Oracle8*i*

Integration Server Overview

Release 3 (8.1.7)

September 2000

Part No. A83729-01

ORACLE

Oracle8*i* Integration Server Release 3 (8.1.7)

Part No. A83729-01

Primary Authors: Thomas Kurian, Chitra Sharma

Contributing Authors: Anna Sears, Jon WIlkinson

Contents

2 Methodology and Solutions

3 Overview of Oracle Integration Server

4 Key Integration Concepts

5 Synchronous Application Integration

6 Data Replication and Gateways

7 Oracle Advanced Queuing and JMS

8 Oracle Message Broker and JMS

9 Directory Services (LDAP)

Index

Send Us Your Comments

Oracle8*i* Integration Server Overview, Release 3 (8.1.7)

Part No. A83729-01

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the chapter, section, and page number (if available). You can send comments to us in the following ways:

- E-mail - infodev@us.oracle.com
- FAX - (650) 506-7228. Attn: Information Development
- Postal service:
  Oracle Corporation
  Server Technologies Documentation Manager
  500 Oracle Parkway, 4OP12
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This document describes the Oracle Integration Server and its applications to your integration solution.

This preface contains:

Objectives of the Oracle Integration Server

Enterprise or e-business integration means streamlining business processes by enabling applications to communicate with each other. Enterprise integration is driven by the need to synchronize data between different systems, to automate a business process, or to isolate packaged applications from each other. e-Business and particularly B2B commerce is both accelerating the demand for traditional enterprise integration and creating demand for fundamentally new kinds of integration.

Oracle Integration Server facilitates the integration of technologies within e-businesses. It works both within Oracle products and with third-party products.

The OIS enables to:

- Synchronize data among systems using advanced replication and gateways
- Enable applications to communicate with each other synchronously and asynchronously using:
  - Message storage and management
  - Message routing and propagation
  - Data and message transformation services
  - Business process and workflow services

Scope

This book provides an overview of the technological requirements for enterprise integration and discusses various components within Oracle Integrations Server that meet these requirements.

Part III describes several advanced topics in integration such as interoperability between OIS and various third-party integration solutions.

Note that the objective of this document is to introduce you to various conceptual elements of integrations and to discuss how Oracle Integration Server can meet these requirements. It also illustrates the specific components of OIS you should use with a particular integration scenario. However, it does not describe the specific details of each component, but links you to the product-specific documentation on how to use the product, how to develop applications using it, and how to deploy and manage it.

Intended Audience

The primary audience for this document are the architects and designers who define, develop, and implement message-based integration solutions with OIS. Database administrators who manage OIS installations and developers who build predefined components of the integrations solution will also find this document useful.

We assume that you have a basic understanding of the complexities and issues of application integration. Knowledge of transaction processing and Oracle8*i* concepts will prove useful.

Structure of the Document

This document is structured in three parts, each introducing you to a specific aspect of e-business integration and the Oracle integration offering.

Part I provides an overview of e-business integration and discusses the business drivers for integration, specific technical challenges that must be addressed, and the key integration concepts of Oracle Integration Server.

- Chapter 1, "Overview of E-Business Integration"
- Chapter 2, "Methodology and Solutions"
- Chapter 3, "Overview of Oracle Integration Server"
- Chapter 4, "Key Integration Concepts"

Part II provides a brief overview of each of the product components of OIS and links you to the product-specific documentation for more information.

- Chapter 5, "Synchronous Application Integration"
- Chapter 6, "Data Replication and Gateways"
- Chapter 7, "Oracle Advanced Queuing and JMS"
- Chapter 8, "Oracle Message Broker and JMS"
- Chapter 9, "Directory Services (LDAP)"
- Chapter 10, "Workflow"

Part III describes several advanced topics in integration such as interoperability between OIS and various third-party integration solutions.

Related Documents

The following documents have more information about the components within OIS.

Oracle8*i* Server Documentation

- Oracle8i Application Developer's Guide - Advanced Queuing
- *Oracle8i CORBA Developer's Guide and Reference*
- *Oracle8i Replication*
- *Oracle8i Distributed Database Systems*
- *Oracle Internet Directory Application Developer's Guide*
- *Oracle Enterprise Manager Administrator's Guide*

Component Product Documentation

- *Oracle8i Workflow User Guide Release 2.5.2*
- *Oracle8i Message Broker Administrator's Guide 2.0.1.0*
- *Oracle8i Supplied PL/SQL Packages Reference*
- Oracle8i Supplied Java Packages Reference
- Oracle8i Online Help for Oracle Objects for OLE Release 3 (8.1.7)
- *Oracle8i XML Reference*
- Oracle8*i* Applications InterConnect 3.1.3

Conventions

This section describes the conventions used in the text and code examples of the Oracle8*i* documentation set. It describes:

- Conventions in Text
- Conventions in Code Examples

Conventions in Text

We use various conventions in text to help you more quickly identify special terms. The following table describes those conventions and provides examples of their use.

Convention	Meaning	Example
Bold	Bold typeface indicates terms that are defined in the text or terms that appear in a glossary, or both.	... the C datatypes such as **ub4**, **sword**, or **OCINumber** ... When you specify this clause, you create an **index-organized table** ...
Italics	Italic typeface indicates book titles, syntax clauses, or placeholders.	*Oracle8i Concepts* You can specify the *parallel_clause* ... Run U*old_release*.SQL where *old_release* refers to the release you installed prior to upgrading.
UPPERCASE monospace (fixed-width font)	Uppercase monospace typeface indicates elements supplied by the system. Such elements include executables, parameters, privileges, datatypes, SQL keywords, SQL*Plus or utility commands, packages and methods, as well as system-supplied column names, database objects and structures, user names, and roles.	You can specify this clause only for a `NUMBER` column. You can change this value in an `ALTER TABLE` statement. ... grouped by the `DEPTNO` column ... Specify the `ROLLBACK_SEGMENTS` parameter ... ... the `DBMS_STATS.GENERATE_STATS` procedure ...
lowercase monospace (fixed-width font)	Lowercase monospace typeface indicates sample user-supplied elements. Such elements include computer and database names, net service names, and connect identifiers, as well as user-supplied database objects and structures, column names, packages and classes, user names and roles, program units, and parameter values.	The `deptno`, `dname`, and `loc` columns are in the `scott.dept` table. Set the `QUERY_REWRITE_ENABLED` initialization parameter to `true` if ... Connect to the `sales@sf.acme.com` database. Connect as `oe` user.

Conventions in Code Examples

Code examples illustrate SQL, PL/SQL, SQL*Plus, or other command-line statements. They are displayed in a fixed-width font and separated from normal text as shown in this example:

```
SELECT username FROM dba_users WHERE username = 'MIGRATE';
```

The following table describes typographic conventions used in code examples and provides examples of their use.

Convention	Meaning	Example
[ ]	Brackets enclose one or more optional items. Do not enter the brackets.	`DECIMAL (digits [ , precision ])`
{ }	Braces enclose two or more items, one of which is required. Do not enter the braces.	`{ENABLE | DISABLE}`
|	A vertical bar represents a choice of two or more options within brackets or braces. Enter one of the options. Do not enter the vertical bar.	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	Horizontal ellipsis points indicate either: ■ That we have omitted parts of the code that are not directly related to the example ■ That you can repeat a portion of the code	`CREATE TABLE ... AS subquery;` `SELECT col1, col2, ... , col`*`n`*` FROM emp;`
. . .	Vertical ellipsis points indicate that we have omitted several lines of code not directly related to the example.	
Other punctuation	You must enter punctuation other than brackets, braces, vertical bars, and ellipsis points as it is shown.	
Italics	Italicized text indicates variables for which you must supply particular values.	`STARTUP PFILE=init`*`sid`*`.ora` In this example, the entire string `init`*`sid`*`.ora` is a placeholder for a parameter file that must contain your particular instance ID or SID.

Convention	Meaning	Example

UPPERCASE	Uppercase typeface indicates elements supplied by the system. We show these terms in uppercase in order to distinguish them from terms you define. Unless terms appear in brackets, enter them in the order and with the spelling shown. However, because these terms are not case sensitive, you can enter them in lowercase.	`SELECT ename, empno FROM emp;`
		`SQLPLUS username/password`
		`INTO TABLENAME 'table'`

lowercase	Lowercase typeface indicates programmatic elements that you supply. For example, lowercase indicates names of tables, columns, or files.	`SELECT ename, empno FROM emp;` `SQLPLUS scott/tiger`

Part I

Overview of E-Business and Integration

Part 1 introduces business concepts that drive e-businesses towards finding integration solutions and discusses specific technical challenges that must be addressed. It outlines the key concepts of the Oracle Integration Server.

1

Overview of E-Business Integration

This chapter provides an overview of the development of an e-business integration solutions and contains these sections:

- Introduction to E-Business Integration
- Reasons for E-Business Integration
- E-Business Integration Technologies and Approaches

Introduction to E-Business Integration

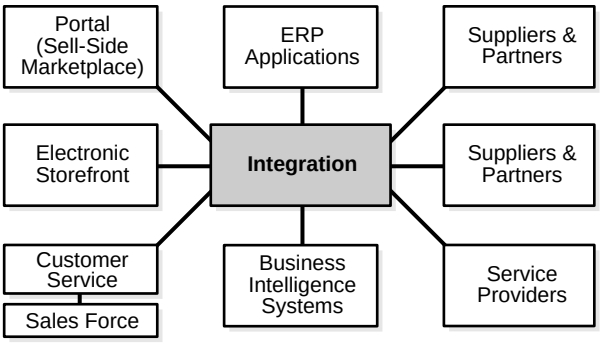
The Internet revolution has advanced to the stage at which every enterprise must become an e-business. This is an imperative and not a choice. Hence, it is necessary to determine when and how an enterprise becomes an e-business.

What is e-business? It is a fundamental change to the way an organization conducts business. An e-business uses Internet technology to:

- Attract, satisfy, and retain the customers who buy its products and services
- Streamline supply chain, manufacturing, and procurement systems to efficiently deliver the right products and services to the customers
- Automate corporate business processes to reduce cost and improve efficiency through self-service
- Capture, analyze, and share business intelligence about customers and company operations. This enables management to make better business decisions and to continually refine business strategy.

An e-business requires a variety of Internet-enabled applications including e-commerce Web sites, portals, supply-chain management, procurement management, online marketplaces, customer relationship management, and enterprise resource planning. All these applications must be integrated with one another to make an enterprise an e-business.

Figure 1–1 Integration, the Key to E-Business Drivers of E-Business Integration



The necessity for businesses to become “zero latency organizations” drives enterprise-wide integration of information systems and applications. For instance, in a smoothly running e-business:

- An order received at an electronic storefront is automatically visible to a customer service representative who must answer customer inquiries about its status.
- The order is automatically propagated to a supply chain application to start a planning and execution operation.
- The order information is exchanged over the Internet with a supplier or partner who provides fulfillment and delivery.

These developments drive the need for e-business integration:

- "Mergers and Acquisitions"
- "Packaged Applications"
- Business Process Re-engineering
- Virtual, Dynamic Supply Chains
- Customer Relationship Management
- Corporate Self-Service
- Business-to-Business Commerce
- Application Service Providers and Hosting

Mergers and Acquisitions

When two or more companies merge, or when one company acquires another, they must automate a company-wide business process for the new entity. To do so, they connect their information systems together to synchronize and share information.

