the common AIA Enterprise Business Object format without worrying about how the order is represented in any of the other end systems, which are likely to change and evolve over time.

One of the most common challenges in application interoperability is inconsistent business semantics among the different applications. Let's use the Sales Order business document from our sample use case. The definition of the Sales Order object in the order capture system, the online portal in this case, can be different from the way it is defined in the two fulfillment systems or the suppliers who may be using third party applications. The business objects can not only differ in terms of content but also in the naming rules and the meaning of the attributes in each of their systems. As the number of applications that are involved in the integration grows, this complexity grows exponentially.
In the use case, the SalesOrder EBO is one of the Enterprise Business Objects that needs to be created for the integration. Before creation of the EBO, it needs to be defined in such a way that it becomes application agnostic. In simple terms, the object definition of the SalesOrder EBO should be such that it is understood by any application, be it Oracle EBusiness Suite or Siebel.

There are various guidelines to follow for identifying a candidate EBO (the SalesOrder EBO in this case). For specific process integrations between different applications, the EBO identifying guidelines are:

1) It should be based on a business process
2) It should be aligned with Fusion Applications
3) It must define a unique business concept
4) The EBO name should be unambiguous across business process and industry

In business terminology, a sales order is an order generated by the product or service provider for the goods or services purchased by a customer. As an analogy in the AIA world, the Sales Order EBO is the document that is passed from the Order Capture systems to backend systems for order fulfillment. It's also the document used by downstream systems to query order status information

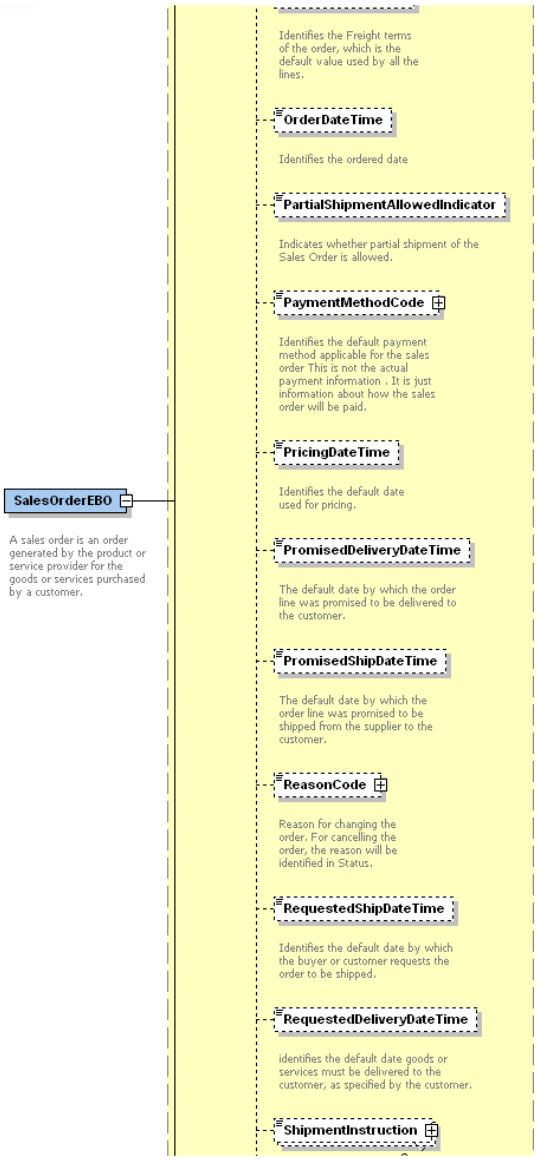
Before developing the SalesOrder EBO, it should be defined so that it has a structure which supports all application flows. The SalesOrder EBO should have the following characteristics:

- Represents business concept of a sales order
- Defined using inputs from multiple applications and content standards
- Common service payload used by all applications
- Designed for extensibility

After the EBO has been identified, the process of its development starts. The reference structure or its constituent elements and attributes are captured if it has an identity and existence in the reference application. There is a need for a reference application as it provides the starting point. This reference structure is used by the all the participating applications (within Oracle along with industry standards) to add their own unique attributes. This is done in the object mapping sheet which is then collated to form the final Object Sheet. In summary, the following activities are involved:

- Start gathering client (XYZ Inc.) artifacts
- Get the reference structure of the object from the reference application (e.g. Fusion Application)
- If the reference application does not have the object, then get the structure from any of the participating applications.
- First stage of mappings – map the levels (e.g. order lines, order schedules)
- Do mapping at the component level before attempting attribute level
- Involve the business users and start detailed attribute mappings

After development the SalesOrder EBO will look like the following diagram:


Identifies the Freight terms of the order, which is the default value used by all the lines.
OrderDateTime
Identifies the ordered date
PartialShipmentAllowedIndicator
Indicates whether partial shipment of the Sales Order is allowed.
PaymentMethodCode
Identifies the default payment method applicable for the sales order This is not the actual payment information . It is just information about how the sales order will be paid.
PricingDateTime
Identifies the default date used for pricing.
SalesOrderEBO
A sales order is an order generated by the product or service provider for the goods or services purchased by a customer.
PromisedDeliveryDateTime
The default date by which the order line was promised to be delivered to the customer.
PromisedShipDateTime
The default date by which the order line was promised to be shipped from the supplier to the customer.
ReasonCode
Reason for changing the order. For cancelling the order, the reason will be identified in Status.
RequestedShipDateTime
Identifies the default date by which the buyer or customer requests the order to be shipped.
RequestedDeliveryDateTime
identifies the default date goods or services must be delivered to the customer, as specified by the customer.
ShipmentInstruction

EBM Concepts

EBM Basics

The Enterprise Business Message (EBM) defines the elements that are used as messages in service operations. The EBM payload is a restricted view of the EBO content and it includes only the content necessary to perform the specific operation. The Web Services accept EBMs as input and return EBMs as output.

EBM Features

- Application-agnostic encapsulation of an EBO
- Generally coarse-grained that operates either on the whole EBO or a subset
- Payload of a web service operation in an EBS
- Semantically precise - performs a specific action i.e. one EBM for one Verb (Operation)
- A CRUD (create, read/query, update and delete) operation or a special operation
- Comprised of a header, verb and an EBO
- Transport protocol agnostic (e.g. SOAP, HTTP, HTTPS, JMS)

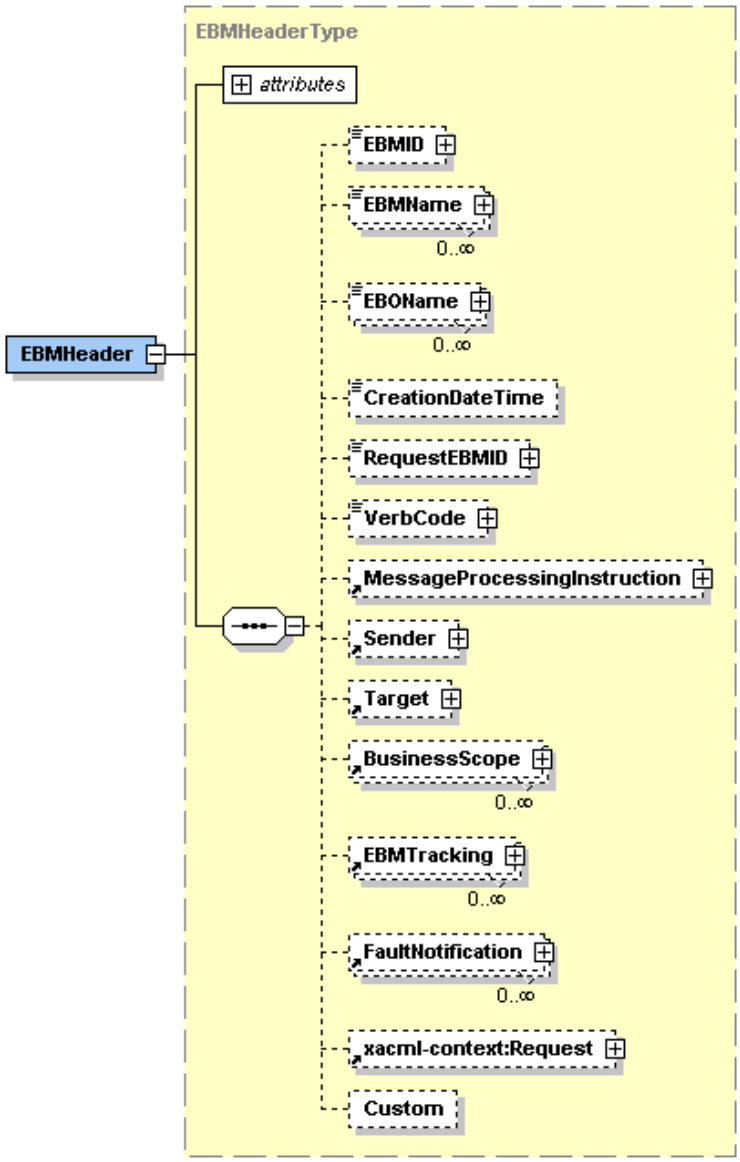
Sample EBMs

- CreateSalesOrder
- ProcessSalesOrderATPCheck
- ValidateSalesOrder
- QueryInvoice
- UpdateInvoice
- DeletePurchaseOrder

