Oracle® Application Integration Architecture - Foundation Pack 2.5: Getting Started with the Oracle® AIA Foundation Pack and Demo

Release 2.5

Part No. E15761-01

October 2009

ORACLE

Oracle Application Integration Architecture - Foundation Pack 2.5: Getting Started with the Oracle AIA Foundation Pack and Demo

Part No. E15761-01

Contents

Chapter 1: Executive Summary

This document discusses how Oracle Application Integration Architecture Foundation Pack offers great opportunities to build state-of-the-art service-oriented architecture (SOA) integrations. Oracle Application Integration Architecture Foundation Pack provides Oracle's best-practice methodology and reference architecture, which are based on a profound service-oriented foundation running on Oracle's best-in-class middleware suites.

This document is targeted mainly at integration architects who want to extend or adjust prebuilt integrations shipped by Oracle, or who plan to build new SOA-based integrations based on Application Integration Architecture. After reading this document, you should be able to understand the Application Integration Architecture approach and the Reference Architecture. We will demonstrate these concepts and components using the Application Integration Architecture Foundation Pack Demo. This demo is the Application Integration Architecture adaptation of the SOA Order Booking Demo, which many readers may already know.

Chapter 2: Introduction

"It is apparently easier to say that a firm will adopt SOA than it is to make specific plans and follow through on them."

Randy Heffner, "Planned SOA Usage Grows Faster Than Actual SOA Usage." Forrester, March 2007.

Many organizations plan to adopt service-oriented architectures (SOAs) to achieve more agile and flexible system landscapes. To substantially support them, Oracle offers customers and partners its own extensive experience with application integration and SOA adoption in the form of Application Integration Architecture for reuse in many application integration scenarios.

Oracle Application Integration Architecture delivers a comprehensive, service-oriented foundation for adaptive integration of applications. Oracle's development organization relies on this architecture to implement Process Integration Packs (PIPs) for a number of industry-specific, out-of-the-box integrations between specific applications in Oracle's portfolio. Oracle Application Integration Architecture also offers customers and partners multiple options to adjust existing integrations to unique business requirements or to even build new integrations. Regardless of the approach, the Application Integration Architecture Foundation Pack substantially supports organizations with a SOA methodology, a well-established common vocabulary, and a rich set of infrastructure components required for a successful SOA-based integration.

This document should be used along with the Application Integration Architecture Foundation Pack Demo to outline the Application Integration Architecture and the requirements for leveraging it.

Chapter 3: Application Integration Architecture Overview

Today, many organizations are planning to or are already starting to transform their IT landscapes into service-oriented architectures (SOAs) to meet growing business challenges that demand more and more flexibility and agility. They need to be able to respond faster than their competitors to new market demands and need more accurate and real-time information about their customers and their business processes.

However, experience shows that SOA adoption is no easy task. Organizations are facing several key issues. First, business processes are often poorly understood and do not follow best-practice industry approaches. Second, implementation requires a big set of technologies, while often no reference architecture or standard business objects are available to rely on. And finally, implementations often lack a mature methodology and a profound governance model to effectively design, operate, and maintain a SOA.

Oracle's Application Integration Architecture delivers many valuable features to master these challenges. First, Application Integration Architecture is a business-centric approach because it always starts with best-practice industry processes such as Order-to-Cash and Opportunity-to-Quote. From there, Oracle defines a common, application-agnostic vocabulary to uniquely standardize business objects and their relevant abstract operations.

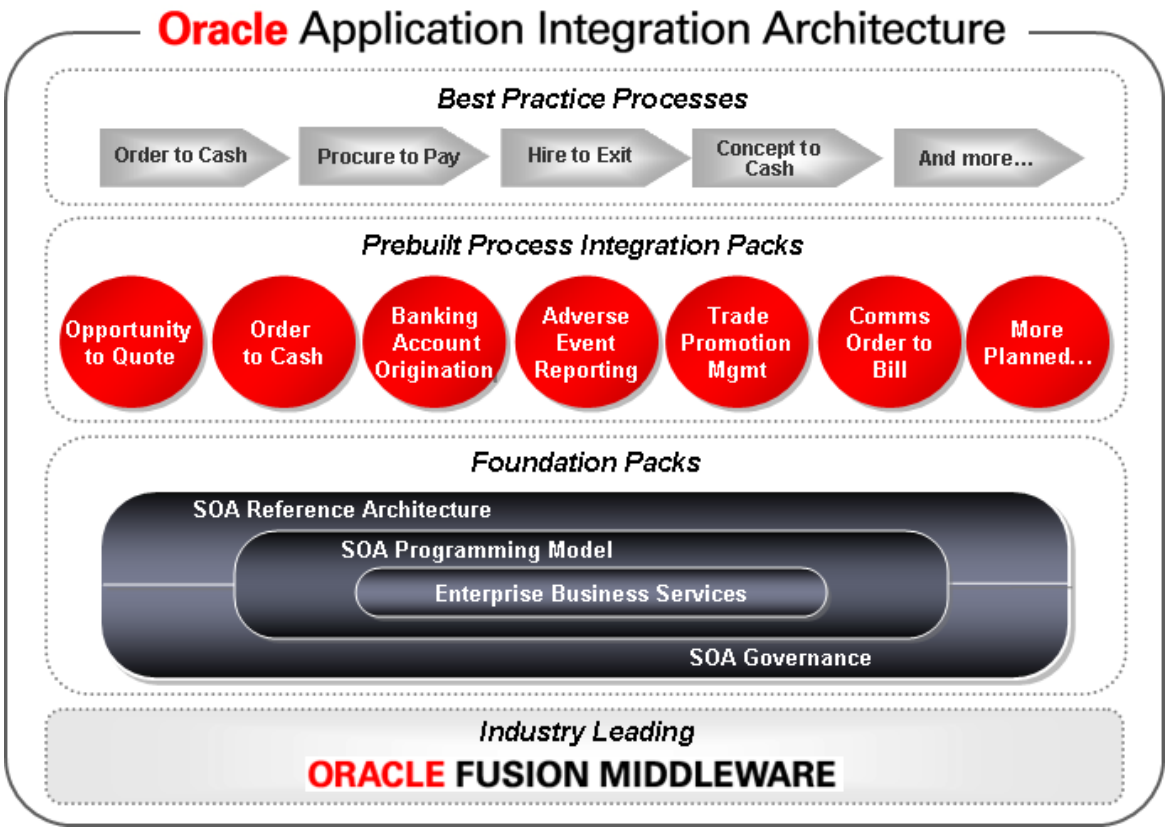

Figure 1: Application Integration Architecture overview

Relying on this foundation of standard industry processes and canonical data models, Oracle identifies Process Integration Packs (PIPs). These PIPs model specific business integration scenarios in the context of a certain industry, Order-to-Bill in the communications industry, for example. These PIPs also contain all required integration aspects in a specific application constellation to cover the business process. For Order-to-Bill, for example, this means the synchronization of customers, products, orders, and other entities between Oracle Siebel CRM and Oracle Billing and Revenue Management.

Application Integration Architecture defines a common vocabulary of common business entities and their corresponding services.

As stated previously, Application Integration Architecture defines a common vocabulary across applications and industries. Enterprise Business Objects (EBOs) are the key elements in this context. They canonically describe standard business entities such as an order or an invoice. Based on these generic business entities, Application Integration Architecture delivers other artifacts such as Enterprise Business Services (EBS), Enterprise Business Messages (EBM), Enterprise Business Flows (EBF), and Application Business Connector Services (ABCS). We will describe their purposes and relationships to each other later when we explain the Application Integration Architecture reference architecture. The combination of all of these SOA artifacts makes up the adaptable and extensible service-oriented reference architecture that Application Integration Architecture stands for.