For instance, as telecommunications companies increasingly consolidate globally through mergers and acquisitions, they require a common view of their consolidated customer base. As a result, they must share and synchronize customer information among their front-end databases, their billing systems, and other front-end applications.

Packaged Applications

As businesses buy packaged applications to streamline parts of their business, they must integrate these applications with other packaged applications and with legacy systems within the enterprise. Enterprise integration technologies enable applications to communicate with each other in order to automate business processes.

Business Process Re-engineering

Business process reengineering drives the requirement for enterprise integration. As organizations redesign and streamline their business processes, the underlying applications infrastructure that facilitates these business processes must communicate with different systems in new ways. Companies undergoing business process reengineering deploy integration middleware to connect different systems together.

Although these three developments provide the primary incentives for enterprise integration within companies, companies undergoing a transition to e-business face additional integration needs. With e-business, customers and suppliers have transparent and direct access to the internal business processes of an organization. As a result, the e-business itself creates a demand for enterprise integration within companies and between businesses. These factors include:

- Virtual, Dynamic Supply Chains
- Customer Relationship Management
- Corporate Self-Service
- Business-to-Business Commerce
- Application Service Providers and Hosting

Virtual, Dynamic Supply Chains

Many e-businesses outsource critical parts of their supply chain execution process to partners. These include manufacturing specialists who build specialized components, fulfillment specialists who provide logistics and fulfillment, and warehouse management specialists who manage a supplier's inventory in the warehouse.

For instance, many personal computer manufacturers assemble the computers themselves but outsource the manufacturing of the PC boards and use a third-party logistics provider to ship the PCs directly to consumers. In such a case, the

company's supply chain applications and business processes must be tightly linked with the supply chain systems of the suppliers. This ensures global visibility of inventory levels and demand patterns to all participants. Business-to-business integration software links these systems together.

Customer Relationship Management

In an e-business, customers reach the company through a variety of facilities. They can:

- Access a company's product information through its Web site
- Order products through the company's Web store
- Call a company's customer service operation to check the status of an order
- Follow up with a company's partners in order to get the product serviced

All front-end applications of the business must be integrated so that customers have a unified view of the company no matter what channel they use to reach it. Ideally all front-end applications are designed to be integrated. In cases where applications are not integrated with each other, the business must use integration middleware to stitch the systems together.

Corporate Self-Service

As companies move toward e-business, they convert customer-, supplier-, and employee-facing business processes to self-service. Customers enter their own orders for products by going to a company's Web store; suppliers enter their own purchase orders and complete the requisitioning process through their own secure Web sites; employees file their own expense reports, create their own purchase orders, purchase office supplies, and buy all of their own travel tickets online.

To facilitate self-service, business processes must be streamlined to conduct Straight Through Processing. For instance, when an employee files an expense report, a workflow process notifies the employee's manager and debits the appropriate dollar amount from the company's financial systems, while crediting the employee in the company's payroll systems. E-Business integration middleware integrates all these discrete systems together to facilitate the business process.

Business-to-Business Commerce

As companies increasingly conduct business-to-business commerce through online marketplaces or exchanges, suppliers and customers who connect with these exchanges must automate their interactions in order to:

- Reconcile their product catalogs and prices with the exchanges
- Respond to requests and bids from customers
- Participate in auctions and reverse auctions

Suppliers and customers are beginning to use e-business integration middleware to tie their enterprise resource planning applications with online marketplaces to streamline business-to-business processes.

Oracle Corporation, for instance, offers a version of its own e-business integration middleware that links suppliers to the exchanges the company is building. This is known as the Oracle Integration Server.

Application Service Providers and Hosting

Companies increasingly focus on their core competencies and outsource their enterprise applications to application service providers (ASPs) or to hosting companies. This creates a fundamental need to connect their own legacy information systems with those of the ASP and to connect the applications of one ASP to those of another. As a result, migrating a company's back-office systems from a corporate data center to that of an ASP creates demand for e-business integration middleware to tie these different systems together.

Reasons for E-Business Integration

The business drivers of e-business integration translate into specific requirements within an integration infrastructure. To understand what kinds of integration middleware are required, we identify three fundamental reasons for e-business integration:

- Synchronizing Data Between Information Systems
- Isolating Applications and Businesses
- Streamlining Multistep Business Processes

Synchronizing Data Between Information Systems

The first reason is the need to synchronize data between information systems. e-businesses require a consistent, global, enterprise-level view of their business objects or information. For instance, they require:

- A single, integrated view of each customer across various lines of business
- A consistent view of their supply chain inventory and of demand patterns with their suppliers and partners
- A consistent view of their finances across all of their disparate financial tracking systems

The fundamental integration need in all of these cases is for a consistent global view of information across the different systems that synchronizes data between the different information systems.

Synchronizing data between systems can be done either periodically by batch transfer or continually by repeated transactions.

Batch-Style Data Synchronization

For instance, synchronizing data between an online transaction processing system and a data warehouse is typically done by synchronization batch transfer. A batch job extracts new information from the transactional system, transforms it into the appropriate format, and loads or populates the data warehouse.

Transactional Data Synchronization

Other scenarios, however, require transactional data synchronization. For instance, when a banking customer makes an account transfer on a self-service Web site, the site must immediately reflect changes to his account. Account information stored in the bank's back-office systems must be synchronized through a transaction with the database backing the Web site. In some cases, the bank does not maintain two copies of a customer's account information: one in the database backing the Web site and another in the bank's back-office systems. Instead, the bank simply maintains customer information in one system and provides multiple applications with access to that system. This method has benefits and trade-offs associated with performance, scalability, and security that we discuss in the next section.

Isolating Applications and Businesses

A second reason for intraenterprise or business-to-business integration is to separate one company's business processes and applications from those of its trading partners.

Isolating Applications from Each Other

The growing complexity of software and the associated difficulty in upgrading to new software versions makes it necessary to isolate applications. As software grows more complicated, developers increasingly break big problems into multiple smaller problems that can be solved separately.

Developers create modules with well-defined interfaces between them that they combine together to develop the complete application. They focus on solving small problems within each module. By limiting communication between application modules through well-defined, standardized interfaces and by not sharing data between modules, developers can modify or upgrade one application module without affecting the other application modules. Program-to-program communication simplifies the application replacement process and minimizes the impact of a decision to change an application, relocate a data center, or even outsource the application completely.

Isolating Businesses from Each Other

Software complexity also affects the way one business communicates with another for business-to-business commerce. Each company looks to its trading partner as a provider of a service with well-defined standard interfaces. Developers standardize program-to-program communication by using application component models that define standardized interfaces for each module and a standardized protocol to communicate between modules.

In the same way, communication between companies is now being standardized with the definition of standard interfaces through which they communicate (captured as XML-based business object documents) and standard network protocols such as RosettaNet and OAGIS. Isolating one company's internal business processes from those of another enables each company to modify its internal processes without affecting its trading partners.

For instance, a company can change its own internal purchase order approval process. Because suppliers send the company purchase orders in a standard format over a standard Internet protocol, they are isolated from the company's purchase order approval process. This separation is a fundamental requirement for business-to-business commerce.

Streamlining Multistep Business Processes

Business process automation, frequently called in the industry straight through processing, defines the process through which a multistep business process is streamlined. Automation eliminates human intervention by enabling one application to communicate directly with another.

For instance, most small electronic storefronts and companies conducting commerce online transfer new orders accepted by their front-end Web store database to their back-office financial and supply-chain systems by using a manual process such as file transfer protocol (FTP). Such manual intervention causes three problems: it introduces the possibility of human error, it reduces the speed and efficiency of order processing, and it raises the total cost of operating the storefront as the volume of orders grows.

Larger Web sites use a number of enterprise integration technologies to automate and streamline this multistep business process. These enable the storefront database to automatically propagate the order first to the financial system where the customer's credit history is validated and then to the supply-chain system to start the manufacturing and delivery process. Business process automation helps companies reduce costs, improve customer satisfaction, speed up business processes, and respond more rapidly to competition.

From an integration point of view, multistep business processes can be streamlined in two ways:

Synchronous Communication: Request and Reply

Certain applications that form part of a multistep business process can be linked together using a request and reply structure. For instance, a company that has set up a Web storefront can check a customer's credit rating and purchase history with the company before permitting the customer to proceed to checkout. In this case, the two steps of the business process, the Web storefront event and the credit rating application, are linked in a request and reply structure; the storefront sends a request to the credit rating application and requires a reply before it can proceed. These two steps in the business process are logically part of a larger composite application and must communicate with each other synchronously.

Interactive composite applications represent the most closely knit integration process. They always work in real time. Composite applications are rapidly growing in popularity as enterprises seek to provide more front-end marketing and service. A composite application generally appears to the end user to be a single Web application, but, behind the scenes, it invokes one or more mainframe transactions or calls to packaged applications or to NT applications.

Asynchronous Communication

Most applications that form part of a multistep business process are not linked in such a request and reply structure and can, instead, be linked together asynchronously.

A multistep process is the most common form of integration because it addresses so many business needs. In a multistep process, the applications are logically independent because each step results from the work of another system earlier in the process. For example, when an order is accepted at the Web store, it is sent to the financial application where it can be tracked and to the supply-chain application where it starts the supply chain planning process. The Web store needs to know only that the message has been delivered to the back-office applications to guarantee once-only delivery of the purchase in the order received in the purchase queue.

The communication between the different applications that makes up the different steps of the business process is performed asynchronously by using messaging middleware. Asynchronous communication has a number of benefits: it couples the two applications together, it isolates the applications from a network or system failure, and it isolates each application from a software failure in the other.

Asynchronous communication is increasingly used to streamline business processes within a company. Though useful internally, it is fundamental to linking together business processes between companies for business-to-business commerce.

To summarize: Enterprise application portfolios are becoming an expanding patchwork of independently designed systems. It is impractical to implement all enterprise-wide business functions using a monolithically designed set of systems. User requirements are inherently too complex and dynamic for any one design team to provide the entire solution. A typical enterprise must deal with applications written in-house many years ago, multiple newer independent software vendor's application packages, end-user client/server applications, and a growing number of Internet and intranet applications. However, unintegrated applications are no longer acceptable: business managers demand an increasing level of integration between their systems. The task of integrating these heterogeneous systems is a central function of the IT organization today. As a result, enterprises must employ a blend of data consistency, application isolation, and multistep business process automation techniques to address their integration needs.

E-Business Integration Technologies and Approaches

In this section, we examine the various technological alternatives available for integration and the integration approaches that these alternatives enable. We have shown that e-business integration consists of three basic kinds of relationships: data consistency and synchronization, application isolation, and multistep business process automation. Now we consider the three primary technological alternatives for e-business integration:

- Data consistency and synchronization technologies that use data replication and database gateways
- Component-oriented development facilities, object request brokers, and synchronous integration methodologies
- Message-oriented middleware and asynchronous integration facilities

Each of these integration technologies is suited to a specific kind of integration problem. No single integration methodology is suited to all integration problems.

We first examine the three different kinds of integration technologies and then discuss how each is suited to solving a specific integration problem. This section includes:

Data Consistency and Synchronization Technologies

Data consistency patterns aim to obtain facts from redundant data that is stored in multiple systems. A number of different mechanisms are used to synchronize data between systems, but they can be broadly classified into two categories:

Data Movement Technologies

One way to synchronize data between different systems is to move data between the systems themselves. In fact, batch data transfer is the default approach to synchronizing data. Traditionally, many companies also use manual processes such as FTP to move data between systems. Database replication technology also synchronizes data stored in databases. Manual processes effectively synchronize data in a batch transfer or in pre-scheduled transactions.