The EBM header carries information that can be used for (but not limited to):

- Tracking important information
- Auditing for business and legal purposes
- Indicating source and target systems
- Error handling and tracing

This figure illustrates the EBM header with its various attributes:



The EBM Data Area contains specific content of the EBO required for an operation. The advantages of having an EBM are:

- Selective content restriction by operation - e.g. Create SalesOrder may require the entire EBO, but Cancel SalesOrder only requires the SalesOrder number
- EBM XML schema module defines all operations available for a given EBO
- Enables trace and audit

There are certain standard/critical elements in the EBM Header which are needed. These are:

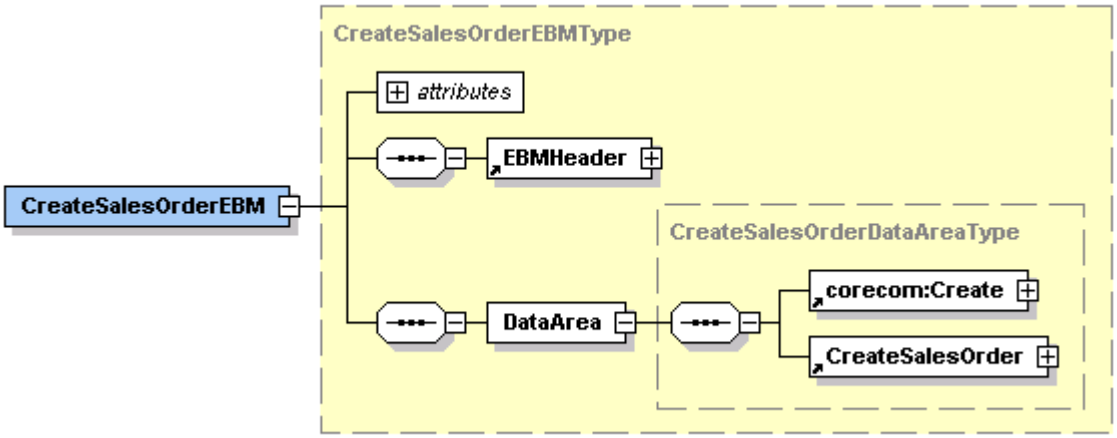
- EBMID
- EBOName
- Version

- SenderSystem
- TargetSystem
- ProcessInfo
- ReferenceID
- CreationDateTime

Message Assembly- Payload

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. EBM is the input/ output for the operations supported by an EBO.
This diagram illustrates that the EBM is comprised of:

- Data area
- Action / verb
- EBM Header (message meta data)



The EBM structure is such that there is one EBM for every combination of EBO and the action, and that an EBM can support only one verb. The combination of *get* and *update* in a single message not allowed.

The EBM header contains information used to process the message content, correlate request-response messages, apply security, and enable message tracking/ auditing. The Data Area identifies the operation to be performed by the EBM and contains the content payload. The standard operations that are defined in the EBM are:

- Create
  The Create verb indicates a request to create a business object using the information provided in the payload of the Create message. It is used in operations that are intended to create a new instance of a business object.
  Operations that use the Sync verb *must* create content and *should not* be an orchestration of other operations.
  The "Create" verb is not used for composite operations - it always involves the creation of one (or more in the case of "List") instance of a business object.