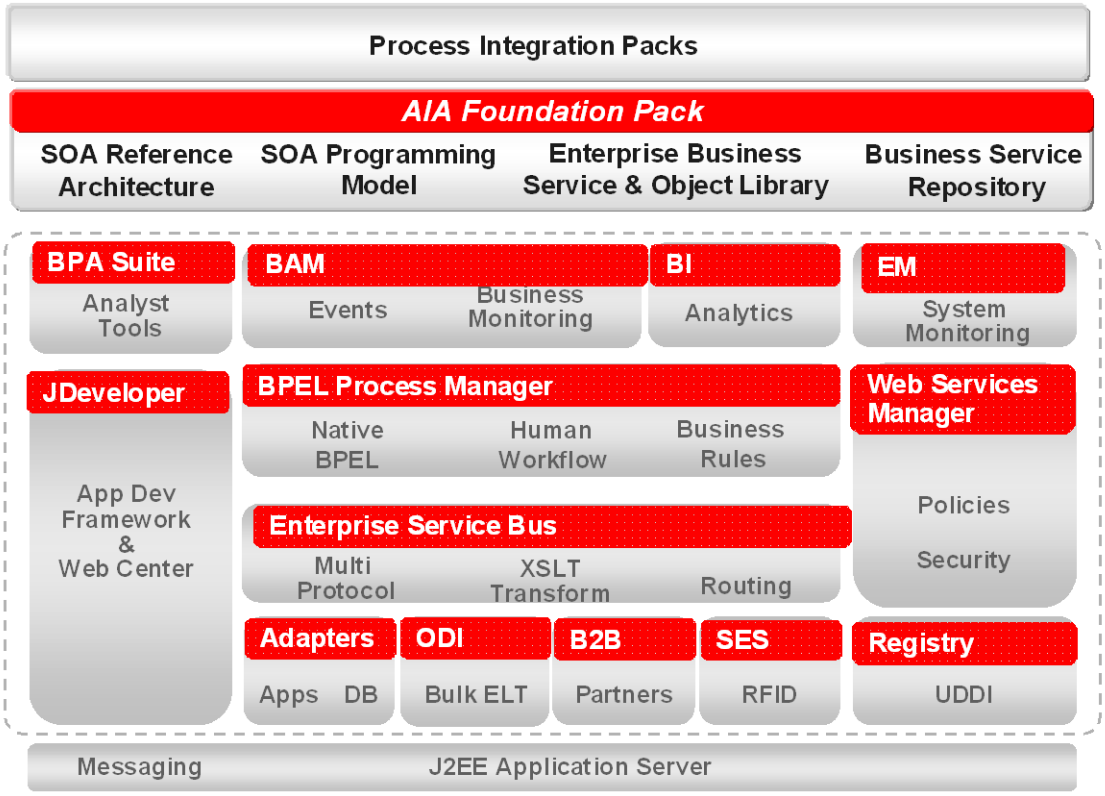

Figure 2: Application Integration Architecture runs on Oracle Fusion Middleware and Oracle Weblogic Server

The Application Integration Architecture infrastructure enables organizations to successfully manage their SOA.

Based on either Oracle Fusion Middleware or Oracle Weblogic Server, Application Integration Architecture provides several additional key infrastructure components that make up an even more powerful SOA stack:

- Installation and Deployment.

  Application Integration Architecture comes with an installation infrastructure that simplifies the deployment of Application Integration Architecture by providing a one-step installation procedure to deploy all Application Integration Architecture components.

- Business Service Repository (BSR).

  As a key component of SOA governance, Application Integration Architecture delivers a catalog of integration artifacts making it unique in the market. The BSR enables customers to understand these SOA integration components and their relationships, as well as manage their entire life cycles.

- Composite Application Validation System (CAVS).

  With CAVS, customers get a powerful tool to define and test single services as well as entire end-to-end flows that make up a particular integration scenario. By providing simulation capabilities, CAVS enables testing even when participating applications are not in place.

- Error Resolution and Logging.

  Oracle provides a unified error-handling and logging solution that allows the identification and resolution of issues across applications, across technologies such as BPEL or Enterprise Service Bus, and across integration patterns.

- Diagnostics.

  Diagnostics enable customers to validate the consistency of their Application Integration Architecture installation by testing against all parts of the Application Integration Architecture stack and participating applications.

All of these SOA artifacts and infrastructure components need a solid technical foundation on which to run. With Fusion Middleware and Oracle Weblogic Server, Oracle delivers the best-in-class middleware suites available today. The parts of the Oracle SOA Suite that are most relevant to Application Integration Architecture are the Enterprise Service Bus (ESB), BPEL Process Manager, Business Rules Engine, Business Activity Monitoring (BAM), Oracle Web Services Manager (OWSM), and Adapters to connect to virtually any application.

Application Integration Architecture Foundation Pack

The Application Integration Architecture Foundation Pack contains everything you need to build custom SOA integrations on a solid foundation.

The same foundation that Oracle uses to develop Process Integration Packs is now also publicly available as the Oracle Application Integration Architecture Foundation Pack. It provides customers and partners a powerful infrastructure to implement strategic SOA-based integrations without the need to reinvent SOA methodologies, common vocabularies, and key infrastructure components. The Application Integration Architecture Foundation Pack offers the flexibility needed to adapt to business changes rapidly and in a governed way.

The Application Integration Architecture Foundation Pack contains exactly what organizations need to develop and maintain SOA-based integrations:

- A proven service-oriented integration approach to implement agile and adaptable business processes.
- A robust, proven, and reliable reference architecture for SOA-based integrations.
- A ready-to-use set of common data models for frequently required business entities and their related services and messages.
- A rich set of additional infrastructure components to enable implementation, operation, maintenance, and governance of SOAs.
- Best-in-class middleware from Oracle to run the best-in-class SOA tools.

Application Integration Architecture Reference Architecture

A proven reference architecture is one of the most important elements of the Application Integration Architecture (AIA) Foundation Pack. It provides customers with very clear guidance and a mature architecture blueprint to build a best-practice SOA.

Enterprise Business Objects

You can think of EBOs as canonical, application-agnostic representations of frequently used business entities.

The architecture starts with the concept of Enterprise Business Objects (EBOs). EBOs can be considered as application-independent representations of key business entities. Examples of some of the EBOs included in the Application Integration Architecture Foundation Pack are:

- CustomerParty
- Item
- SalesOrder
- PriceList
- Invoice
- InstalledProduct

Developing the definition of such a common view of business data is always a major challenge in integration projects. Customers now can rely on Oracle's experience with thousands of application implementations when using these EBOs.

Enterprise Business Services

An EBS provides the generic operations that an EBO should have.

Enterprise Business Services (EBSs) are the centerpiece of the AIA Reference Architecture. They implement the required operations on EBOs in the right coarse-grained granularity that SOAs demand. For every EBO, Application Integration Architecture also ships generic service definitions to cover standard operations such as create, update, query, delete, and sync. Furthermore, an EBS may additionally provide several other business-specific operations such as GetAccountBalance for the CustomerPartyEBS. Only with these application-agnostic Enterprise Business Services can the loose-coupling of systems—a necessity in a true SOA—become reality.

Enterprise Business Messages

An EBM is the message format that is specific to the input or output of an EBS operation.

Enterprise Business Service operations require specific message formats called Enterprise Business Messages (EBMs) for service requests and responses. For instance, the request message for the QueryCustomerParty operation requires only a unique customer number. However, the response message needs to carry the entire CustomerPartyEBO structure as part of the message because this is what the caller of the service expects as the result when calling QueryCustomerParty.

Application Business Connector Services

An ABCS implements a particular service operation in the context of a specific application.

To enable applications to integrate into these generic, application-independent structures, you can implement Application Business Connector Services (ABCSs). Such an ABCS calls or is called by the appropriate EBS, depending on whether the application is in the requestor or provider role in a particular scenario. The main responsibilities of ABCSs include:

- Conversion between the generic EBO and the application-specific format.
- Cross-referencing of key attributes and message validation.
- Conversation with the specific application.

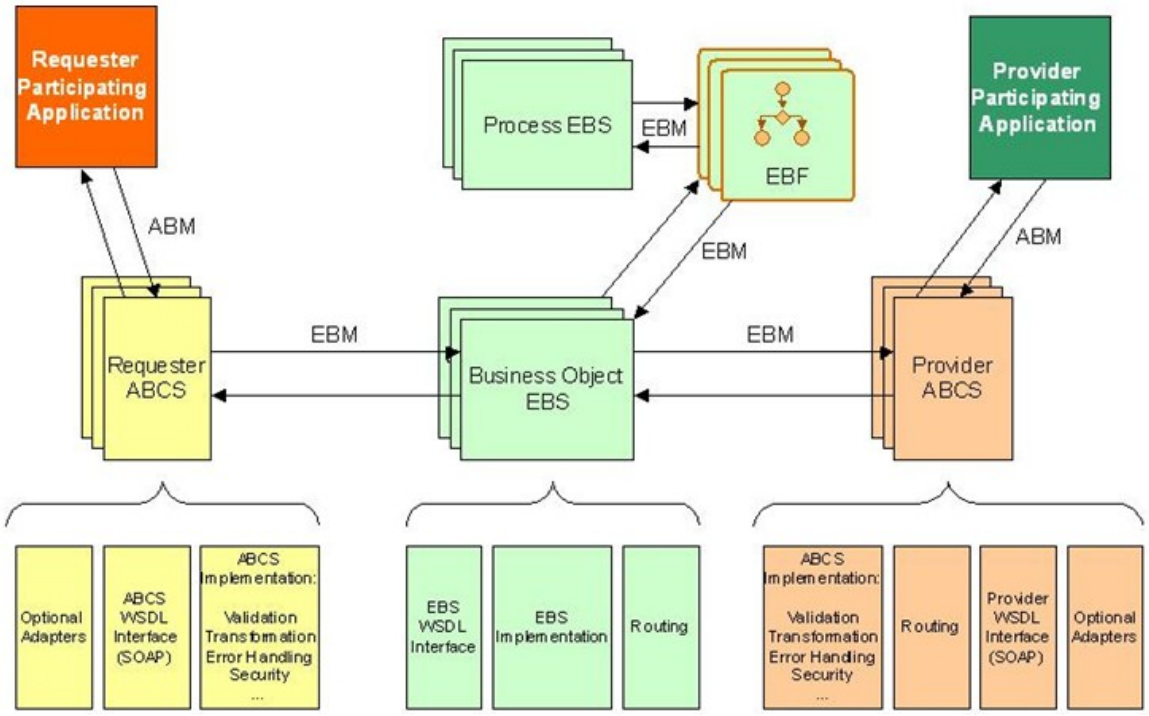

Figure 3: Application Integration Architecture Foundation Pack Reference Architecture

Figure 3 illustrates the way in which these components fit together. While the requesting application talks to the requester ABCS, the requester ABCS transforms the request into an EBM and initiates the EBS. The EBS now calls the provider ABCS to query the response from the provider application. The response finally flows back through the same services to the requester application. On the way back, transformation of the provider's application-specific format to the EBM and back to the requester's format must be performed.