However, as e-business drives the necessity to keep information consistent between systems, frequent real-time intervention is required. As a result, e-business requires that data movement use automated technologies such as database replication. Rather than relying on infrequent data batch runs, you must synchronize data using near-real-time transfer of individual updates as soon as you recognize them.

Heterogeneous Data Access Technologies

An alternative to moving data between systems is to keep data in one place and access it from multiple heterogeneous applications. This eliminates the necessity to constantly synchronize data by moving it between systems, something that is further complicated as systems multiply. To simplify, all data that must be accessed by different systems are stored in one central place to which all applications requiring subsets of the data have access.

Although this method eliminates the need for continual data movement, you must consider three fundamental concerns to determine whether you can solve the data synchronization problem by using a single database:

- The overall performance and scalability requirements for data access: how many different applications need to access the data and how frequently
- Whether the deployment of the applications is centralized or decentralized

  For example, if applications are distributed across a low bandwidth wide area network (WAN), data does not move smoothly or efficiently across a WAN. Thus, in this case, you might have to move data between the systems by batch or manual transfer. Heterogeneous data-access technologies are primarily gateways to enable access to heterogeneous data stores such as mainframe databases, hierarchical data stores, flat file stores, and others.
- Whether the transactions performed involve multistep business processes

  For example, an order may create a series of logically related transactions over a period of many days involving order entry, sales management, manufacturing, and supply chain planning and execution systems. In some cases, batch data transfer technologies are used to automate multistep business processes.

For critical business processes, workflow is automated through workload management tools such as workflow systems. Less critical and less well managed critical processes are often controlled through batch data transfer involving human intervention. As the need for faster end-to-end processing grows, batch transfers can become a bottleneck. Strategies such as end-to-end business process automation usually require messaging facilities to send individual events immediately to other systems. Data movement and data access technologies are best suited to

synchronize information between different systems to provide a consistent global view of the information.

Component-Oriented Development Technologies

In order to isolate applications from each other, limited communication between applications occurs through a small set of well-defined interfaces that remain stable even as the applications change. In the modular development paradigm, components interact through program-to-program communication using well-defined standardized interfaces. Applications do not share data. By isolating program-to-program interaction to a small set of public interfaces, components are able to encapsulate business logic. An important added benefit of modularity is the ability to reuse components: if components are designed correctly, applications can be built by assembling these components in a plug and play fashion.

Three primary component models are used in the industry today:

- The most widely accepted is the Java2 Enterprise Edition (J2EE) or Enterprise JavaBeans (EJB) component model supported by major software vendors including Sun, Oracle, and IBM.
- The Common Object Request Broker Architecture (CORBA) standards committee within the Object Management Group (OMG) is also putting forward a CORBA component model.
- Microsoft has its own COM+ object model.

As applications are increasingly developed using component-based techniques, a fundamental integration issue is the manner in which components communicate with each other. They can do so in two ways:

Synchronous Communication

Each of the three common component models provides its own hosting environment for application components (called a container), which provides a set of services that enable components to operate. These include transaction services, naming and directory services, and brokering and trading services. For EJBs, these services are provided by a container called an Enterprise JavaBean Transaction Server; for CORBA, it is called an Object Request Broker or ORB; and for COM+, these services are provided by the Windows NT operating system itself.

These containers manage communications between components using a synchronous remote procedure call (RPC) mechanism. Synchronous communication is ideal when applications need to be isolated but are related to each other in a request and response structure. Component middleware such as an

ORB is well-suited for such types of integration because it documents program-to-program interface definitions and manages the communications.

Asynchronous, Message-Oriented Middleware

It is technically possible, although rare, to implement a one-way asynchronous event notification between heterogeneous applications using traditional ORB calls. The center of a standard ORB is designed for two-way, request-reply interactions; it lacks the sophisticated messaging facilities required to loosely couple applications, particularly those that facilitate a multistep business process. Message-oriented middleware solutions, particularly the new generation of integration brokers, are much better suited to carry out such integrations.

Message-Oriented Middleware Technologies

As automation of business processes within companies and business-to-business commerce increases, a new generation of middleware technology is emerging. It is based on asynchronous communication that loosely couples these applications and businesses together. The fundamental principle of messaging is to isolate information providers from information consumers so that an application can be added, dropped, or changed without affecting any other system.

Message-oriented middleware enables applications and business processes to communicate by sending a message from one application to the other. Since the applications are mission-critical, the middleware provides features such as guaranteed once-only delivery of the message, store queuing, and forward queuing. Additionally, these messaging platforms add sophisticated message routing and distribution facilities. These include:

- Content-based routing in which the message is sent to a different location based on its content
- Topic-based or subject-based routing in which the message is sent to a different location based on its subject
- Publish-subscribe routing in which a sender simply publishes a message to a queue to which subscribers who are interested in the message subscribe. The publisher does not know who will receive the message.

Most component models such as Enterprise JavaBeans are now adding asynchronous communication interfaces to enable them to communicate with each other in a loosely coupled fashion. You can use basic message-oriented middleware both within a distributed application and to integrate one application to another because it is inherently connectionless. Asynchronous messaging facilities are best

suited to applications that must be loosely coupled together such as when they form part of a multistep business process (and are not related in a request-response fashion).

Messaging middleware connects dissimilar applications in a fundamentally different way than do direct server-to-application gateways. Direct gateways are best at tactical, request-reply interactions especially when extending one or two back-end applications with a Web front end.

Message-based solutions are best suited for asynchronous applications that require either data consistency or multistep process applications and for systematic composite applications that have multiple heterogeneous participants. Messaging introduces an incremental layer of communication semantics and administration. This complexity is not necessary for some projects. However, messaging provides a rich, comprehensive infrastructure that handles consistency and multistep and composite patterns in one solution. Components dominate intra-application architecture and often are used to connect into message-based integration infrastructures.

Ultimately, message-centric integration connects all applications to each other through a general purpose enterprise hub. All applications publish information to this integration hub without needing to know where to send it, who will receive it, or what format the receiver prefers. The whole application portfolio remains flexible because connection logic and delivery instructions reside in the infrastructure rather than in the applications themselves.

Messaging enables a program to act as a producer by placing a message in a queue and then proceeding with its work. The queuing system reliably delivers the message to the appropriate recipient. The recipient, acting as a consumer, retrieves requests from the queue and acts on them. By isolating requests for service from the supply of services, messaging increases efficiency and provides the infrastructure to schedule complex tasks. With messaging, programs do not communicate with each other directly. They are disconnected from each other and communicate through the messaging system that serves as a communication hub among different application programs. Messaging, therefore, provides a useful paradigm for getting many programs to communicate with each other.

Three specific application design issues frequently motivate the use of a messaging service for interapplication communication.

No Request-Response Requirements

Messaging is ideally suited for applications in which a program can proceed with its own work after sending a message to another program: the first program does not need to wait for a response from the second program to proceed. It is also suited for applications that can continue their work until a message must be retrieved.

If the first program requires a response to proceed, messaging is an inappropriate communication mechanism. A synchronous communication mechanism such as Net8 or CORBA RPC is preferable. In a synchronous mechanism, the first program sends a request to the second and then waits until the second sends a reply. The first uses this response to carry out further processing.

Messaging, in contrast, is suited to applications that do not require such a relationship; the first simply places a message on a queue and continues with its work without waiting for a response from the second.

Note that you can use these two models, synchronous communication and messaging, together in the same application. For instance in a shipping application, the order entry program communicates with another customer management program to check the validity of a customer before accepting his or her order. This requires synchronous communication, since the order entry program requires the reply from the customer management program before continuing to process the order.

However, when the order is complete, the order entry application notifies the shipping program that an order must be sent to the customer. This communication is best done using messaging since the order entry application does not need to wait for the response from the shipping program to further process the order.

Isolated Processing

The second factor that could influence you to use a messaging system for inter-program communication is the deployment architecture of the application. For synchronous communication to work, all programs must be running and available at the same time. The network that connects the programs must be available, the systems that run the programs must be up, and all the programs must be available simultaneously. If any nodes of the deployment environment are unreliable, then messaging provides a more robust solution. Messaging removes the time-dependent relationship between programs. As a result, applications are less vulnerable to program failure. For deferred execution to work correctly in the event of network, system, and application failures, messages that constitute requests for service must be stored persistently and processed exactly once. Being able to preserve messages is fundamental in an enterprise messaging infrastructure for four reasons:

- **Inability to process messages as they arrive:** Applications must deal with many unprocessed messages arriving simultaneously from clients. They might not have the resources to process all the requests immediately. A messaging system must be able to store the message in a persistent queue and deliver it later when the recipient can process it.
- **Message scheduling:** Messaging systems also require message persistence so that they can deal with priorities: messages arriving later may be of higher priority than messages arriving earlier; messages arriving earlier may have to wait for messages arriving later before actions can be executed. Such message priorities may also change over time; messages in a particular queue can become more important than messages in other queues during certain time windows, for instance. Message persistence enables messages in high priority queues to be processed first, while low priority messages can be stored and processed later without interfering with high priority messages.
- **Message auditing:** Message persistence is also critical because the control component of the message can be as important as the payload information itself. For instance, the time that a message is received and dispatched can be a critical part of a message. In an e-commerce environment, message persistence stores information on orders from various customers and can be queried to identify periods of peak demand or to determine the status of an order. The message, therefore, can remain important even after it has been executed. A persistent messaging store is critical to ensure that information can be warehoused and queried or audited.
- **Failures:** The communication links between messaging clients might not be available all the time or might be reserved for some other purpose. If the system cannot process messages immediately because of either a lack of processing resources or a failure, the messaging system must be able to store the message persistently and deliver it when the resource is available. Such guaranteed message delivery of each message to each recipient exactly once is critical when integrating enterprise applications.

2

Methodology and Solutions

This chapter introduces you to the methodology of e-business integration and offers examples of ways in which Oracle Integration Server meets integration challenges. The topics in this chapter are:

- Selecting the Appropriate E-Business Integration Methodology
- Application Integration: The Solution Spectrum
- Business-to-Business Integration

Selecting the Appropriate E-Business Integration Methodology

We have examined the three primary integration technologies available: data synchronization technologies, component-oriented development technologies, and message-oriented middleware. Now we consider how these technologies optimally map to the three fundamental integration problems: synchronizing data between systems, isolating applications from each other, and automating multistep business processes within companies and between companies.

Although the choice of methodology usually requires a close study of the specific environment in which integration is implemented, you can use certain broad architectural principles to guide integration decisions.

This section includes:

Synchronizing Data Among Systems

The fundamental architectural choice in determining how to synchronize data among systems is whether you need to move data between different systems or whether you need to provide data access to a variety of system from a central location. If you must provide data access, then the ideal choice of integration technology is to use gateways to access the databases and legacy systems in which the data resides.

If you must move data between systems, you can use two different mechanisms: database replication and asynchronous messaging. Three factors influence your choice of technology:

- Firstly, if data can be synchronized in a relatively simple fashion such as when extending one or two back-end applications to a Web front end, simple database replication of information is sufficient. Similarly, if distributing information from a central database to a number of smaller workgroup databases, database replication is sufficient. In these cases, message-based solutions introduce an additional level of complexity (for instance, the need to deploy specialized middleware) and are better suited for more complex applications.
- Secondly, in most cases in which replication can be used, the sender and the recipient of information must be homogeneous databases, for example, both Oracle databases. The databases must have a similar schema representation. If

data needs to go through a complex transformation as part of data movement, then asynchronous messaging technology is probably more appropriate.