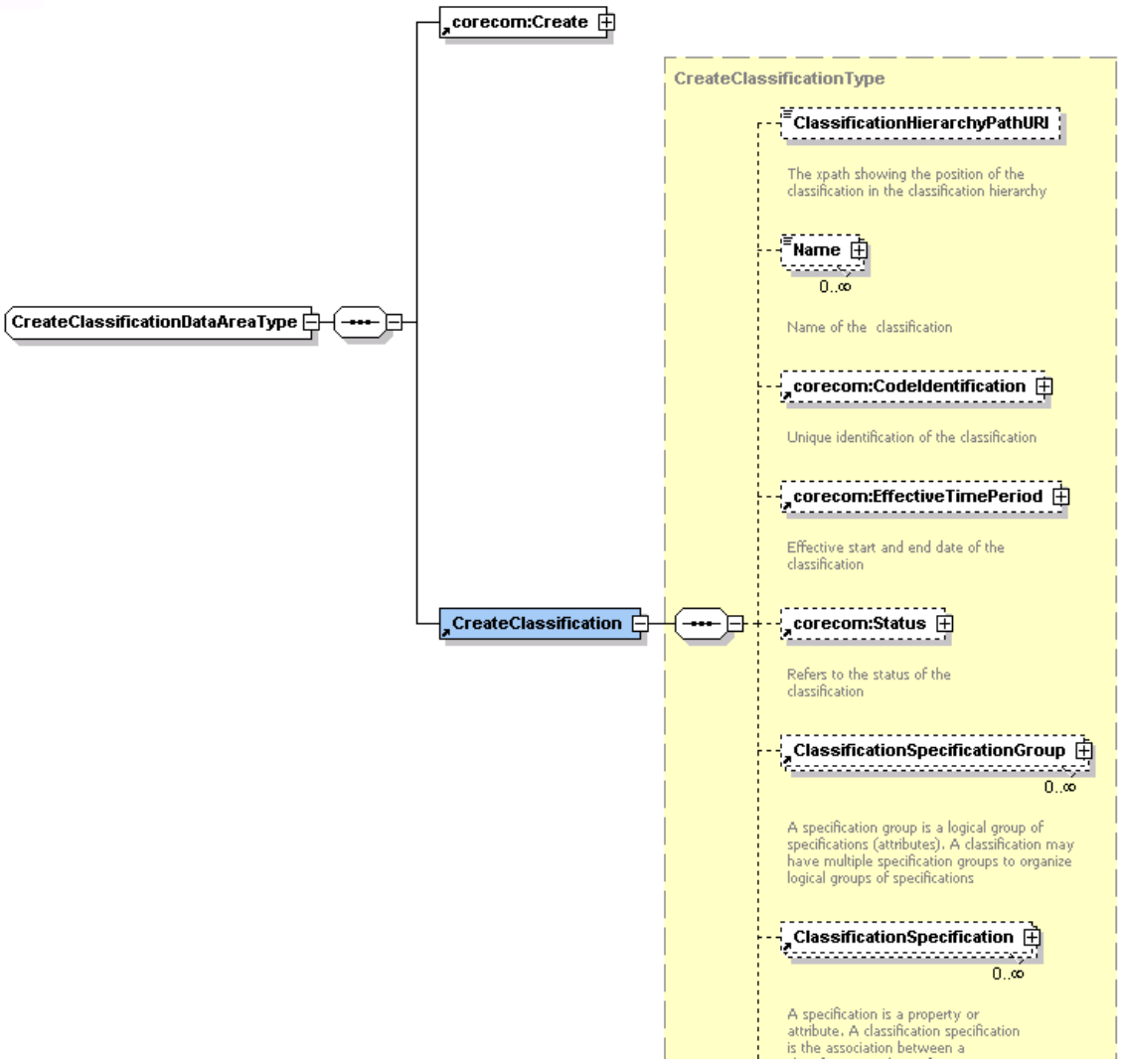
Generally speaking, a business process would invoke a create operation in cases where the source event that triggers the invocation is ***not*** the creation of the object in the requesting system. If both the service requester and service provider have the same object, then Sync would be a more appropriate verb to use. However, usage of Sync would also be conditional to the service provider having an implementation of a "Sync" operation.
A typical usage of a "Create" operation is a front end customer management system that could take service requests from customers and based on the information provided, request a work order system to create a work order for servicing the customer.

- Query/Read
  The Query verb is a request to a service provider to execute a query on a business object using the content provided in the payload of the Query message, and return the query result using the corresponding response message associated with the query. The requestor has the ability to optionally request a specific subset of the response payload.

- Update
  The Update verb indicates a request to a service provider to update an object using the payload provided in the Update message. It is used in operations that are intended for updating one or more properties of a business object or array of business objects.
  Operations that use the Update verb *must* create/ update content and *should not* be an orchestration of other operations.
  Similar to Create, a business process would invoke an "Update" operation mainly in cases where the source event that triggers the invocation is ***not*** the updating of the object in the requesting systemIf both the service requester and service provider have the same object, then Sync would be a more appropriate verb to use. However, usage of Sync is conditional to the service provider having an implementation of a "Sync" operation.
  An example of an "Update" operation is a receiving system updating a Purchase Order line with the quantity of items received, or an order capture system updating the customer record with customer address/ other information.

- Delete
  The Delete verb is a request to a service provider to delete the business object identified using the Object Identification provided in the payload of the Delete message. It is used for operations that are intended to delete a business object.
  Operations that use the Delete verb *must* delete content and *should not* be an orchestration of other operations

- Synchronization
  The Sync verb indicates a request to a service provider to synchronize information about an object using the payload provided in the Sync message. It is used where applications provide operations that have the ability to accept the payload of the operation and create/ update business objects as necessary
  The common CRUD operations are supported by most EBOs. Additionally some EBOs support more specific business operations



This figure illustrates the data area of an EBM containing the "Verb" and the operation specific "payload".