Enterprise Business Flows

An EBF orchestrates a number of EBSs to implement a complex integration flow.

Sometimes, the integration scenario is not just a simple requester-provider relationship. In those cases, an Enterprise Business Flow (EBF) orchestrates any number of EBSs required to implement a specific business flow. The EBF completely controls the business logic and calls the appropriate EBS methods, such as QueryCustomerParty for the CustomerPartyEBS.

The following sections provide a description of the Application Integration Architecture Foundation Pack Demo. The demo covers a typical case in which an EBF orchestrates several EBSs, which are themselves wired to ABCSs to communicate with applications.

Chapter 4: Application Integration Architecture Foundation Pack Demo

The Application Integration Architecture Foundation Pack Demo is designed to present the most important components and approaches of Oracle Application Integration Architecture. It is based on the SOA Order Booking Demo, which was built to showcase particular aspects of the Oracle SOA Suite. In the context of a simplified business scenario, the demo integrates a reasonable number of lightweight applications involved in an order fulfillment process. The Application Integration Architecture Foundation Pack Demo implements the same business flow, but additionally leverages the entire Application Integration Architecture Foundation.

Foundation Pack Demo Order Booking Business Process

The Application Integration Architecture Foundation Pack Demo implements a simplified Order-to-Fulfillment business process in the context of a fictitious retail company called Global Company, which offers various electronic equipment in an online shop.

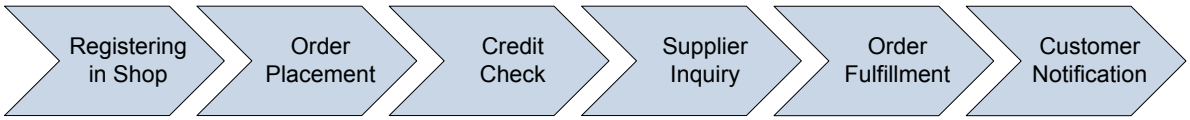

Figure 4: Order-to-Fulfillment Process Diagram

Customers can select products while browsing through Global Company's offerings in the online shop. After logging in or registering as new customers, they can select electronic items in the shop application. After purchase order submission, the backend system automatically initiates a business flow to process the order through all stages of the flow. After propagating the order to the Order Management application, it collects basic information about the customer and requests an external credit check. If the credit check is successful, the process queries two suppliers for quotes for the ordered items. Then, the process selects the best offer and proceeds to the order fulfillment stage. For simplicity, the order fulfillment consists of placing a shipment order at shipper *Exiting Shipper* and finally notifying the customer about the shipment.

Note. We will use the terms *Existing Shipper* and *New Shipper* throughout this document. In a real Foundation Pack Demo environment, replace these with *FedEx* and *USPS* to match the Application Integration Architecture artifacts that are shipped with the demo.

In addition to this order handling process, a synchronization process is available to propagate new customers' data to the CRM system whenever a customer registers in the online shop.

Foundation Pack Demo Architecture

The Application Integration Architecture demo flow integrates a number of loosely coupled applications. The Application Integration Architecture demo flow needs to orchestrate communication between participating applications accordingly to realize the required functionality.

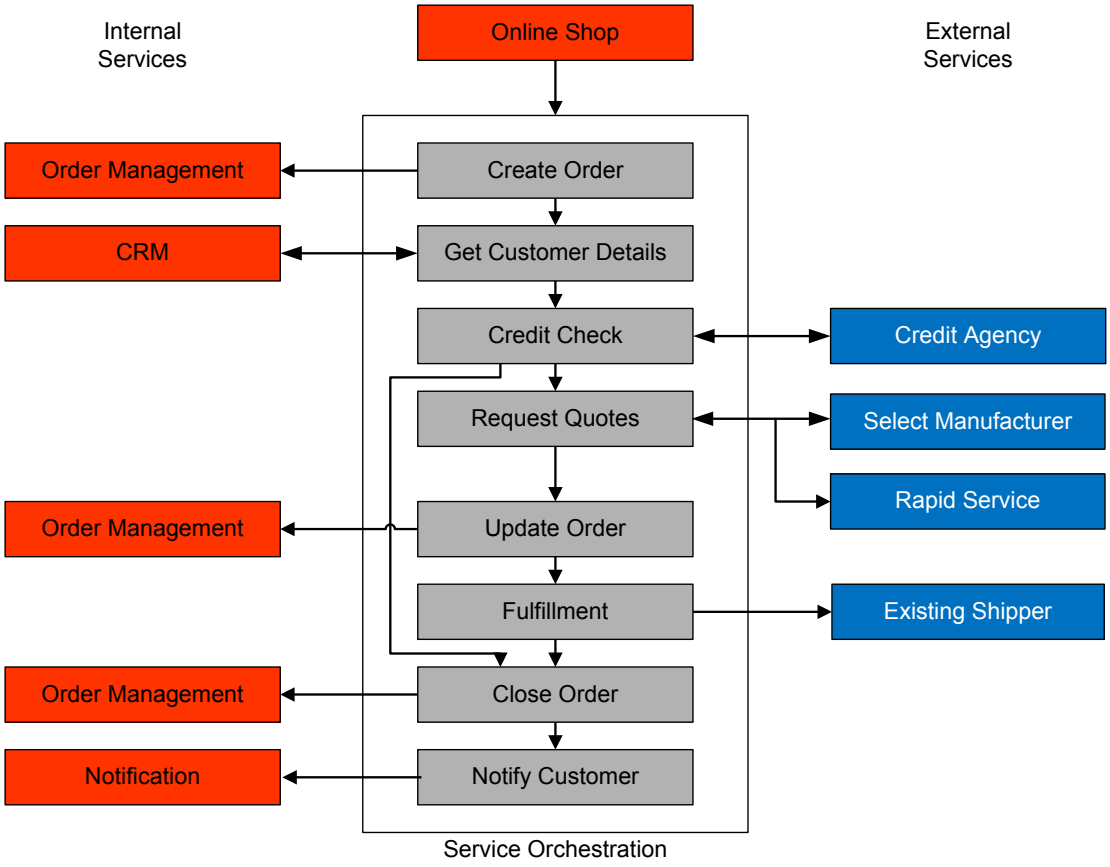

Figure 5: High-level system architecture

Figure 5 outlines the business flow in more detail and shows the participating applications and data flows that make up the Order-To-Fulfillment process.

At a more technical level, the following diagram shows the technical details for the Create Order part of the process:

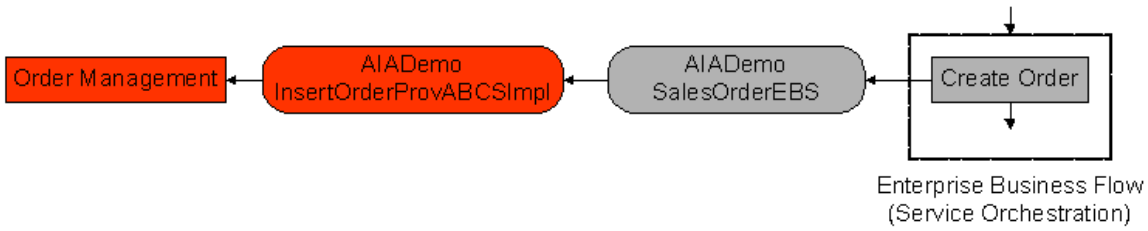

Figure 6: Application Integration Architecture SOA artifacts for the Create Order process step

The Create Order process step, as shown in Figure 6, creates a new order within an Order Management system. In the demo implementation, we decided to use the UpdateSalesOrdermethod of the AIADemoSalesOrderEBS to propagate both the create and update events on orders. Because the Order Management system is known to be the provider of this service, the AIADemoSalesOrderEBS calls the appropriate method of the AIADemoInsertOrderProvABCS (or AIADemoUpdateOrderProvABCS for an existing order) by handing over the complete order in the SalesOrderEBO canonical format as a part of the UpdateSalesOrderEBM message structure. The AIADemoInsertOrderProvABCS can extract this SalesOrderEBO from the EBM and transform it into a structure that is specific to the Order Management system. Finally, the ABCS submits the order to the Order Management system by using methods and protocols that the application understands. In the case of the Application Integration Architecture Foundation Pack Demo, this results in the insertion of a record into the Order Management database.

Chapter 5: Creating the Foundation Pack Demo

To implement the Application Integration Architecture Foundation Pack Demo, we will follow all main aspects of the Application Integration Architecture reference architecture. In particular, this requires the following development effort:

- Design and implement the enterprise business services (EBSs) based on the following enterprise business objects (EBOs): CustomerPartyEBO, SalesOrderEBO, and ItemEBO.
- Design and implement the ABCSs to implement the service operations in the context of the participating applications.
- Design and implement the EBF to orchestrate all of the EBSs to realize the Application Integration Architecture Foundation Pack Demo flow.