- Thirdly, in order to use database replication, the receiving application must provide the sending application direct access to its schema. Two issues are involved with providing such direct access to the schema: because the two applications do not communicate through a clearly defined set of interfaces, direct schema access violates encapsulation and component-based development. Further, direct schema access can circumvent application-level security policies.

Despite these concerns, in a number of cases where simple data synchronization between two data sources is the only integration requirement, using database replication simplifies how you build and deploy the applications and is probably the most appropriate choice.

Isolating Applications and Businesses from Each Other

If the integration scenario is primarily focused on isolating applications and businesses from each other, data synchronization technologies such as database replication and database gateways are not appropriate. To isolate applications from each other, communication between the applications must be limited to a standard set of well-defined public interfaces. Data synchronization technologies, by definition, violate encapsulation and as a result are not suited for application isolation.

When you need to isolate applications from each other while facilitating communication between them, the primary choice is between a synchronous or an asynchronous communication facility. Three factors determine which communication mechanism to use:

- Firstly, if the two application components are part of a larger composite application and work together in a request-reply structure, then a synchronous component-oriented middleware facility such as an Object Request Broker (ORB) and a communication protocol such as CORBA IIOP is the best choice. In this case, both applications should be wrapped as CORBA services with well-defined public interfaces defined in the CORBA Interface Definition Language (IDL) and they should communicate through a middleware hub known as an ORB. The ORB provides services to the two applications including:
  - Registering them as services

  - Making the public interfaces of each application accessible to the other applications
  - Routing each protocol request from one application to the other
  - Discovering the object by looking up its location on the network
  - Activating the object when the request is received
- Secondly, if the two application components are not related in a request-reply structure but rather form steps within a multistep business process, then an asynchronous message-oriented middleware solution is the most appropriate choice. The primary choices in such a situation depend on the complexity of the messaging environment that needs to be deployed and on the messaging architecture. These choices are similar to those that need to be made in automating multistep business processes and will be discussed in the next section.
- Thirdly, if the integration requirement is to connect the business processes of two companies using the Internet, then synchronous integration is not feasible and loosely coupled message-based integration is the only solution.

Automating Multi-step Business Processes

Automating a multi-step business process requires applications to communicate with each other in a loosely coupled structure. The only appropriate choice of integration technology in this case is asynchronous message-oriented middleware. In choosing to deploy a middleware solution, the primary architectural decisions that you make depend on the complexity of the integration problem. There are four key issues that you must consider:

- Integration Topology
- Messaging Architecture
- Data Transformation
- Business Process Management and Workflow

Integration Topology

When deploying messaging middleware, you must determine whether to use a point-to-point interface to link applications or to use a hub-and-spoke architecture to link applications. Although point-to-point connectivity is simple when connecting two applications, it quickly becomes unmanageable if more than two applications need to connect.

In such cases, use a hub-and-spoke architecture. In this architecture, applications are not connected directly with each other. Instead, each application is connected to a hub that provides connectivity between all the applications. A change or upgrade to any one application changes only its relationship with the hub and does not affect all the other applications with which it must be integrated.

Messaging Architecture

Next, you must determine the messaging architecture to use based on three questions:

Must messages be stored persistently? If messages are business critical and must be audited and tracked, for example, to resolve disputes between companies or to track information flows, the message header and contents must be stored persistently in a database.These messages can be warehoused and analyzed using standard decision support tools. If the messages are not business critical, then the message brokering facility can provide volatile queuing facilities.

Must messages be propagated with guaranteed delivery? If messages are business critical or if the messaging system connects two mission critical applications, then the messages need to propagated with guaranteed once-only, in-order delivery; otherwise messages can simply be propagated impermanently.

How must messages be routed? Messages can be routed between applications and business processes based on their subject or topic, their content or payload information, or by using the publish-subscribe method.

In most e-business integration scenarios, messages are routed based on their content: in most cases, invoking a specific workflow to process the message in a specific way before sending it to an appropriate destination.

Data Transformation

Applications that form part of a business process usually store and manage data in different data formats. For instance, the Oracle E-Business Suite of packaged applications stores and manages data in SQL format. SAP applications operate on data in iDOCs format, which is a derivative of ASCII.

Four data transformation issues must be addressed when connecting one application to another:

- **Datatype transformation** means converting data from the format of the originating application to that of the target application. For instance, when connecting an Oracle customer relationship application to an SAP application,

Oracle data types must be converted from SQL format to an ASCII format appropriate for iDOCs.

- **Semantic transformation** means converting a customer name from an Oracle format into an SAP format. For instance, Oracle represents the customer's name in three fields: a first name, a last name, and middle initial. SAP represents the customer's name in a single field: a last name followed by a comma followed by the first name. A semantic transformation must be applied to concatenate the last name followed by the first name.
- **Functional transformation** means mapping the business events in the sending application to the business events and interfaces in the receiving application. For instance, when a purchase order is sent from an Oracle application to an SAP application, a specific set of business events must start and a specific set of public interfaces must be invoked. Such functional transformation information must be captured in order to propagate a message from one application to another.

  There are two commonly used approaches to data transformation: The first converts the data directly from the format of the originating application to the format of the target application. The second converts the data first into a canonical, intermediate format before converting it into the format of the target application. Although direct conversion is faster, converting to an intermediate format isolates one application from changes in other applications. For instance, when you connect an Oracle CRM application with a Baan, SAP, or PeopleSoft application, mapping information to an intermediate representation isolates the Oracle CRM application from an upgrade to the Baan application. The only change necessary is to modify the mapping between the intermediate representation and the Baan application.
- **Locational transformation** means determining the location where data transformation is conducted. In some cases, data transformation is done close to the application at the spoke. The hub simply sends the transformed message from one application to another. In most cases, however, data transformation must be conducted in the hub:
  - A message that must be audited is stored in its pretransformed form.
  - A message that must be sent to many different target applications is transformed within the originating application into an intermediate form that is sent to the hub. The hub then applies various transformations to convert the information into the format of each target application.
  - A message that must be routed to different locations based on its content is routed by the hub after it is transformed in the hub. However, this depends

on the capability of the hub itself and on the kinds of services it provides to the applications that require connection.

Business Process Management and Workflow

Finally, you must decide whether messages can simply be sent from one application to another or whether they require processing before they are propagated. For instance, when a company sends a purchase order from its supply chain application to the procurement system of another company, the purchase order can require the approval of a purchasing manager at the originating company. If so, the messaging middleware must invoke a workflow application before forwarding the message to the trading partner. In this case, the messaging platform requires a facility for business process management of workflow.

Application Integration: The Solution Spectrum

Remember that all integration scenarios involve three fundamental integration problems: synchronizing data between systems, isolating applications and business from each other, and automating multistep business processes. Integration requires a range of different technologies, each of which is appropriate to a particular type of integration problem.

Oracle Corporation recognizes that integration is not a single, narrowly defined problem that can be solved by any single technology. A complete solution to this complex problem requires a variety of technologies, all of which must be seamlessly integrated. This section clarifies your choices by illustrating some typical integration scenarios. These should help you gain an appreciation of the different types of integration technology that constitute a comprehensive solution.

This section includes:

- Data Integration
- Application Integration
- Business Process Modeling and Execution
- Business Process Intelligence

Data Integration

Scenario: Your enterprise has a number of applications, each with its own databases. The applications change the information in their databases frequently and independently of each other.

Problem: How does an application get the most current information residing in the database of another application? For example the order entry application needs to access the same customer data as the accounts receivable application in order to use the address.

Solution: **Data Access Gateways and Replication:** Gateways enable applications to easily and directly access other databases to get the information they need. Replication automatically synchronizes the information in multiple databases so that each database has the most current information. Changes in any database are immediately reflected in other databases.

Application Integration

A business process involving multiple applications requires integration of application logic and application functionality. To facilitate this integration, applications must communicate to exchange important business information. The two different communication models are synchronous and asynchronous communication.

Synchronous Communication with Functional Interfaces

Scenario: You want to create a front-end e-commerce Web application that accepts orders. Before accepting the order, the application requires credit card authorization, customer credit rating, inventory levels, delivery schedules, and pricing. If any of these services is not available, the order application cannot complete the transaction. This might mean asking the customer to try again later.

Problem: The services required for order entry are provided by other dedicated applications. So, how does the e-commerce application access these services? How can these applications be integrated to implement and automate the business process?

Solution: **Synchronous, request-reply protocol based on functional interfaces:** Each application offers services by defining a set of specific functions as public interfaces. To request these services, applications invoke the corresponding

interfaces. Examples of request-reply protocols based on functional interfaces include Remote Procedure Calls (RPC), Common Object Request Broker (CORBA), COM, and Java Remote Method Invocation (RMI). Standards for semantically richer interfaces, specifically aimed at distributed enterprise transactional applications, are emerging with component models such as Enterprise JavaBeans and COM+.

Asynchronous Communication with Message-Based Interfaces

Scenario: You want to implement an end-to-end order-fulfillment process that involves a number of applications such as order entry, manufacturing, inventory management, distribution, and billing. The order fulfillment process originates with the order-entry application with which a customer places an order. After that, business objects and events must flow between the applications.

Problem: These applications are distributed across a wide area network, are owned by different organizations, and have heterogeneous internal architecture. They were designed as single-function, standalone applications, with no plans for integration. The network that connects the applications might be unreliable. The organizations that own the application might change, relocate, or replace the applications at will, without informing the organizations owning the other applications. Furthermore, one application does not require a response from another in order to proceed. Hence, synchronous communication integration is not a viable option.

Solution: **Asynchronous communication with message queuing:** Applications communicate with each other by exchanging information as messages through queues. Each application defines a set of messages that it accepts as input and a set of messages that it publishes as output for other applications. The messages represent business objects like customer records or business events like a new shipment requests. Asynchronous communication with message queuing enables loose coupling, so that individual applications are completely isolated from application, network, and system failures.

An example of a message queuing interface is the industry-standard Java Messaging Service (JMS).

Business Process Modeling and Execution

Scenario: Consider the order-fulfillment scenario described in the previous section. Assume that the applications are integrated and that the entire business process is automated.

Problem: Business managers decide that the business process must be modified. For instance, the order-fulfillment process moves from a clerk-oriented system to a Web-based self-service system that requires a different approval process. The flow of the process must be changed and new applications must be added to the process. How quickly can the business decision be translated into reality? How do you minimize the time that elapses between decision making and implementation?

Solution: **Business process modeling and execution:** Firstly, a business analyst models the entire business process using a graphical modeling tool. Then, a technical analyst fills in the details and maps the model to the underlying integration infrastructure. Next, the model is validated and generated in a repository. Finally, it is executed by a business process coordinator. To change the process at any time, only the high-level model requires change. You can quickly implement the entire cycle, from identifying the change to executing the new model.

Business Process Intelligence

Scenario: Same as in the previous section.

Problem: The end-to-end business process is not performing to expectation, even though each of the individual applications is finely tuned. A holistic view of the business process is not available. Standard system management and monitoring tools provide performance metrics only about individual components, not about the entire process. To complicate matters further, performance varies daily and seasonally with market variations, thus making it hard to pinpoint the exact cause of the problem. So, how do you identify the inefficiencies and bottlenecks to help you to deploy your resources intelligently?