AIA makes an explicit distinction between operations that can process a single instance of a payload vs. operations that can process multiple instances of a payload. Distinct operations are provided for both cases. Only the "standard" operations have this distinction implemented.
This distinction is made within the payload structure (single instance of data area vs. multiple instances of data area) and also by adding a suffix "List" to the payload root element and the content root element. For example, CreatePurchaseOrderListEBM is a payload root element that has a CreatePurchaseOrderList content root element.

Special Operations

The common CRUD operations are supported by most EBOs but there might be some specific business activities that the CRUD operations cannot handle. These specific business activities can be defined as Special Operations. This is done by extending the Enterprise Business Message (EBM). The EBM for special operations should have specific content of EBO. For example, the

ApproveServiceRequest operation might only require an identifier for its execution. The identifier can be a complex type. The message structure of special operations can have additional content apart from the ones defined in the EBO.

Verbs in Use

An Enterprise Business Message (EBM) represents a message that is used in collaboration. An EBM encompasses one verb, one or more nouns (EBOs) of the same type, and the EBM header. Each of the service requests and the response is represented in an EBM by using a distinct combination of a verb and a noun. For example, a single QueryAccount EBM business document sends the request to a billing system for retrieving account details for one or several accounts. This can be accomplished by using a single 'Query' verb and several Account nouns. The billing application can respond to this request by sending a response message which is populated with details. The EBM cannot process details about more than one type of noun or verb. For example, it is not possible to have an Invoice and Order noun in the same message. Similarly, the presence of a Query and Update verb in the same message is not allowed.

The verb in the EBM identifies the action that the sender or the requester application wants the receiver/provider application to perform on the EBM. The verb also stores additional information pertaining to the action that needs to be carried out on the noun.

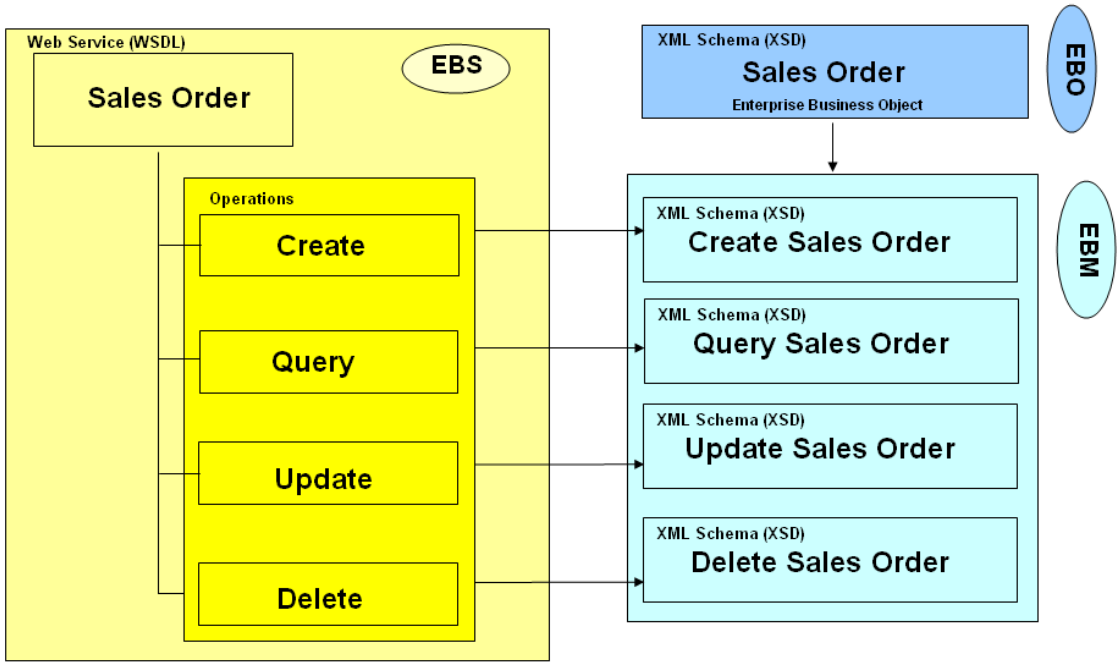
For each of the verbs that can be used with an Enterprise Business Object (EBO), an Application Business Connector Service (ABCS) must be defined by the participating requestor application and another ABCS must be defined by the service provider application.

Although a single ABCS can be used to handle multiple verbs, it is highly recommended to allow only a single ABCS per verb. This approach greatly reduces the complexity of designing a generic ABCS. If a single ABCS is designed to handle multiple verbs, it should be keep in mind that it needs to have the verb information accepted as a part of the input. This allows the ABCS to decipher the actual action to be performed and enables it to perform the appropriate transformations and invocations. In addition, allowing a single client side ABCS to handle multiple verbs means that a single client-side application specific ABO will encompass all of the information pertaining to all of the verbs.