The following sections describe the implementation of these Application Integration Architecture artifacts on a mainly conceptual level. For more details regarding Application Integration Architecture methodology and the technology, please consult the *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide*.

Creating the Enterprise Business Services and Messages

Always start with identifying the enterprise business services that you need to implement your integration scenario.

For the Application Integration Architecture Foundation Pack Demo, we have identified the following Application Integration Architecture-delivered EBOs as being relevant to the business flow:

- CustomerPartyEBO: Represents the customer purchasing items.
- SalesOrderEBO: Contains all items in a customer's purchase.
- ItemEBO: Defines details of the items in a customer's purchase.

To handle these EBOs, we need the following EBSs with their corresponding operations, as shown in the following table:

Enterprise Business Service (Enterprise Business Object)	Operation	Enterprise Business Messages
AIADemoCustomerPartyEBS (CustomerPartyEBO)	CreateCustomerParty	CreateCustomerPartyEBM
	QueryCustomerParty	QueryCustomerPartyEBM, QueryCustomerPartyResponseEBM

Enterprise Business Service (Enterprise Business Object)	Operation	Enterprise Business Messages
	CheckCredit	CheckCreditEBM, CheckCreditResponseEBM
AIADemoSalesOrderEBS (SalesOrderEBO)	CreateSalesOrder	CreateSalesOrderEBM
	UpdateSalesOrder	UpdateSalesOrderEBM
	FulfillOrder	ProcessSalesOrderEBM
	NotifyCustomer	CreateSalesOrderEBM
AIADemoItemEBS (ItemEBO)	RequestItemPrice	RequestItemPriceEBM, RequestItemPriceResponseEBM

Note that some of the operations follow the asynchronous fire-and-forget pattern (for example, CreaterSalesOrder), while others are synchronous request-reply operations that immediately deliver a response for a request. This is why the CreateCustomerParty, CreateSalesOrder, UpdateSalesOrder, FulfillOrder, and NotifyCustomer operations do not need a response EBM, whereas the remaining operations require response messages.

EBSs are always implemented as lightweight services in Oracle Enterprise Service Bus (ESB). A routing service in ESB implements the exact service defined in the EBS WSDL.

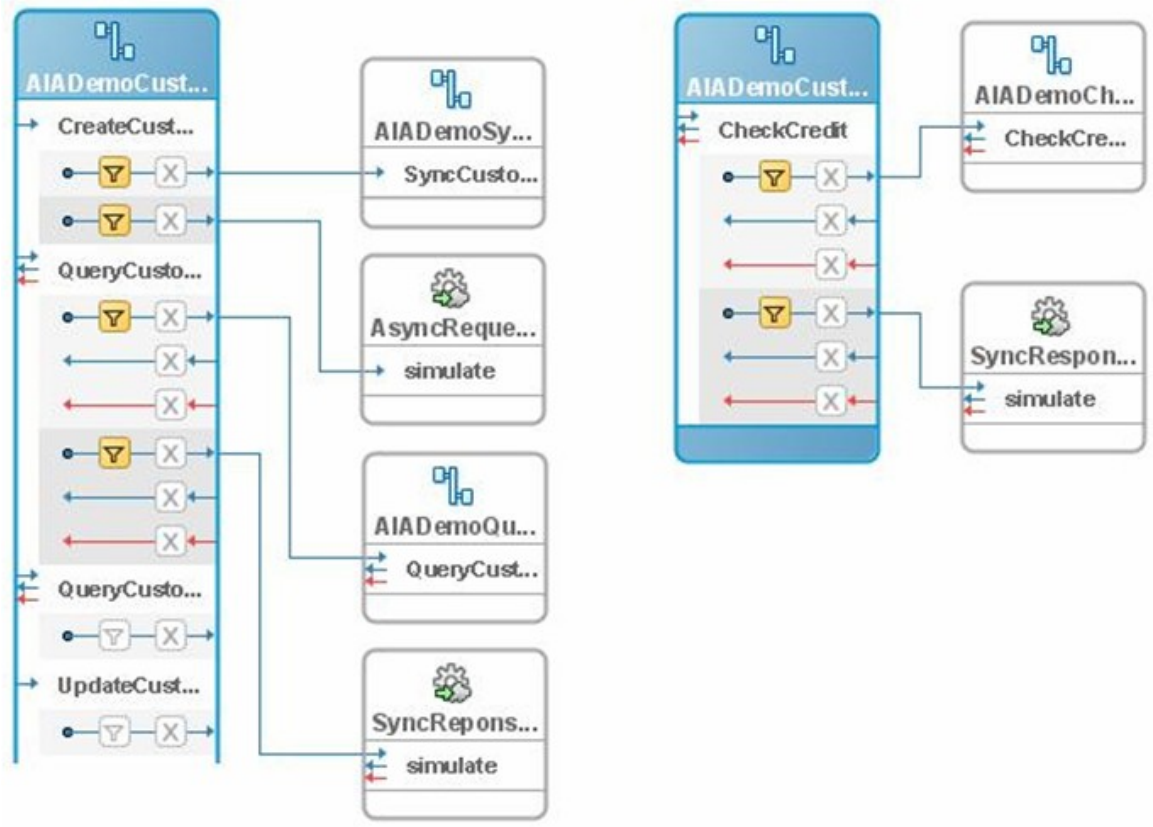

Figure 7: ESB implementation of the EBS AIADemoCustomerPartyEBS

As an example, Figure 7 shows the AIADemoCustomerPartyEBS. It consists of two ESB implementations (AIADemoCustomerPartyEBSV1 and AIADemoCustomCustomerPartyV1) to clearly separate the standard operations delivered with the Foundation Pack (left part) from additional custom operations such as CheckCredit in this case (right part). The routing rules of the ESB routing service usually forward the requests to the respective ABCS that implements the desired functionality. Routing rules also exist to forward the request to the CAVS simulator (SyncResponseSimulator and AsyncRequestRecipient) when the EBS is run in CAVS mode. Details on CAVS enablement can be found in the *Oracle Application Integration Architecture – Foundation Pack: Integration Developer's Guide*.

Creating the Application Business Connector Services

Create Application Business Connector Services to implement the identified services in the context of specific applications.

For every operation of the generic Enterprise Business Services that we identified, we now need to implement the respective ABCSs. ABCSs take the EBMs as input and output (for request-reply) and transform them into application structures and method invocations. For the Application Integration Architecture Foundation Pack Demo, we have the following ABCSs along with the participating applications with which they are communicating:

Application Business Connector Service	Role	Participating Application or Service
AIADemoBookOrderReqABCSImpl	Requester	Online Shop
AIADemoInsertOrderProvABCSImpl	Provider	Order Management
AIADemoQueryCustomerPartyProvABCSImpl	Provider	CRM
AIADemoCheckCreditProvABCSImpl	Provider	Credit Check
AIADemogetSMItemPriceProvABCSImpl	Provider	Select Manufacturer
AIADemogetRSItemPriceProvABCSImpl	Provider	Rapid Service
AIADemoUpdateOrderProvABCSImpl	Provider	Order Management
AIADemoExistingShipperProvABCSImpl	Provider	Existing Shipper
AIADemoNotifyCustomerProvABCSImpl	Provider	Notifications
AIADemoSyncCustomerPartyReqABCSImpl	Requester	Online Shop
AIADemoSyncCustomerPartyProvABCSImpl	Provider	CRM

An ABCS can be implemented as an ESB service or BPEL process. The decision depends mainly on the complexity of the ABCS.

If the participating application provides services in the right granularity, a fairly simple ESB process should work. On the other hand, if the requirement exists to implement a chatty conversation with an application, this usually indicates the need to use BPEL.

As stated before, an ABCS can have either the requester or provider role in an integration scenario. The requester ABCS AIADemoBookOrderReqABCSImpl, for instance, is invoked by the Online Shop application to process a new order. It then transforms the order into the EBM containing the SalesOrderEBO structure and invokes the AIADemoSalesOrderEBS. On the other hand, a provider ABCS, such as AIADemoCheckCreditProvABCSImpl, is called by AIADemoCustomerPartyEBS. It receives the request EBM (including the CustomerPartyEBO) to carry out the credit check for a particular customer. It transforms this into the format that the Credit Check Agency's service understands. The service answers itself with its specific format and the ABCS must take care to convert it back to the response EBM that the calling EBS AIADemoCustomerPartyEBS expects. Figure 8 shows the implementation of AIADemoCheckCreditProvABCSImpl. It mainly performs the following tasks:

- Receives the CheckCreditEBM (including the CustomerPartyEBO) and transforms it into the format expected by the credit check service.
- Invokes the credit check service (ValidateCreditCardService).
- Transforms the response into the CheckCreditResponseEBM and responds to the service caller, AIADemoCustomerPartyEBS.

The remaining parts of the BPEL process are responsible for a proper integration into the Application Integration Architecture Error Handling framework and the Application Integration Architecture CAVS testing framework.

Figure 8: BPEL implementation of AIADemoCheckCreditProvABCSImpl, including support for CAVS and Error Handling

Creating the Enterprise Business Flow

The Enterprise Business Flow is the BPEL process that orchestrates the EBSs to implement the required business flow.