Solution: **This solution utilizes Business Process Intelligence as its strategy.** The only way to get a holistic view of the entire business process is to track each transaction end-to-end. This implies tracking all the information (data, messages, and business events) in order to reconstruct the transaction later. Further, you must collect this information over an extended period of time, and it must be warehoused

and analyzed to gather intelligence about the business process. The accumulated information can be mined to discover patterns and to provide insight into how resources can be optimally deployed.

Business-to-Business Integration

The previous section discussed the range of technologies required for a complete application integration solution. The focus of most integration projects in enterprises to date has been on integration of applications within the enterprise. The few projects that involved integration beyond the enterprise with partners and suppliers were limited in their scope and used simple mechanisms like FTP and e-mail. The real mission-critical, inter-enterprise transactions are conducted over private value-added networks (VANs) using proprietary protocols like EDI, HL-7, and SWIFT.

e-Business integration requires extensive, flexible, and dynamic cooperation and collaboration with customers, suppliers, and partners. Some examples of new business models that require this type of integration include:

- Corporate Web-based self-service applications that requires users to have a unified and up-to-date view of business processes
- Virtual, extended supply chains
- Procurement through online marketplaces or business-to-business commerce exchanges
- Dynamic order fulfillment
- Hosting applications by Application Service Providers

The integration must take place over the low-cost and ubiquitous Internet, so that an enterprise can choose from a worldwide selection of partners, suppliers, and vendors. This necessity places additional requirements on a solution. These requirements include:

- **End-to-end security:** Business partners must have complete confidence that a transaction they conduct over the Internet is completely secure throughout its entire life cycle. Information must be secure while it resides in databases, is transmitted over networks, and is processed by applications.
- **Auditing and tracking:** Traditionally, private network vendors have provided value-added services such as auditing and tracking of all business transactions between partners. The Internet provides a cheaper and ubiquitous alternative to

private networks. However, an integration solution that exploits this inexpensive transport must provide auditing and tracking services.

- **High availability:** When conducting business online and globally, enterprises must be open for business all the time, with no downtime. Therefore, the integration software linking these companies and their applications is mission-critical and needs to be highly reliable, scalable, and always available. The integration software is at least as mission-critical as the applications it links.
- **Complex business processes:** In terms of duration, number of organizations involved, and number of applications involved, e-business integration must deal with complex business processes. The highly autonomous, heterogeneous, and distributed nature of these applications places additional requirements on all aspects of an integration solution.
- **Internet standards:** By their very nature, business-to-business transactions must take place within agreed-upon standards. Protocols specific to one enterprise or to one integration vendor do not necessarily scale across a broader set of partners, each of which might have its own integration solution. General purpose software must support all the standard Internet protocols such as
  - HTTP as the wire or transport protocol
  - XML for message formats with standard definitions for common business objects such as those proposed by the Open Applications Group
  - Business process protocols such as Open Buying on the Internet (OBI) and RosettaNet

3

Overview of Oracle Integration Server

This chapter provides an overview of Oracle Integration Server (OIS). OIS is a complete suite of software that addresses the e-business requirements for integrating different components. It has the breadth of functionality, the robustness, and the tools to address the most demanding and complex integration scenarios.

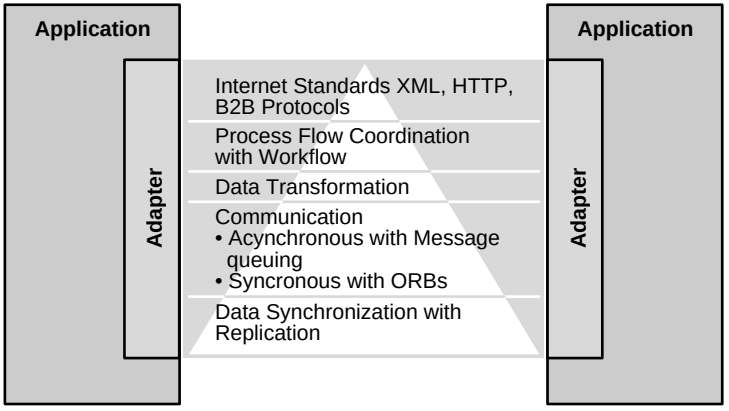
This chapter includes the following topics:

Introduction to OIS

OIS addresses a wide range of integration problems ranging from simple front-office to back-office integration within a company, to creation of a strategic IT integration infrastructure within a company, and finally to business-to-business integration. The functionality provided by the Oracle Integration Server includes:

- Data Integration
- Replication
- Application Integration
- Business Process Intelligence
- Data Transformation
- Application Adapters
- Business Process Modeling and Execution
- Systems Management

Figure 3–1 Oracle Integration Server facilities



Data Integration

Oracle Data Access Gateways enable applications to access and manage data from heterogeneous data sources

- Procedural Gateways provide procedural access to non-Oracle transactional systems such as CICS, IMS/TM, IDMS-DC, and so on.
- Transparent Gateways provide SQL access to over thirty non-Oracle databases such as Sybase, Informix, SQL Server, and DB2.

Replication

The Oracle Integration Server includes Advanced Replication, which provides powerful and flexible replication capabilities for synchronization of data across multiple distributed databases**.** The key features include:

- Replication of tables and related objects for each application as a single group
- Full and subset table replication
- Automatic conflict detection and resolution, with user-selected conflict resolution rules
- Synchronous and asynchronous replication, with user-definable replication intervals at replication group level
- Automatic parallel data propagation for improved performance

Advanced Replication is centrally managed, configured, and maintained with the Oracle Replication Manager.

Application Integration

Synchronous Communication

The Oracle Integration Server supports synchronous-communication-based integration with a built-in Java-based CORBA 2.0 compliant Object Request Broker (ORB) and an Enterprise JavaBeans (EJB) server.

Additional CORBA facilities in the Oracle Integration Server include:

- **Transactions**: Provides a Java Transaction Service (JTS) and CORBA Object Transaction Service (OTS) for development of transactional CORBA applications.

- **Directory naming:** Provides standard Java Naming Directory Interfaces (JNDI) and COSNaming interface to the Lightweight Directory Access Protocol (LDAP), the industry-standard directory service
- **Object Adapter:** Implements an Object Adapter for persistent CORBA objects, which provides two important functions. Firstly, it serves as a directory of CORBA Objects. Secondly, it helps to locate and load CORBA objects upon initial activation by CORBA clients.
- **Java and CORBA services:** Provides a number of features that make it easy for Java programmers to develop CORBA services. Caffeine is a direct Java-to-IIOP mapping tool that eliminates the need for IDL. Other tools such as `java2iiop`, `idl2java`, and `java2idl` simplify application development.
- **Security**: Features include encryption through Secure Sockets Layer (SSL) over IIOP, authentication using a username and password, and access control using roles and privileges.

The Oracle JDeveloper is the development tool used for developing EJB components. Caffeine is the Java-to-IDL compiler. Oracle Integration Server can interoperate with the Microsoft Com+ component model through third party bridges.

Asynchronous Communication

Oracle Integration Server provides support for asynchronous integration of application through its Advanced Queuing feature. Advanced Queuing is a full-service messaging queuing system. Some of the key features include:

- **Guaranteed, exactly once delivery:** Advanced Queuing guarantees that each message is delivered to its destination exactly once within the specified time interval, despite possible network, system, and application failures.
- **Subject-based and content-based publish and subscribe:** Publish-and-subscribe is a communication model that enables loose coupling of applications. The two well-known publish-and-subscribe models are subject-based and content-based. Advanced Queuing supports both.
- **Propagation:** Applications can enqueue messages to a local queue and specify the remote queue destination for the message. The Advanced Queuing propagator transparently moves messages between local queues and remote queues, enabling communication between distributed applications.
- **Java Messaging Service:** Advanced Queuing is one of the first message queuing systems to implement the industry-standard Java Messaging Service

(JMS). Along with JMS, you can access Advanced Queuing functionality through PL/SQL, C, C++, and Visual Basic.

- **Message Management:** This unique feature of Advanced Queuing, derived from its fundamental ability to retain messages, is discussed in the next section in the context of Business Process Intelligence.

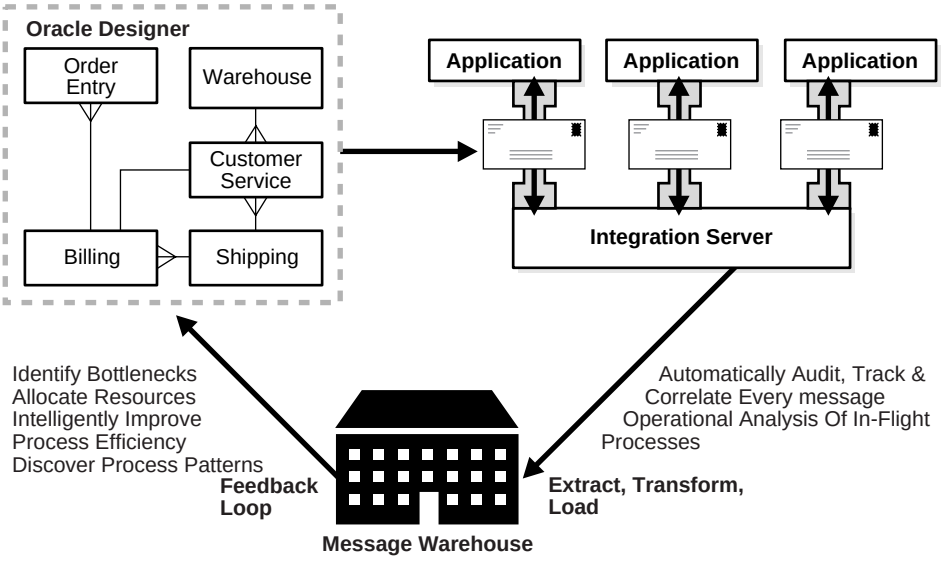
Business Process Intelligence

The Oracle Integration Server, along with related tools, provides enterprises with the ability to analyze and gather intelligence about their business processes.

The Advanced Queuing feature of the Oracle Integration Server automatically retains and tracks every message that flows through its queues. It tracks all relevant details of the message: enqueue times, dequeue times, destinations visited, identification of transactions that processed the message, and the relationship between the message and exception conditions. It retains this information in queryable queues that are accessible through standard query and reporting tools. You can exploit this information to improve business processes in two ways:

- **Analysis of live or in-process transactions:** You can determine the complete history and current state of any transaction from information retained in the queue. For example, you learn how it arrived at a particular state and determine the next processing step. In exception situations or stalled processes, the administrator can ascertain the exact cause of the problem and obtain hints on rectifying it and restarting the process. This information can be combined with statistics provided by Advanced Queuing to improve performance of the business process in real time.
- **Message warehousing and analysis:** The tracking information can be retained and accumulated over an extended period of time to create a message warehouse from which you can query messages using standard SQL. The information can be extracted, transformed, and loaded into an Oracle data warehouse for analysis and mining. Since all the information required to completely reconstruct every business process is available, you can determine exactly how the system performs under a variety of changing conditions.