The verb would be the same in the case of a synchronous request/response pattern. In the case of a delayed response pattern, the execution of the request will be implemented by one verb and the execution of the response will be implemented by another verb.

EBM in Sample Use Case

A SalesOrder Enterprise Business Object is a logical representation of the sales order business entity. An Enterprise Business Message on the other hand is the implementation of an Enterprise Business Object. In this case the SalesOrder EBM would be the implementation of the SalesOrder EBM. The Enterprise Business Message is designed to be operation specific so that there is no overhead of passing the entire Enterprise Business Object to every service operation. For example, the CreateOrder service operation requires more attributes to be passed in the Sales Order EBM than a DeleteOrder operation which may require only a unique identifier. The EBM also contains an EBM header which has additional attributes that are used to provide robust auditing and exception management. Any EBO by default supports the CRUD operations. This figure shows the relationship between the EBO, EBM and EBS.



When the CRUD operations on an EBO are not sufficient to support a specific business process, special operations might be required. This is done by extending the Enterprise Business Service (EBS). The following steps are only required if the Enterprise Business Service requires a new operation. This step is to:

- Create a new WSDL with new operations – XYZSalesOrderEBS.wsdl
  - For new operations – define new message types (use one message type per operation as per the WS-I requirements)

```
<portType name="XYZSalesOrderEBS">
        <documentation>
            <svcdoc:Interface>
                <svcdoc:Description>This interface contains operations for the Request-Response and Request-Only patterns</svcdoc:Description>
                <svcdoc:DisplayName>RequestSalesOrder EBS Interface</svcdoc:DisplayName>
                <svcdoc:LifecycleStatus>Active</svcdoc:LifecycleStatus>
            </svcdoc:Interface>
        </documentation>
        <!-- operation support for creation -->
        <operation name="CreateSalesOrderListSync">
            <documentation>
```

The above definition shows the steps to create a new operation called "CreateSalesOrderListSync"

EBS Concepts

EBS Basics

Enterprise Business Services (EBSs) are the foundation blocks in AIA. EBS represents the application independent web service definition for performing a business task. It is self-contained, that is, it can be used independent of any other services. In addition, it can also be used within another EBS. EBS's are standard business level interfaces 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 an EBS has a well-defined interface described via XML. This interface description is composed of all details required for a client to independently invoke the service.

Standard Message Definition

Enterprise Business Services are first-class objects within the Enterprise Service Bus (ESB). The 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. EBS's are generally coarse-grained and typically perform a specific business activity. For example, the activity performed could be one of the following:

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

Each EBS has a well-defined interface described via XML. This interface description is composed of all details required for a client, such as a BPEL process, application business connector (ABC) service, or Fusion application, to independently invoke the service. Such details include protocol bindings and transport details. These interfaces are described using an implementation-agnostic approach. Hence, these interfaces are participating-application agnostic.

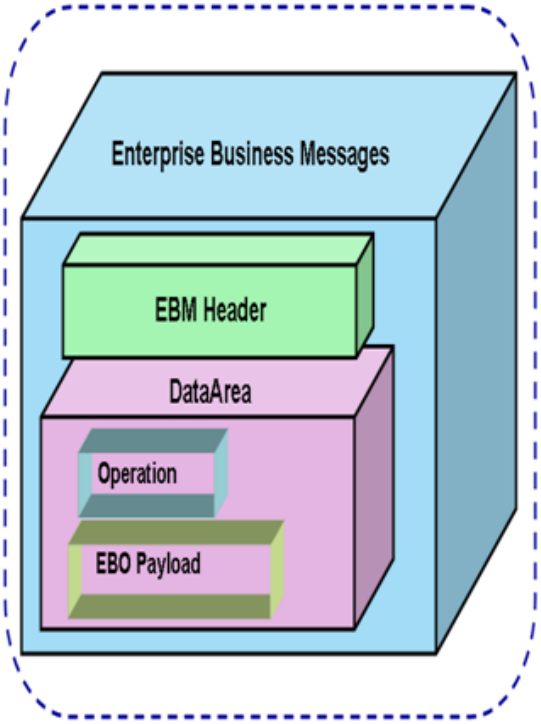
Operation Naming

The default operations that are supported by the EBS are the CRUD (Create, Read/Query, Update and Delete). If an operation cannot be represented by the CRUD, then a specialized operation is needed.