After defining the EBSs and ABCSs, we have everything in place that we need to orchestrate the entire Order-to-Fulfillment business flow. For the orchestration of process activities, we need an Enterprise Business Flow (EBF). In the Application Integration Architecture Foundation Pack Demo, the BPEL process AIADemoOrderEBF has this responsibility.

In essence, the AIADemoOrderEBF implements a series of sequential and parallel calls to all of the EBSs we have implemented so far. The EBF itself is launched by the CreateSalesOrder operation of the AIADemoSalesOrderEBS. It then invokes the QueryCustomerParty and CheckCredit operations of the AIADemoCustomerPartyEBS, along with several other operations as required by the business process. Figure 9 shows the part of AIADemoOrderEBF that requests quotes from its suppliers (Rapid Service and Select Manufacturer) in parallel and then selects the best offer.

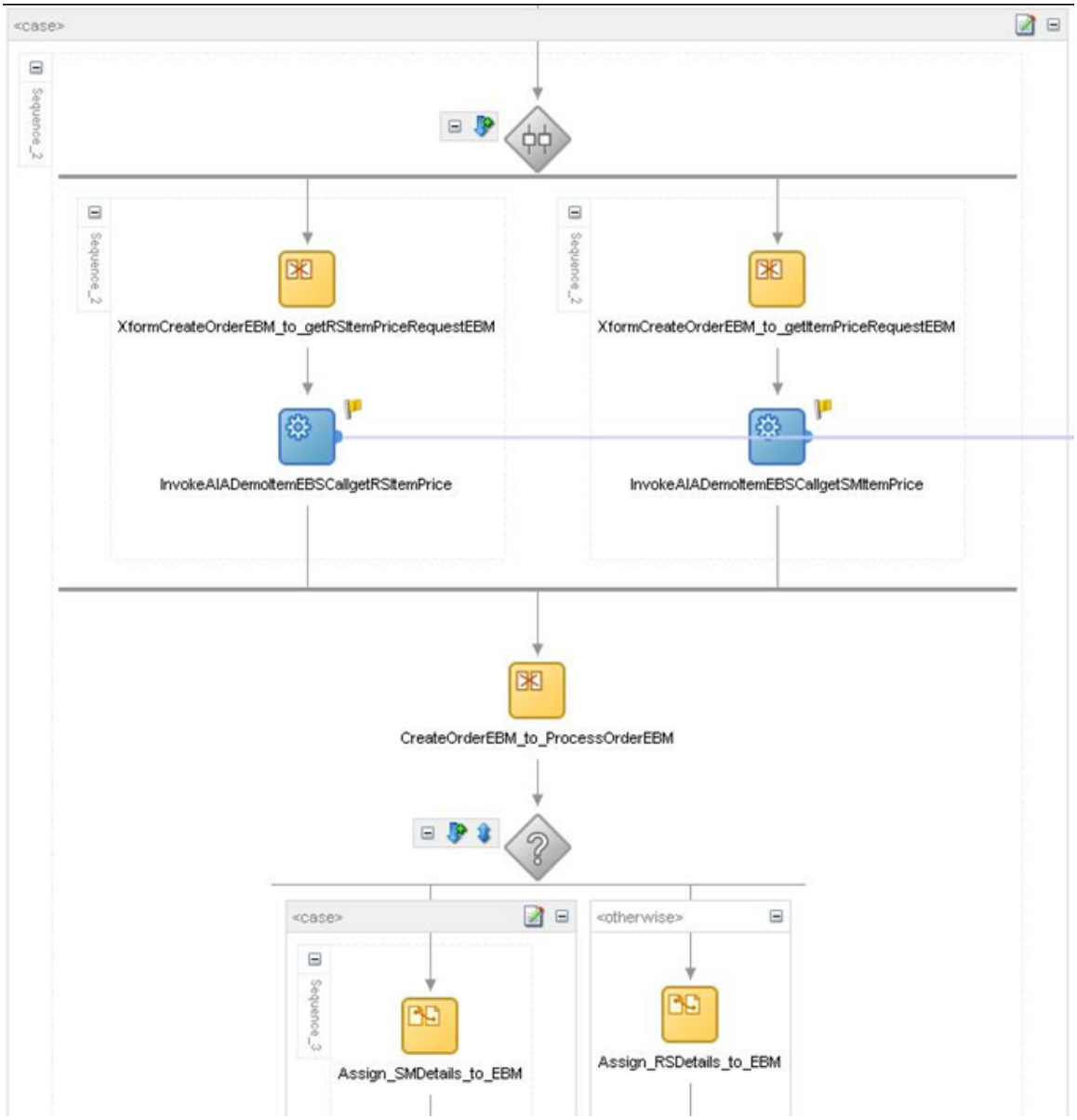

Figure 9: The part of the BPEL implementation of AIADemoOrderEBF that requests quotes from suppliers

With the implementation of the EBF, we have completed the implementation of the Application Integration Architecture Foundation Pack Demo. We are now ready to deploy all built services to an Oracle SOA Suite instance with the Oracle Application Integration Architecture Foundation Pack to see how the integration flow works.

Chapter 6: Running the Foundation Pack Demo

The Foundation Pack demo reuses many parts of the Oracle SOA Suite Order Booking Demo. As such, the Foundation Pack Demo uses the same online store application to place new orders and to kick off the demo order processing. You can access this application on the Oracle SOA Suite instance with the deployed Foundation Pack Demo using the URL http://<server>:<port>/soademo. You can either create a new user account by clicking Register or use the seeded account sking@soademo.org with password welcome1. After logging in, for example, you can browse through the items, add items to the cart, and finally place the order.

Figure 10: The Foundation Pack Demo Online Store application

After placing the order, you can check the initiated BPEL processes and ESB instances in the respective consoles provided by the Oracle SOA Suite. The instance of the AIADemoOrderEBF BPEL process provides an especially good overview of the process steps that were followed to fulfill the placed order.

For details on how to access and use the Oracle SOA Suite BPEL and ESB console, review Oracle SOA Suite documentation.

For further documentation on the underlying Oracle SOA Suite Order Booking Demo, consult the Quick Start Guide.

Chapter 7: Extending the Foundation Pack Demo

Extensibility is a key principle in the design of Application Integration Architecture.

One of the key principles for the design of Application Integration Architecture is its extensibility model. Application Integration Architecture offers extensibility in different shapes. First, the design of the enterprise business objects (EBOs) includes mechanisms to extend these generic objects in an upgrade-safe manner by providing hooks to plug in additional industry-specific or customer-specific information. Second, the whole architecture is prepared to consistently add additional applications to an existing integration scenario by allowing the plugging in of additional ABCSs.

The following sections describe typical extensions to existing integrations. First, we will cover how the Application Integration Architecture Foundation is prepared to offer customers ways to tailor common objects such as EBOs and EBMs according to their unique business requirements.

Second, these sections cover how to consistently plug another application into a new or existing integration scenario. For this purpose, we will change the fulfillment process to choose the distributor depending on the actual amount of the order. To achieve this extension, we implement another ABCS for the additional distributor service and wire that into the existing AIADemoSalesOrderEBS.

Extending an Enterprise Business Object

Oracle has invested heavily in creating the proper definitions of Enterprise Business Objects, such as SalesOrderEBO and CustomerPartyEBO. These prepackaged canonical representations of typical business entities were created by considering both standard approaches, such as OAGIS, as well as the rich set of Oracle-owned applications, such as E-Business Suite, Siebel, PeopleSoft, and others.

However, these definitions should never contain any attributes that are specific to certain industries or customers. This is why the Application Integration Architecture Foundation Pack provides the means to extend EBOs in a nonintrusive way. These extensions will survive any patches and upgrades to protect customers' investments. For this purpose, Application Integration Architecture provides a custom schema definition file (XSD) for every EBO that is shipped with Application Integration Architecture. These custom schemas are incorporated into the actual EBOs and are never touched by patches or upgrades.

The custom extensions of Enterprise Business Objects are preserved across patches and upgrades.

In the context of the Application Integration Architecture Foundation Pack Demo business process, we use this mechanism to store the attributes SupplierName and SupplierPrice at the order header level. We want to store the name of the supplier with the best offer we requested along with the total supplier price within the order. Because these attributes are not part of the base SalesOrderEBO, we can add these using the Application Integration Architecture extensibility model. In our example, we want to extend the custom schema CustomSalesOrderEBO.xsd to add the additional attributes to every order.

To add the attributes on the order header level, we need to change the following part of the schema definition:

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

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

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

When we have done this, 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 our case, these are the CreateSalesOrderEBM, UpdateOrderEBM, FulfillOrderEBM, and NotifyCustomerEBM. As the EBMs are also extended, the extended message definition also extends all EBSs and ABCSs that work with these EBMs.

Extending an Integration Scenario

In this section, we drill down into what is required to extend an existing business flow. For the Application Integration Architecture Foundation Pack Demo, let's assume that Global Company decides to use different distributors depending on the total amount of orders. If the order amount exceeds 2000 USD, we want the order to be shipped by *New Shipper*, otherwise, *Existing Shipper* should work as before. Figure 11 illustrates how this changes the overall flow.