Figure 3–2 Business Process Intelligence to Improve Process Efficiency



The warehouse can answer such questions as:

- What time elapsed between the placing of a customer order through the Web site and the delivery of the product to the customer?
- How did this elapsed time vary over each of the last twelve months?
- How did this elapsed time vary over the course of a day?
- Over the last six months, on the average, which step in the process took the longest to execute?
- Where is the best place to deploy additional resources?
- Rated by response time, who is your best supplier?
- How does this supplier rate on a per-component basis?

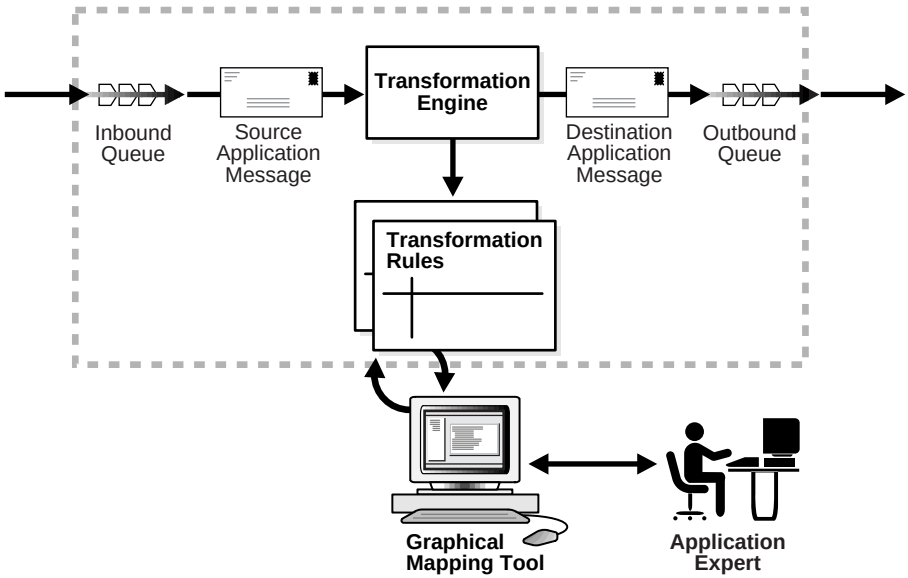
Oracle can create extremely large message warehouses and perform complex analytical queries on them with fast response times. Oracle tools to help you in information analysis include Oracle Report, Oracle Express and Oracle Discoverer.

Business process analysis provides business managers with the information they need to make intelligent decisions about their business, streamline processes, deploy resources more effectively, and increase overall efficiency.

Data Transformation

Oracle Integration Server connects heterogeneous applications and smooths the flow of data, messages, business objects, and business events between applications. Each application has its own definitions of business objects, schema, and message formats. To enable applications to communicate, the Oracle Integration Server provides transformation services that convert the output format of one application into the input format of another.

Figure 3–3 Encapsulation of Messages from Source to Destination



OIS offers a design-time visual tool for definition of datatypes and transformation between different types of data. Transformations that can be defined using the visual tool include string operations, mathematical operations, date formatting, and shape changes. You can provide your own transformation routines through call-out mechanisms. The tool can import type definitions from the repositories of Oracle

and SAP applications. It also can import XML Document Type Definition (DTD) for transformation of XML messages. Future releases of the Integration Server will also support XSL-T-based transformation of XML messages.

The data types and the transformation between types are design-time activities. The results are stored in the server repository. The runtime transformation engine in the Oracle Integration Server uses this information to transform data as it flows between applications. The Oracle Integration Server defines an open API that enables you to use any other compatible transformation instead.

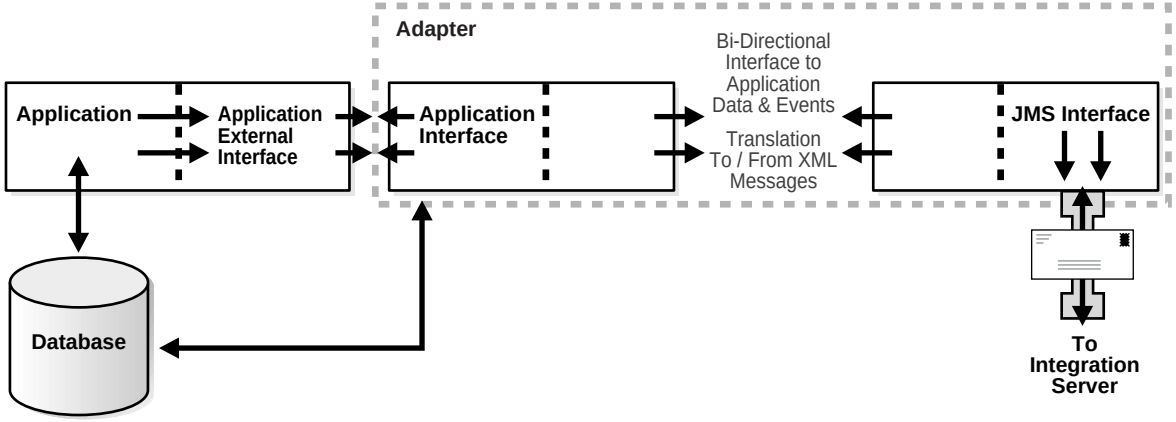
Application Adapters

Oracle Integration Server is a non-intrusive integration solution. This implies that it requires no reengineering of existing applications, or only minimal reengineering. Because most applications were not designed to be integrated, they have no simple way of communicating with the Oracle Integration Server. Thus, adapters are required to bridge between applications and the Integration Server.

Functionality: Adapters interface with applications to do the following:

- Submit and extract well-defined business objects to and from the application, using the native interface of the application
- Detect business events occurring in the applications (such as creation of a new purchase order) and publish this information for other applications

Figure 3–4 Overview of the Architecture of an Adapter



- Optionally translate the business objects and events into an XML-formatted message
- Perform validation and error checking as required
- Transmit the information to the Integration Server using one of the standard interfaces such as JMS

Deployment: The architecture of adapters varies depending on the application with which they interface. Typically they run on a computer in close proximity to the application. In some cases, they run in the execution environment of the application itself, such as in the Oracle Application / Database Server. They can also be deployed in the execution environment of the database, especially if they are implemented in Java or PL / SQL. In all cases, the goal of Oracle Corporation is to provide a centrally managed environment for hosting and executing adapters.

Adapter SDK: Oracle will partner with a number of vendors to provide adapters to packaged and industry-specific applications. Oracle will also provide an Adapter SDK to simplify the development of adapters for custom and legacy applications. The Adapter SDK will provide libraries to deploy triggers, parse XML messages, and use JMS.

Business Process Modeling and Execution

Oracle Integration Server includes a graphical, visual process modeling tool. You can use the tool in scenarios like these:

- A business analyst describes the business process graphically using the industry-standard Universal Modeling Language (UML) activity diagrams. At this stage, the applications involved in the process are identified and the flow of the process is detailed. The model is generated persistently into the server repository.
- A technical analyst maps the high-level model to the underlying integration infrastructure. The details of the application interfaces, business events, transformations, and messages are filled in. The completed model is again stored in the repository.
- The model is validated and made ready for execution by the runtime execution engine.
- Either the business analyst or the technical analyst can modify the runtime process using the tool. The Business Process Coordinator supports multiple models simultaneously.
- The tool provides a holistic view of the business process by enabling you to visualize and understand the overall functionality.

Figure 3–5 Modeling Steps

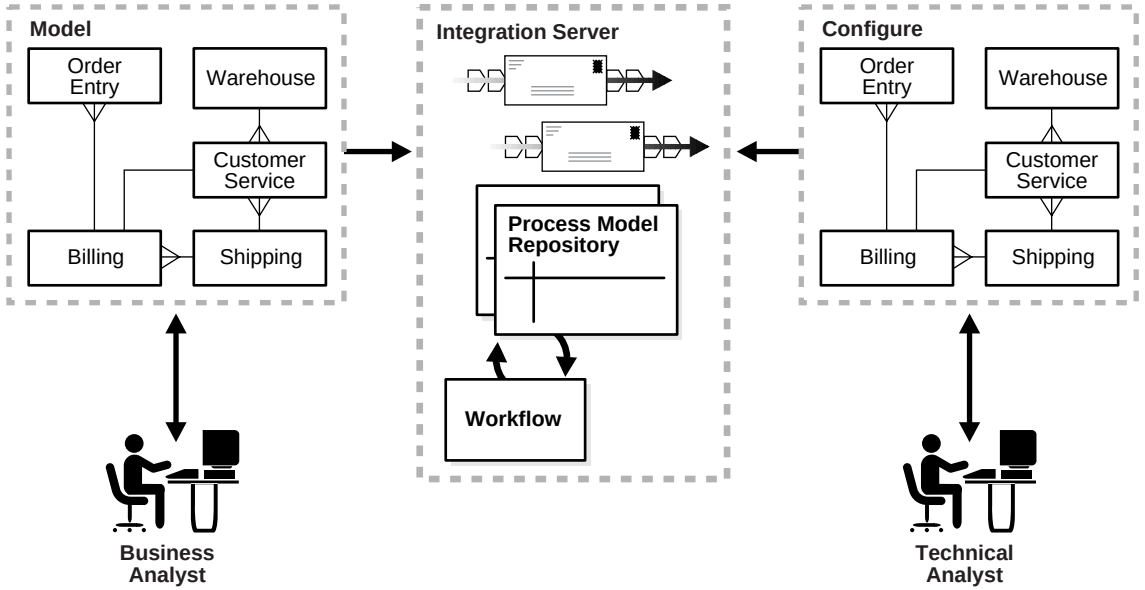
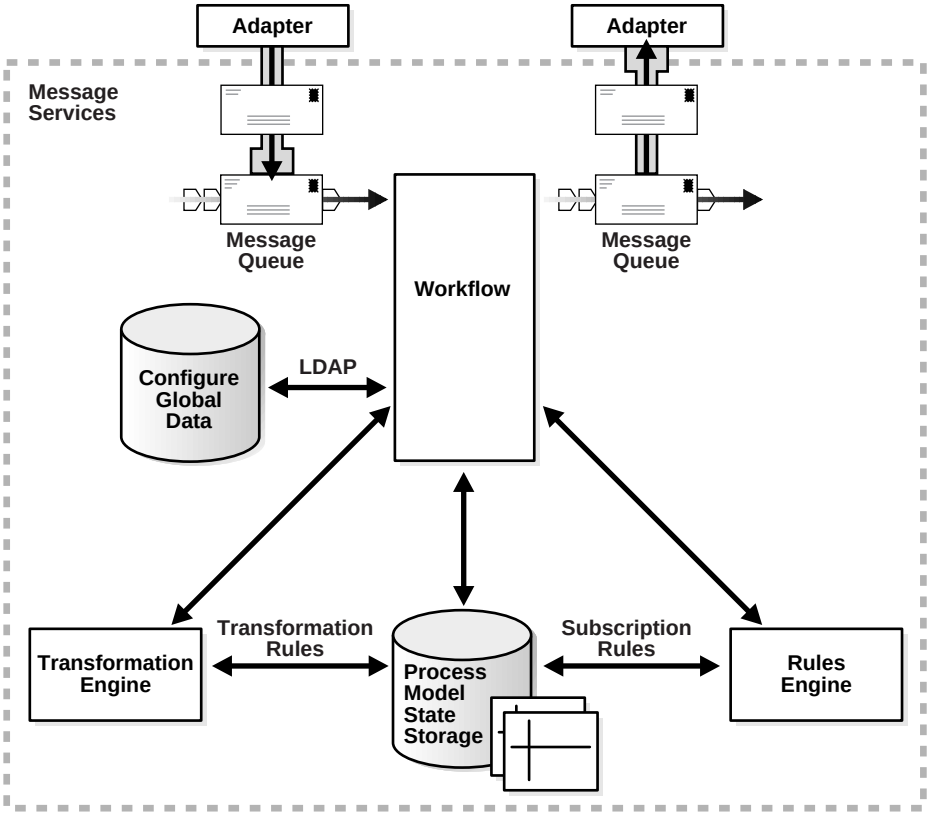


Figure 3–6 Workflow and its interaction with other components



Execution Engine: The validation model is executed by a runtime engine called the Business Process Coordinator. The Oracle Business Process Coordinator maintains the transition state information for all the processes it is executing. It uses all the other functional components to complete its task. The Oracle Business Process Coordinator is unique in that it can automate processes that require human intervention (traditional workflow) as well as those that do not (system-to-system processes).