Payload Definition

As shown in this figure, the EBS has a standardized EBM as payload which consists of the following:

- Noun – Enterprise Business Object
- Verb - Service Operation
- Message Header – message meta data



Interaction Patterns

Interaction patterns that are used in AIA include:

- Request/response
- Notify/request-only (fire-and-forget)
- Delayed response

Request/Response

In this interaction pattern the client-side participating application makes a request to the ABCS and receives the response from it. In this scenario, the client-side ABCS has two transformations – one for each direction:

- Transformation of the request details available as an Application Business Object (ABO) into an EBM that can be handed over to the Enterprise Business Service (EBS).
- Transformation of the response details available in an EBM into ABO that can be returned to the participating application as response.

The server-side ABCS also has two transformations – one for each direction:

- Transformation of the request details available as an EBM into an application-specific business object that can be handed over to the application implementing the service.
- Transformation of the response details available in ABOs into an EBM that can be returned to the EBS as a response.

Notify/Request-Only (Fire-and-Forget)

In this pattern the client-side participating application makes a request to the ABCS and does not expect to receive any response. In this scenario, the client-side ABC service will have one transformation:

- Transformation of the request or notification details available as an Application Business Object (ABO) into an EBM that can be handed over to the Enterprise Business Service (EBS).

Similar to a client-side ABCS, the server-side ABCS has only one transformation. It receives the request from the EBS as an EBM. The ABCS needs to pass the request to the application implementing the service. The transformation occurs as follows:

- Transformation of the request details available as an EBM into an application-specific business object that can be handed over to application implementing the service.

Delayed Response

In this scenario, the client makes a request to the participating application and the participating application sends a response back to the calling application at a later point in time. This is the callback usage scenario. This can be visualized as a composition of two synchronous request / response usage scenarios, one initiated by the consumer and the other by the producer. In the case of a delayed response or subscription pattern, the consumer ABCS will have only one transformation:

- Transformation of the response or subscription details available as an EBM into an Application Business Object (ABO) that can be handed over to a participating application.

In the case of fire-and-forget as well as asynchronous request / response interaction patterns, AIA leverages queue messaging. The architecture mandates that the pre-built integrations interact using industry standard JMS interfaces.

EBS in Sample Use Case

The abstraction layer in AIA is provided by the EBS. This service has three key purposes:

- Hides service implementation complexity from the service requester. This way the service requesters can focus more on solving the business problem than dealing with integration challenges
- Exposes a service contract that is application-independent. This means any application can invoke the service in the same way irrespective of who the service provider application is.
- Ensures contract-first development which forces the organization to take a more strategic enterprise architecture approach than building a tactical point-to-point integration. It narrows the gap between IT and the business because the coarse-grained EBSis designed to meet business needs rather than be dictated by the complexity or nuances of the participating applications.

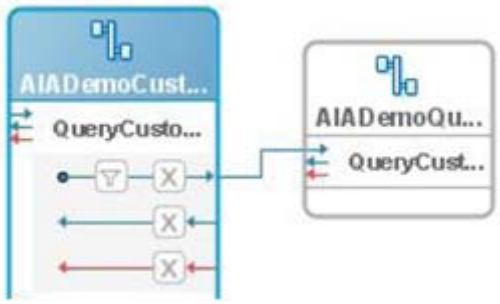
Standardized service definitions across applications are created so that they can be implemented by all Oracle applications. The advantages of having standardized service definitions are:

- A single service supports multiple operations – e.g. SalesOrder Service may support Create, Cancel, Update Operations
- Each operation uses SalesOrder EBO as standard input and/ or output
- Multiple applications may provide the same service e.g. EBS, Siebel, Enterprise and E1 can support Create Sales Order

- Objective is to be able to switch the service provider without affecting the service e.g. switch from EBS or E1 to Fusion (or any other partner applications that provide the same service
- Long term, all cross- pillar integrations – data synchronization or business process services to be implemented using standard services
- Standard services require standard payloads to be truly application independent
- EBOs are standardized representations of business objects that will serve as the payload (input or output) for standard services

In the architectural schema for the order booking sample use case every service invocation is done through the EBS. So when the process needs customer details, it invokes the AIADemoCustomerEBS service and the QueryCustomerParty operation in that service. Taking this approach ensures that if the legacy application that acts as a customer service provider today is upgraded to an enterprise CRM application tomorrow, the process doesn't have to change. The Enterprise Business Service interface remains the same. This enables customers to leverage their existing investments and evolve their IT systems at their own pace.

This is an example of how an Enterprise Business Service is implemented using the Foundation Pack WSDL interfaces:



The following table lists some of the EBOs and EBSs used in the sample use case:

Enterprise Business Service (Enterprise Business Object)	Operation	Enterprise Business Messages
AIADemoCustomerEBS (CustomerPartyEBO)	QueryCustomerParty	QueryCustomerPartyReqMsg, QueryCustomerPartyRespMsg
AIADemoOrderEBS (SalesOrderEBO)	CreateOrder	CreateOrderReqMsg
AIADemoItemEBS (ItemEBO)	QueryItem	QueryItemReqMsg, QueryItemRespMsg

Versioning

Versioning is the mechanism for content evolution within the library. All content is versioned. Versioning is distinguished between major and minor, with specific characteristics for each.