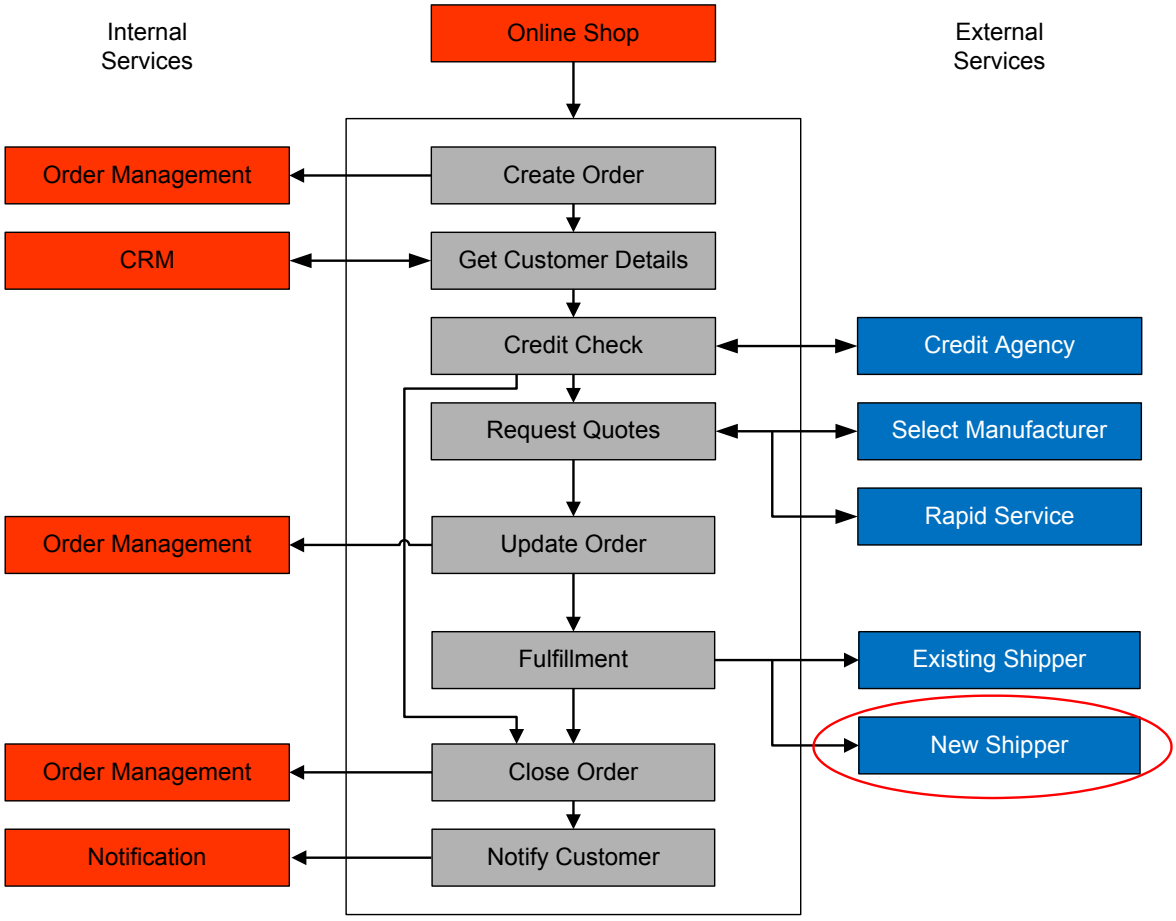

Figure 11: Extending the flow with another application (*New Shipper*)

To achieve this desired change to the flow, we need to identify and understand the services that make up the flow. In particular, we need to know the relevant EBF, EBOs, and ABCSs of the integration.

Identifying these services for an existing flow can be quite challenging. Doing so is simple in the context of the Application Integration Architecture Foundation Pack Demo, but in an actual business scenario we would first consult the Business Service Repository to understand the existing artifacts and their relationships to each other. For more details, see the "Oracle Business Service Repository" section of this document.

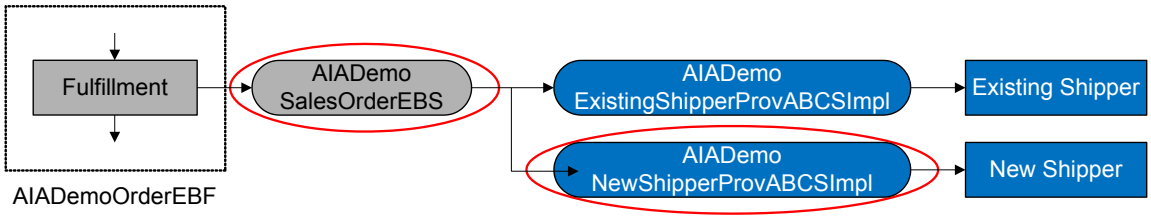

Figure 12: Adding an additional service provider (New Shipper) into the flow

For the extension of our Application Integration Architecture Foundation Pack Demo, we first require a new ABCS implementation for the New Shipper service called AIADemoNewShipperProvABCSImpl. Then, we will implement a moderate change to the AIADemoSalesOrderEBS to include the new application in the existing flow, as shown in Figure 12.

Creating an Application Business Connector Service

Now we want to implement the ABCS for the *New Shipper* distributor and call this AIADemoNewShipperProvABCSImpl. This time, we implement this as an ESB service because we need a fairly simple transformation of the ProcessSalesOrderEBM into the format that *New Shipper* requires. Also, this time we implement an asynchronous “fire-and-forget” operation.

With an implementation in ESB, the service smoothly integrates into the Application Integration Architecture Error Handling framework and the CAVS testing framework in the same way as a BPEL process does. The new AIADemoNewShipperProvABCSImpl ESB service process is shown in Figure 13.

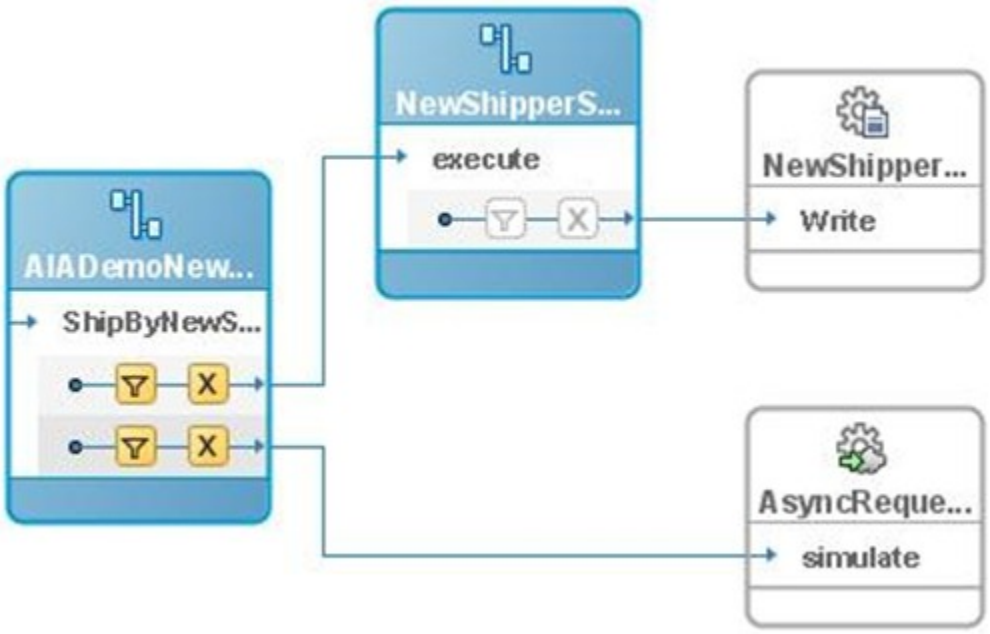

Figure 13: ESB implementation of AIADemoNewShipperProvABCSImpl

Technically, in this case the application is just another simple ESB service (called NewShipperShipmentService in the center of Figure 13) that stores a structured file using the file adapter delivered with the Oracle SOA Suite. Also note the content-based routing rules in the ABCS that are used to route the message to the actual application (NewShipperShipmentService) or to the generic CAVS service simulator. For details on the routing rules that are used to enable CAVS, consult the *Oracle Application Integration Architecture – Foundation Pack: Integration Developer’s Guide.*

Extending an Enterprise Business Service

As a second step, we need to include the AIADemoNewShipperProvABCSImpl in the business flow. To achieve this, we need to extend the respective Enterprise Business Service, which is AIADemoSalesOrderEBS. The actual implementation of the AIADemoSalesOrderEBS, which is responsible for the custom operations FulfillOrder and NotifyCustomer, is called AIADemoCustomSalesOrderEBSV1.

Enterprise Business Services are implemented with the Oracle Enterprise Service Bus (ESB). It is a fairly simple task to add a new target into the message routing as shown in Figure 14.