Systems Management

The Oracle Integration Server is managed and monitored by the Oracle Enterprise Manager (OEM). The OEM is a systems management tool that can also monitor Oracle applications and instances of the Oracle Databases / Application Servers. It enables an administrator to manage a distributed environment from a single, central console. It has been enhanced to manage all the moving pieces of an integrated environment. Enterprise Manager manages system objects at three levels of granularity:

- **Queues, messages, queue propagation:** start, stop, schedule propagation, view statistics
- **Individual business processes:** start, stop, resume, query
- **System processes:** Integration Server, Message Broker, adapters, applications and databases

Oracle Integration Server Design Objectives

The previous sections provided a high-level functional overview of the Oracle Integration Server. This section describes the design objectives of the Integration Server to help you better use and benefit from it. This section contains:

Strategic Infrastructure, Not Tactical Point Solution

Most integration products on the market are aimed at solving a specific integration problem, for instance, connecting a particular front-office application to a particular back-office ERP application. Their value is the quick integration of two applications to solve an immediate business problem. Some products are even more limited: they just specialize in integrating a specific transaction between two applications.

Although the Oracle Integration Server can solve specific problems, it is designed to go beyond point solutions and to encourage enterprises to view integration as a strategic component of the IT infrastructure.

Oracle Corporation believes that enterprises should begin with an integration architecture designed independently of any specific application. They should

choose a product that meets both their current and their future needs. The Oracle Integration Server is designed to be the integration backbone for the entire enterprise. It has all the characteristics expected of enterprise-quality software: robustness, scalability, open architecture, adherence to industry standards, and management and development tools.

Choose And Use As You Go

The Oracle Integration Server is a comprehensive solution that offers a number of technologies to help you solve different integration problems. The Server offers great breadth of functionality within each of these technologies.

Oracle Corporation does not restrict you to the use of any particular technology: you can choose the technology that exactly solves your problem. All the technologies are seamlessly integrated. However, you can use Oracle technologies independently of each other, thus avoiding useless complexity when you choose a particular technology. For example, you can use the Advanced Queuing feature for asynchronous communication without knowing anything about the synchronous ORB-based communication infrastructure or about Advanced Replication.

This is possible because the Oracle Integration Server has been designed from the ground up to offer you a full spectrum of technologies. Most integration solutions on the market originated in niche technologies such as message-oriented middleware, publish-subscribe engines, data transformation, and workflow. The functionality of these products outside of each one's core competency tends to be weak. These products compensate for their weakness by loosely integrating with partner products. The result is a product that forces you to address every type of problem by using the same technology.

Mission-Critical, Enterprise-Wide Integration

The Oracle Integration Server is designed for mission-critical enterprise-wide business implementations. Towards this end, it offers two major features:

- High reliability, availability, scalability (RAS)
- Enterprise-wide management.

RAS Capabilities

The Oracle Integration Server is based on the proven process architecture and runtime environment of the Oracle8*i* platform and transparently inherits all the RAS characteristics from it. Oracle8*i* is proven in large-scale online transaction processing applications (OLTP) that support thousands of concurrent users and

extremely high transaction volumes. It scales linearly in a both symmetric multi processor (SMP) architecture as well as in clustered architectures. Oracle8*i* has a number of high reliability solutions including hot stand-by servers, transparent application failover, and fast recovery from failures.

Management

Oracle Corporation recommends a distributed hub-and-spoke deployment model to simplify integration management. We believes that, as with database servers, it is preferable to consolidate all integration logic into a few large servers, rather than to distribute it over many small servers. The Oracle Integration Server supports such an architecture with the high RAS capabilities described in the previous section.

Many products try to compensate for their limited RAS capabilities by recommending a highly distributed bus architecture. It has been repeatedly proven that such an architecture leads to extreme complexity in management and is therefore not suitable for mission-critical environments. However, note that the architecture of the Oracle Integration Server does not preclude deployment in a distributed **bus** topology.

Leveraging Your Investment

Most enterprises have invested in developing skills and expertise in Oracle database technology. Oracle Integration Server leverages and protects your investment in these skills. It uses tools with which you are already familiar: Oracle Enterprise Manager, Oracle Designer, Oracle JDeveloper, Oracle Report, and Oracle Discoverer. You can access Integration Server functionality through standard languages like Java, C, C++, as well as PL/SQL. You can use existing database administration skills to manage and administer the Oracle Integration Server.

Functions of OIS

Oracle Integration Server (OIS) comprises a number of products used together to enable applications to communicate. OIS is designed to handle complex integration scenarios.

Oracle Integration Server provides these functions:

- Message queuing that enables asynchronous communication between applications
- Tools to transform business objects as they move among multiple applications
- Modeling, generation, and automatic execution of complex business processes

- Data integration through data access gateways
- Data synchronization through replication
- Standard request and reply protocols that enable synchronous communication between applications
- Business intelligence tools to refine, improve, and implement business processes
- System management that enables you to manage and monitor the entire environment from a single console

Oracle Integration Server uses the Oracle8*i* database:

- For data storage and message storage through Advanced Queuing
- For transactional integrity by using the commit model
- As a repository for business processing and transformation rules

Oracle Integration Server includes:

- Oracle Workflow for business process control
- Oracle Message Broker (OMB) for interoperability with other messaging technologies
- Workflow Builder with graphical tools for the definition of automated process flows

These products support messaging and e-business standards such as Java Messaging Services (JMS) and eXtensible Markup Language (XML).

Key Objectives for OIS

Security

Oracle Integration Server takes full responsibility for the secure transit of communications from the point of creation to the point or points of receipt. It ensures the integrity of the communication between the points.

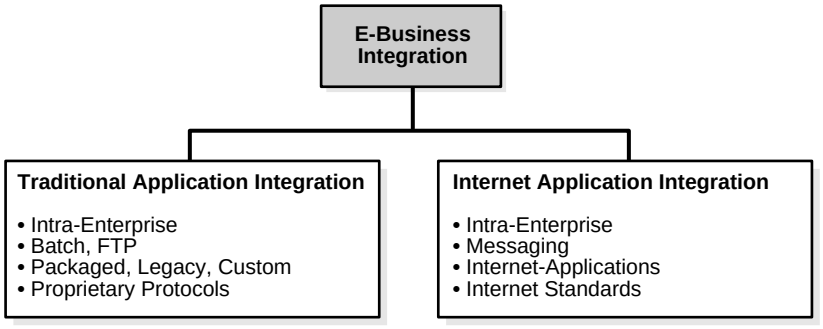
OIS stores a secure copy of the communication on a persistent medium to protect against loss in the event of a system failure and to provide an audit trail of the communication. It ensures that a communication exists only as part of a transaction so that it does not get lost in transit or recorded incorrectly. It maintains an accurate record of the receipt of the communication.

This section contains:

Product Development Life-Cycles

OIS supports the integration of Internet applications, transport layers, protocols, and the languages of legacy applications. It provides versioning capabilities and an interface for performing configuration, development, and management, in both development and production environments.

Figure 3–7 Product Development Life-cycles



Extensibility

You can develop and extend OIS over time without the necessity for extensive modification of the original components. This feature is called extensibility and it makes it easier for you to:

- Add new applications to the solution
- Extend the interfaces of existing applications
- Add new integration components
- Manage integration component upgrades
- Speed exploitation of new channels to market
- Support new standards for protocols and messages
- Support new regulatory controls

Encapsulation

Encapsulation makes the object self-contained by limiting communication with that object to a defined interface. All the products that make up OIS can be encapsulated.

Encapsulation enables you to replace an component without making an impact on any of the components with which it communicates. The new component can be successfully encapsulated if it communicates through the same defined interface or if it provides the equivalent or better integration services.

Encapsulation simplifies your analysis of the impact of replacing a component, it minimizes redesign of the interface of the component, and it minimize the impact on the code and interface of other components.

Component-Based Architectures

Component-based architecture follows logically from the concepts of extensibility and encapsulation. Its objective is to design the architecture as a set of parts, each of which provides a logically grouped set of services and functionality. The removal or replacement of any of the individual parts has only a limited impact on functionality. The basic architecture remains valid and appropriate.

Even if the requirement for a particular component changes significantly, the impact of changes on the other components is minimized. To achieve this, the components must be loosely coupled with one another and must communicate using standard protocol and communication transport.

To loosely couple the software and hardware, the software must be easily portable and able to communicate across different hardware platforms.

New Messaging Technologies

Oracle Corporation has developed messaging technologies that incorporate new features important to the development of an asynchronous messaging infrastructure for enterprise-wide application integration. The most significant of these include:

- Auditing and Tracking
- Business Process Coordination
- Business Intelligence

Auditing and Tracking

Electronic business must incorporate the same rigorous accounting practices that are required of traditional business. It must be possible to capture and reproduce an audit trail of business-event-driven electronic interactions. By incorporating persistent storage of messages in the database and ability to retain messages even after they have been processed, Oracle Advanced Queuing provides a strong audit trail.

Persistent storage also enables you to track business events while they are in progress, so that you can quickly see evaluate the current state of business transactions.

Business Process Coordination

Business process coordination is essential to the effective management of an integration environment. Business process coordination software is a logical extension of the concept governing workflow engines. Workflow engines manage the flow of documents in business processes that include manual steps performed by users.

The principle is the same for business process coordination except that it manages an automated rather than a manual business process. Business process coordination manages the flow of messages between automated processes and also manages the state of the message flows. It lets you manage the state of long-running business transactions independently of the technology used to manage the transaction.

We have extended the Oracle Workflow product to interface with Advanced Queuing in order to provide business process coordination.

Business Intelligence

The Oracle Advanced Queuing technology creates close coupling between the database and the messaging function. This close coupling enables you to use Oracle Business Intelligence software to analyze business events and to identify trends and patterns. You can also monitor the service levels and response times that are provided by process steps called from within the Integration Server.