A *major* version defines a non-backward (breakage) compatible revision from a supporting software, data structure, or semantic processing perspective. These changes consist of, but are not limited to:

- Changing element, type, and attribute names.
- Changing the structures so as to break polymorphic processing capabilities.
- Deleting or adding mandatory elements or attributes.
- Removing or changing values in enumerations.

A *minor* version defines a backward-compatible (no breakage) revision from a supporting software, data structure, and semantic processing perspective. Multiple minor versions may exist within a major version.

It is imperative that backward compatibility within the minor version sequence be maintained as minor versions evolve. Each subsequent minor version must incorporate the previous minor version revisions. These changes consist of, but are not limited to:

- Adding optional elements or attributes.
- Adding values to enumerations.

The versioning mechanism comprises two distinct approaches. In both approaches, the major version is explicitly identifiable. The two approaches are:

- Group versioning, using directory names
- Individual versioning, using indicators

In both cases, however, major version numbers are always reflected within content name schemes, and minor version numbers are not reflected.

Group versioning using versioned directory names is used to accumulate major revision changes for groups of related content, appropriate for evolving together over time. Multiple versions of content using group versioning may exist within one release. This approach provides revision isolation so that changes that affect only specific groups of EBOs and the associated EBMs do not affect other groups of EBOs that do not require the changes. The directory name reflecting the major version of the group will be the same as the highest major version across the group content. Concerning minor versioning of this approach, the minor version changes are provided within the content of the group by overwriting the previous schema module. Group versioning within the library is used for:

- Core/Industry Common information
- Core/Industry Common EBOs
- Core/Industry Custom Common information
- Core/Industry Custom EBOs

Individual versioning consists of the major version and this approach allows revision of content on an individual basis. Using this approach, the minor version changes are provided within the content by simply overwriting the previous schema module.
Individual versioning within the library is used for:

- Core/Industry EBOs
- Core/Industry Custom EBOs

Extensibility

Implementations always have unique requirements, either specific to their business or specific to the industry. The ability to build extensions that are sustained and preserved across new releases and upgrades is one of the key features of the Oracle Application Integration Architecture. Every single AIA component can be extended. The following section illustrates the different extensions by using an order booking sample use case.

Extending an Enterprise Business Object

The EBOs are delivered as a set of XSD files. For every EBO, AIA also provides a custom XSD file in which all customer extensions are stored.

The SalesOrder Enterprise Business Object that is shipped by Oracle has Supplier Name and Supplier Price attributes defined at the line level. However, the third party system used by the company's supplier requires that these attributes be passed at the header level. To be able to integrate with those systems, the company's IT department will have to extend the Sales Order Enterprise Business Object to add these two new attributes at the header level.

To achieve this, we will extend the custom schema CustomSalesOrderEBO.xsd to add the additional attributes. As we want to add the attributes on the order header level, the following part of the schema definition needs to be changed:

```
<xsd:complexType name="CustomSalesOrderType"/>
```

After adding the attributes, this section of the schema definition looks like:

```
<xs:complexType name="CustomSalesOrderType">
<xs:sequence>
<xs:element name="SupplierName" type="xs:Name"/>
<xs:element name="SupplierPrice" type="xs:double"/>
</xs:sequence>
</xs:complexType>
```

The SalesOrderEBO is now ready to carry the custom attributes.

Note that the extension of the underlying SalesOrderEBO also extends all EBMs that reference the SalesOrderEBO. In this case, this is CreateOrderReqMsg. As the EBMs are also extended, the extended message definition also extends all EBS and ABCS that work with these EBMs.

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

It is also possible to have an EBO which is very industry-specific like Communications or Utilities.

References

1. White paper on Oracle AIA, Ravi Sankaran (available for Oracle.com/aia)
2. Accelerate your Enterprise SOA by Leveraging the AIA Foundation Pack, White paper by Sameer Phatarpekar (available from Oracle.com/aia)
3. Oracle AIA -Enterprise Object Library 2.2.1: Enterprise Business Objects and Messages XML Naming and Design Rules (available from Metalink)
4. OAGi OAGIS 9.0 Naming and Design Rules Standard version 0.7
5. UN/CEFACT Core Components Technical Specification Draft 2.2
6. UN/CEFACT XML Naming and Design Rules, Draft 2.0

ORACLE

EBO Concepts – Concepts, Structure, Terminologies and Design Rules
August 2009
Author: Pijush Gupta

Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

0109