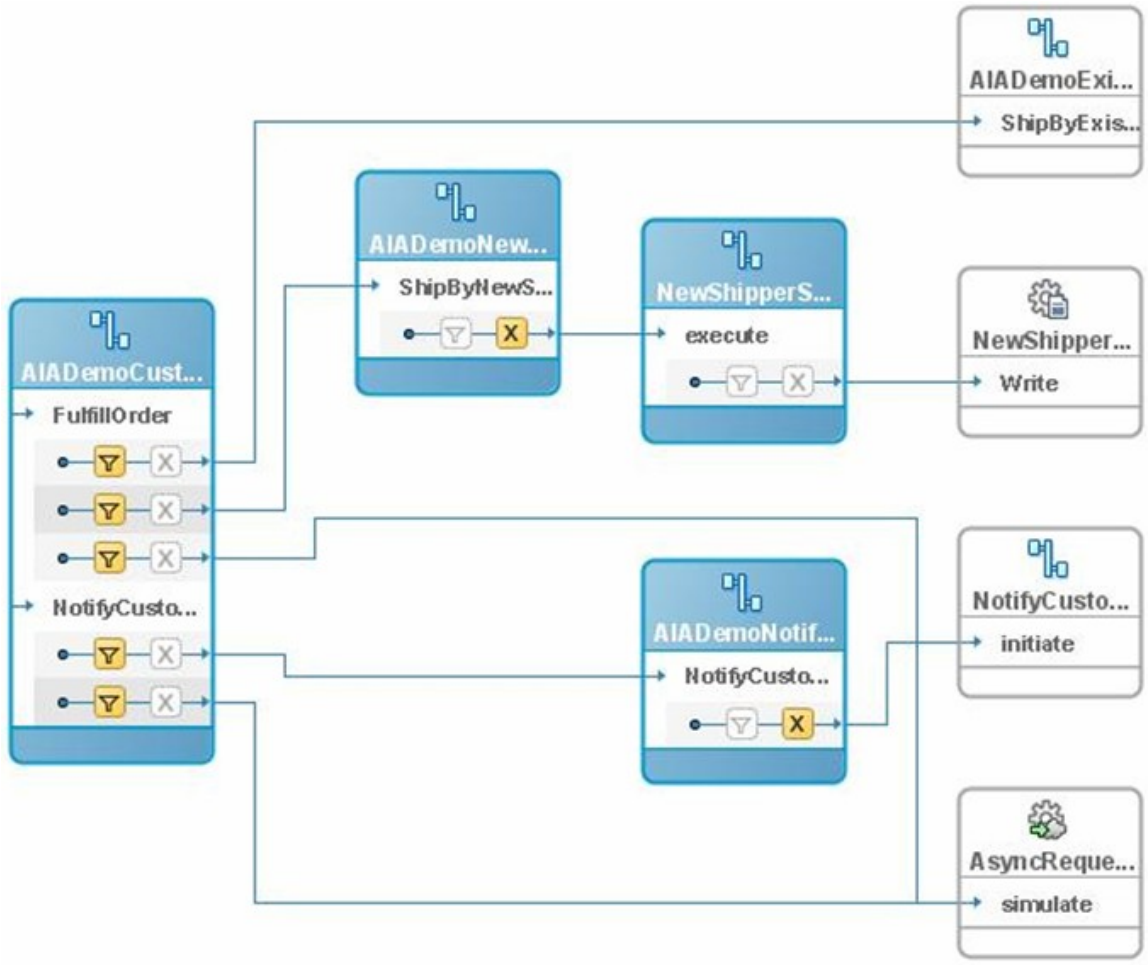

Figure 14: The extension of AIADemoSalesOrderEBS (part AIADemoCustomSalesOrderEBSV1) in the ESB design tool

While the basic implementation of the AIADemoSalesOrderEBS directly routes the fulfillment message to the AIADemoExistingShipperProvABCSImpl, the logic needs to change now. With the availability of the AIADemoNewShipperProvABCSImpl, the content-based routing rules now need to define which ABCS to call for a particular order. For this purpose, we assign the following XPath filter criteria to the appropriate routing rules within the AIADemoSalesOrderEBS.

Existing Shipper:

```
/ProcessSalesOrderEBM/DataAreaProcessSalesOrder/SalesOrderCharge/Cha
rge/Amount <=2000
```

New Shipper:

```
/ProcessSalesOrderEBM/DataAreaProcessSalesOrder/SalesOrderCharge/Cha
rge/Amount >2000
```

In an actual business implementation, this decision would not be hard-coded but would be based on configurable business rules defined in Oracle Business Rules, a component of the Oracle SOA Suite.

After deploying the new ESB service AIADemoNewShipperProvABCSImpl and the modified BPEL process AIADemoSalesOrderEBS to the Oracle SOA Suite, we are done and the new flow is ready to go. Readers can now easily verify that new orders are shipped by *Existing Shipper* or *New Shipper* depending on the amount of the order.

Chapter 8: Application Integration Architecture Infrastructure Components

Service-oriented architectures (SOAs) require a special set of infrastructure components to successfully manage the entire life cycles of SOA artifacts such as services, message definitions, integrations scenarios, and so forth. The remainder of this document explains the Application Integration Architecture infrastructure components in more detail and shows how organizations can benefit from them. For additional details, readers can also consult the Application Integration Architecture Core Components Guide.

Oracle Business Service Repository

Service-oriented environments usually consist of hundreds of technically independent services. Organizations can quickly get lost without actual and consistent information about their SOA assets. They absolutely need a single source of truth to answer questions such as:

- Does a service exist that I can reuse for a certain purpose?
- What is the impact of a change on a service?
- How can I plug an additional service into an existing business process?

Oracle BSR provides deep insight into an organization's SOA artifacts.

Oracle Business Service Repository (BSR) delivers this single source of truth that organizations need to manage their SOA. The BSR enables users to easily browse through the catalog of SOA artifacts, starting from a high-level perspective, such as the integration scenario.

For the Application Integration Architecture Foundation Pack Demo, we had to understand the artifacts that make up the integration scenario FulfillOrder Provider before we could start extending this process step in the previous section. We can search for the integration scenario within the BSR by accessing http://<server>:<port>/AIA and navigating to the Business Service Repository search page as shown in Figure 15.

Figure 15: Searching for Application Integration Architecture integration scenarios

By drilling down into the details of this scenario in the BSR (see Figure 16), we see that the Enterprise Business Service AIADemoSalesOrderEBS is connected to the *Existing Shipper* service provider through the Application Business Connector Service AIADemoExistingShipperProvABCSImpl.

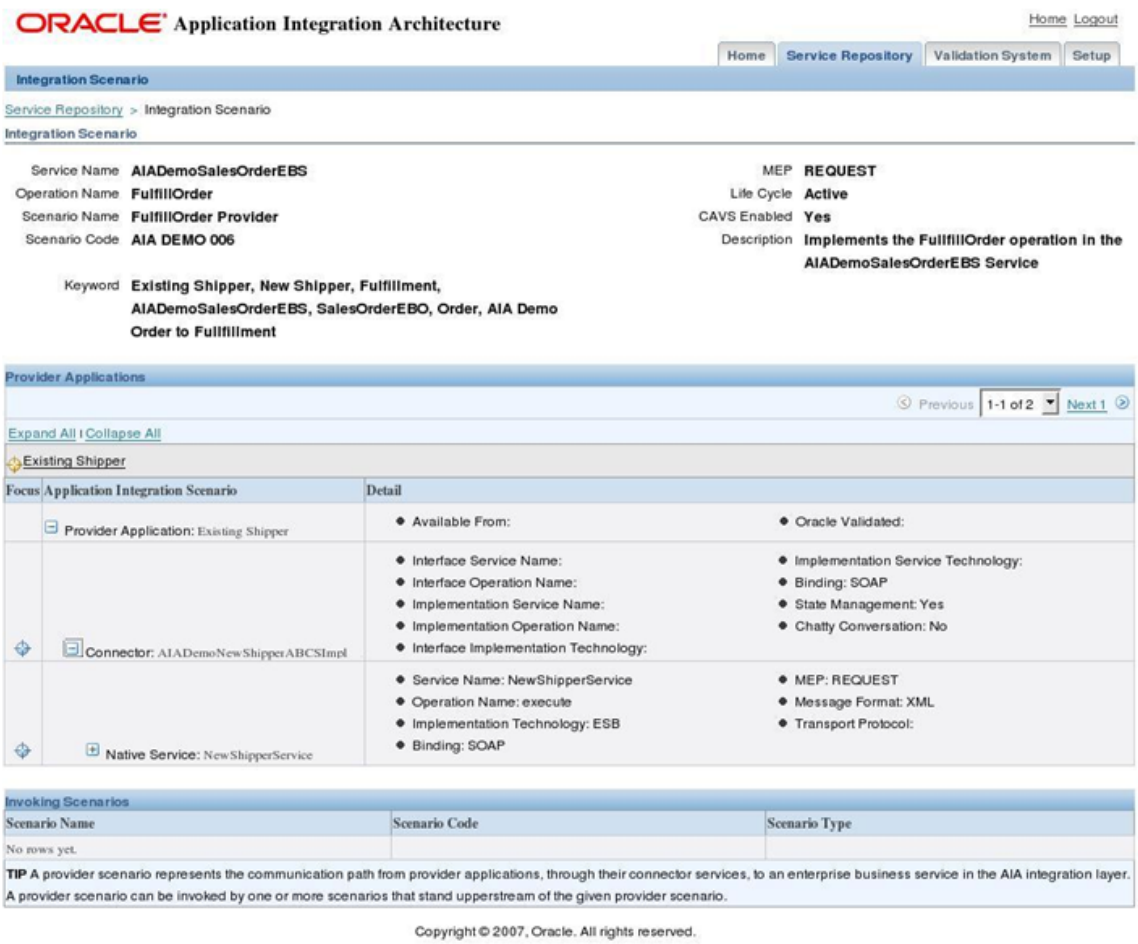

Figure 16: Browsing through the catalog of Application Integration Architecture artifacts (Fulfill Order Provider)

This kind of insight into an organization's SOA is harder to achieve with standard UDDI approaches because they may not cover concepts such as an integration scenario, for instance.

Although identification of relevant artifacts seems to be obvious in the context of our simple Application Integration Architecture Demo scenario, this task can be a huge challenge when an organization's infrastructure contains hundreds or even thousands of available services and no repository such as the BSR is available.

However, the BSR is not only about searching and browsing through a structured catalog of services. It also enables customers to manage the entire life cycle of SOA artifacts, from the identification of services to implementation, and finally to the retirement of services.

Composite Application Validation System

Testing a SOA-based system has several unique requirements that need to be addressed by the SOA infrastructure, especially when the integrations are complex and involve external services. Because Application Integration Architecture integration scenarios consist of several services calling each other, you must be able to test against a single service or an orchestration of services to be able to validate the health of the entire integration.

With CAVS, you can define, perform, and evaluate test invocations against any web services.

For exactly this purpose, Application Integration Architecture provides the Composite Application Validation System (CAVS). With CAVS, customers can declaratively define tests on any kind of service. These tests can be performed at any time to ensure that services are doing exactly what they are supposed to. Even if the participating applications aren't available, CAVS is still able to test single services or whole integration scenarios by simulating these applications. By substituting applications with simulators, CAVS also allows the testing of single services or groups of services in isolation because the simulation removes undesired dependencies on specific behaviors of real services.

For the implemented Application Integration Architecture Foundation Pack Demo extension, we want to set up a test scenario to verify that orders are being routed to the correct distributor. To check this, we define two tests on the FulfillOrder operation within the AIADemoSalesOrderEBS. Test order A is a small one (total amount=100 USD) and should be shipped by *Existing Shipper*. The second order, B, is about 5000 USD and must be shipped by *New Shipper*, according to the demo business process.

CAVS simulators can emulate any web service. You can test integrations even when participating applications are not in place.

Because we don't want to or just can't invoke external services when testing an integration scenario, we can set up simulators that simulate the behaviors of any service providers. In CAVS, we can define a simulator by assigning it the expected incoming message. Additionally, we can define a result message (only if it simulates a synchronous service), as well as XPath expressions. These XPath expressions are validated during processing to check whether the incoming message follows expected rules.

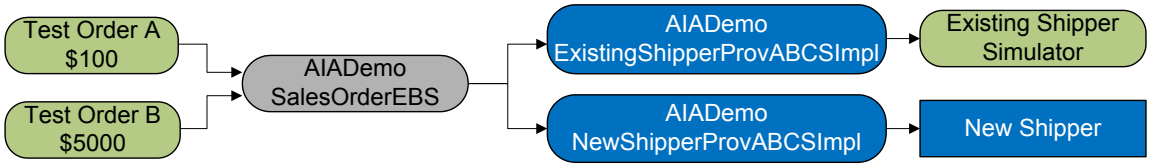

Figure 17: Test and simulation scenario in CAVS

In our example, we assume that the *Existing Shipper* service is not available to be used for tests. Therefore, we define a CAVS simulator to prevent us from sending test messages to the real application. For *New Shipper*, we assume that we can safely use the real *New Shipper* service for test messages and don't require another simulator.

We expect the *Existing Shipper* service simulator to receive only orders with an amount below 2000 USD, whereas the *New Shipper* service should only receive orders above that amount. Figure 17 shows the whole test scenario with the test definition against the AIADemoSalesOrderEBS. The EBS routes the message to one of the ABCSs. The *Existing Shipper* ABCS is aware (by setting a property in the Application Integration Architecture configuration file) of working in the CAVS context and forwards the messages to the CAVS simulator instead of the actual service endpoint. Finally, the *Existing Shipper* simulator validates the incoming messages according to their structures, and with the order amount being below 2000 USD, verifies the proper message routing in the AIADemoSalesOrderEBS.

CAVS enables the definition of tests on particular service endpoints and evaluates the result of a service invocation depending on the service type. In our case, we have an asynchronous notification scenario (fire-and-forget) in which the caller of the service doesn't receive a response. In a synchronous scenario, the test could, of course, also evaluate the response delivered by the invoked service.

Error Resolution and Logging

A centralized error handling approach across applications, technologies, and integration patterns is key for successfully operating a SOA.

The operation and maintenance of modern SOAs require an error handling and logging solution. The Application Integration Architecture Foundation Pack includes a unified approach for error handling and resolution across applications, technologies, and integration patterns. The whole architecture is designed such that all kinds of errors that could arise within the integration can be captured and handled easily. For this purpose, Application Integration Architecture provides a declarative setup for actual applications and error types to route error notifications to the appropriate people or group of people, depending on which type of error occurred in which system.

For the Foundation Pack Demo, we want to assume now that one of the participating services is not available due to an unplanned system outage. We can simulate such a scenario by stopping the CreditService application in the Oracle Enterprise Manager Console. The next invocation of the AIADemoCheckCreditProvABCSImpl must then fail because the service is down and an appropriate error notification is generated and assigned as shown in Figure 18.

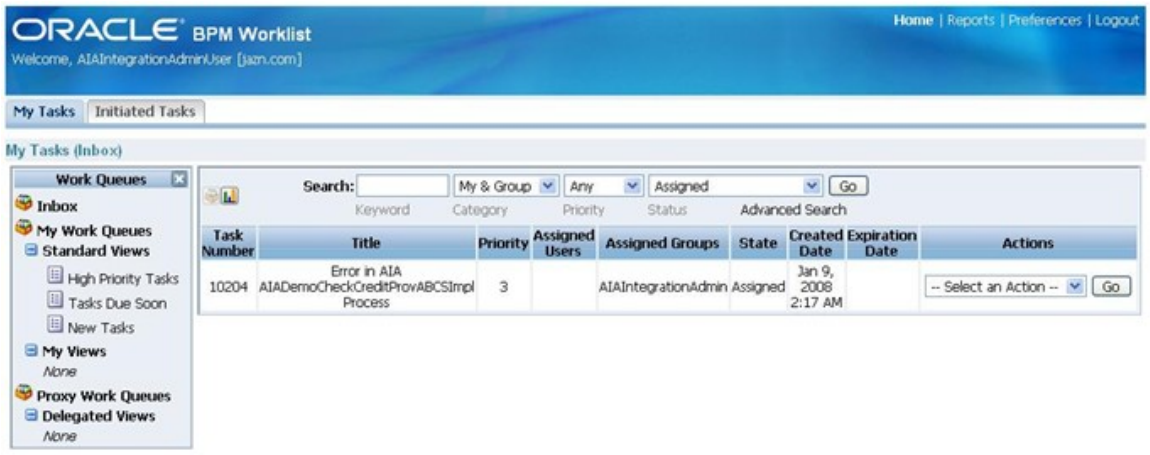

Figure 18: Error tracking within the BPM Worklist application

Application Integration Architecture leverages the BPM Worklist application to assign errors to particular roles. Integration architects or integration maintenance staff can then pick an unresolved error to get further information about the issue, including the initial application, the integration context, EBOs and EBMs involved, references to particular BPEL or ESB processes, and much more. With this knowledge, the integration specialists can resolve the issue and track it in the Worklist application.

In the preceding example, the notification contains information about the respective BPEL process and fault details such as error codes and error descriptions. In this case, the notification's message text contains the error summary:

```
summary=exception on JaxRpc invoke: HTTP transport error
```

The integration administrator will know that the communication with the partner link ValidateCreditCardService is broken. After resolving the base issue and bringing up the service again, the administrator can resubmit the order and it will be processed successfully.

The Application Integration Architecture Foundation also takes care to log any error occurrence into a central log. As with any other application running on Oracle middleware, this log is accessible through the Administration Console. Administrators can either browse the log or search using integration-specific attributes, such as message identifiers, applications, EBOs, and EBMs.

The Application Integration Architecture error handling mechanism can also be plugged into other error tracking applications that may already be in place in an organization. Application Integration Architecture provides an open architecture so that the centralized error capture mechanism can queue any error into any system that provides an appropriate interface.

Diagnostics

Application Integration Architecture Diagnostics enables you to validate the consistency of your whole integration landscape at any time.

To ensure the integrity of an Application Integration Architecture-based integration, Application Integration Architecture provides a rich set of diagnostic tests. These tests can check almost any relevant area of the integration stack, including the underlying technology, changes in the predefined Enterprise Business Objects and Messages, changes in the seed data, shipped BPEL or ESB processes, changes in the underlying database schemas, and many other areas.

These tests enable users to quickly check the consistency of the integration infrastructure, particularly after upgrades and patches to applications or to the integration infrastructure itself. Furthermore, these tests can help give Oracle Support Services fast insight into the architecture to effectively support customers.

Chapter 9: Summary

The Application Integration Architecture Foundation Pack delivers everything organizations need to implement state-of-the-art integrations to better support their business processes. Leveraging service-oriented concepts along with a sophisticated governance model provides the means to achieve and maintain the agility businesses require today and tomorrow.

Index