4

Key Integration Concepts

You have seen how the Oracle Integration Server supports a wide variety of e-business integration architectures and learned about the primary drivers of e-business integration and the specific components of Oracle Integration Server that meet these requirements. Next, we examine in greater detail the use of asynchronous messaging for enterprise integration. This chapter includes:

Asynchronous Message-Based Integration

Asynchronous message-based integration is rapidly emerging as the primary technology required for enterprise integration. To help you understand how to use it most effectively, we next:

- Evaluate an end-to-end enterprise integration scenario and discuss how to use messaging to facilitate integration
- Compare two different conceptual message-based architectures that you can use for integration, point-to-point integration and hub-and-spoke integration, and discuss the relative benefits and trade-offs of the two architectures
- Summarize the key technological components involved in using messaging for enterprise integration, discussing message storage and management, message routing and propagation, and message transformation, as well as message interoperability and the use of standards

In order to understand how messaging can best be used for e-business integration, we now consider in detail the technical architecture associated with connecting a supplier's supply chain application to a business-to-business trading marketplace or exchange. This section includes:

An Example of the Use of Messaging for B2B Integration

For example, a supplier must automate the way it interacts with a B2B exchange so that it can automatically

- Post up-to-date pricing and inventory information to the exchange
- Respond rapidly to auctions and requests for information (RFIs) from the exchange

Let us consider how the integration works from both the supplier standpoint and from the exchange standpoint. From an integration point of view, three primary technological requirements enable such communication. These include:

Communication Between the Supplier and Exchange

To send a message from its supply chain system to the exchange, the supplier must use a solution that receives a message payload and propagates it to the exchange. The message propagation facility must provide a number of services:

- Asynchronous communication: The supplier's systems and the exchange must be connected using asynchronous messaging for three reasons:
  - Supplier and exchange are not tightly coupled in a request-response model because both can continue to process business activities without requiring a response.
  - Communication between supplier and exchange must be resistant to application, system, and network outages.
  - Neither the supplier nor the exchange enables another trading partner to carry out the two-phased commit operation across its systems that is required for a synchronous interaction.
- Guaranteed delivery: Because messages exchanged between supplier and exchange are business-critical, the messaging facility must guarantee exactly once, in-order delivery of messages.
- Message storage and management: In order to resolve potential disputes between the supplier and its trading partners, the messaging facility must store messages sent by and received from the exchange so that you can audit and track them.

Message and Data Transformation

In propagating the message from its supply chain system to the exchange, the supplier must address two message or data transformation issues:

- Message propagation protocol: Since communication between the supplier and the exchange and other trading partners occurs over the Internet, the message must be propagated over the standard HTTP-S protocol. Prefer SSL over HTTP for security reasons.
- Message propagation format: Although the message is propagated over HTTP-S, the message payload must be sent in a format that both the exchange itself and the other trading partners can understand. Most businesses send the message payload in an XML format agreed to by all the trading partners and the exchange.
- Message and data transformation: Because the supplier's supply chain application is likely to store data in a proprietary format, you require a message

transformation facility to accept the data from the supply chain application and translate it into the appropriate XML format.

Business Process Management and Workflow

Before sending the message to the exchange, the supplier may need to get approval for the price list from an appropriate executive. Similarly, in responding to a purchase request from the exchange, the supplier may need to send the message to the company's Financial Application and to update the inventory status in the Supply Chain Application. To manage this multi-step business process, the supplier must maintain a business process management or workflow facility that coordinates messages between the different applications and the exchange with this:

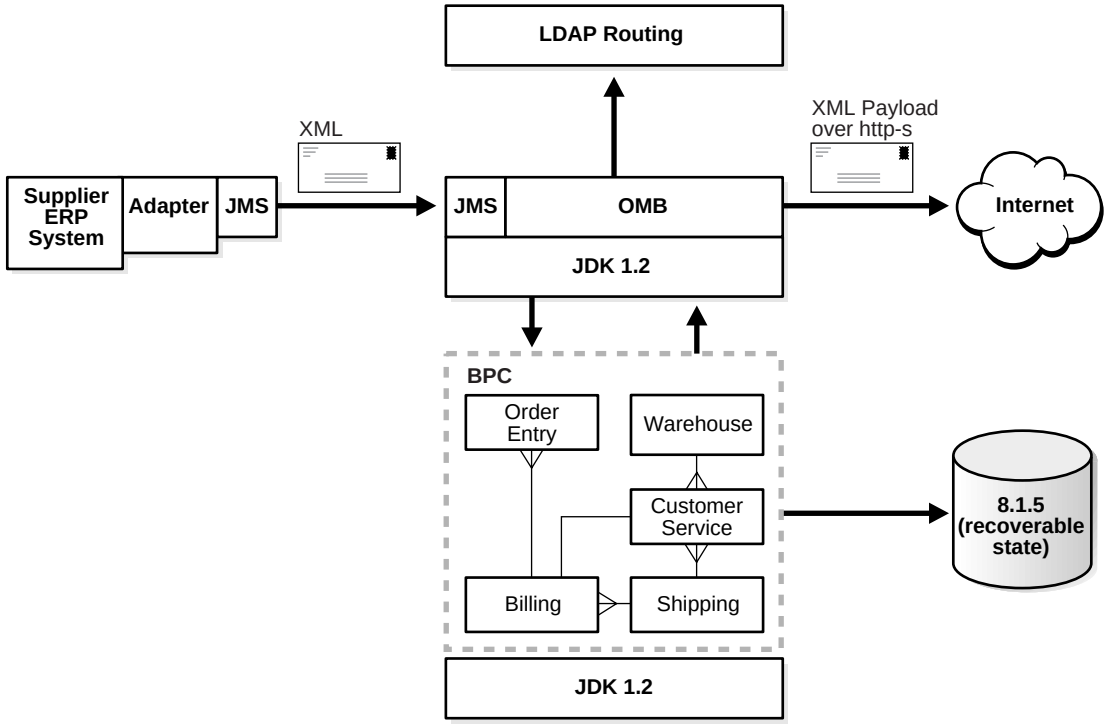
Before sending the message to the exchange, a local workflow process may need to be invoked in order to determine whether there is sufficient inventory associated with each of the items on the new price list. In this case, a business process management facility will be required at the supplier end. The business process management or workflow facility will need to route the new price list from the backoffice application to the company's inventory application to compare inventory status for items on the price list. To manage this multi-step business process, the supplier must maintain a business process management or workflow facility that coordinates messages between the different applications and the exchange. Further, since a single business process may involve coordinating business events between the company's enterprise applications and with an exchanging or an external trading partner, a single business process management or workflow facility should be used for both intra-enterprise and business-to-business interactions.

Now that we understand the primary requirements for this integration scenario, let us examine in greater technical detail the specifics of the integration from the perspective of both the supplier and the exchange.

Exchange Integration Scenario: Supplier Perspective

Various steps and integration components link the supplier's supply chain application with the business-to-business exchange. In this scenario this is an Oracle exchange. The steps and components involved in establishing such connectivity are illustrated in Figure 1.

Figure 4–1 Integrating with a B2B Exchange: Supplier Perspective



These steps and components include:

- Adapter technology: For the application to connect to the exchange, the supply chain application must first use an adapter to post a message to the exchange integration facility. The adapter waits for new inventory status information to be posted on a specific message queue. The adapter receives the message in the specific data format of the application, so the adapter then converts the data into XML.
- Message propagation infrastructure: The application adapter first enqueues the message into a message propagation infrastructure through a standard messaging interface. The messaging infrastructure can then provide three additional services:
  - It can route the message to the exchange either statically or based on its subject or content.

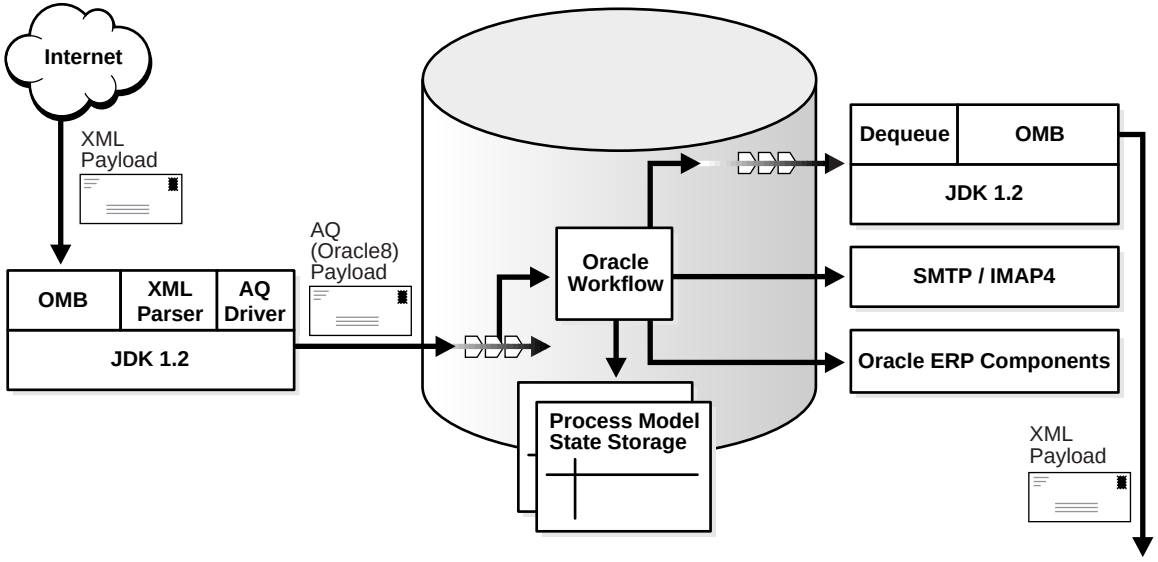
- It can consult an LDAP directory service to determine where to send the message.
- It can optionally store the message in a persistent store so that the message can be audited or tracked.

- Local Workflow Processing: If local processing is necessary, the local business process management facility can dequeue the message from the message propagation infrastructure, process it locally, and, when this is completed, place the message back in the message propagation infrastructure in order to send the message to the exchange.

Exchange Integration Scenario: Exchange Perspective

The integration architecture of the Oracle exchange resembles the architecture described in the previous section.

Figure 4–2 Integrating with a B2B Exchange: Exchange Perspective



An XML payload is received over the wire on the HTTP-S protocol and goes through the following stages:

- Message receipt and propagation: The message management facility receives the message as an XML payload over HTTP-S.
- Data transformation and parsing services: When the facility receives the message, it frequently places the message in a persistent store so that it can be audited and tracked for conflict resolution. The message header and payload may often need to be parsed using an XML parser for two purposes:
  - To determine where to route information based on either the subject or topic of the message or the content of the payload. For instance, the message may need to be sent through a specific approval cycle if the supplier is posting inventory levels that are below the threshold for the exchange.
  - To parse the message into a structured format to enable it to be placed in a business process management or workflow facility for processing
- Business Process Management: The workflow system dequeues the message from the message propagation infrastructure and carries out an entire workflow process to update the various applications that form part of the exchange. The workflow facility can also:
  - For example, send an e-mail message to notify a small supplier by directly calling its own SMTP / IMAP4 interface
  - Send an XML message outbound: If the workflow system needs to send an XML message outbound to another supplier over HTTP-S, it can serialize the message into the XML payload format appropriate to the target, determine how to route the message by consulting an LDAP directory, and then enqueue the message into the message propagation infrastructure.

Messaging Technology and Architecture

We have now explained the concepts of the integration of applications and business processes using an asynchronous messaging infrastructure. In this section, we discuss in greater detail the specific technical aspects of the use of asynchronous messaging for e-business integration: This section contains:

Messaging Technology - An Overview

Highly interdependent processes, often described as closely coupled or tightly coupled processes, are more difficult to maintain, more failure prone, and more difficult to recover than processes that operate independently of each other. For this reason, developers are increasingly designing modular applications with as much independence between processes as possible.

To limit the dependency between two processes, developers use

- Messaging: formalized communications between processes
- Decoupling: a technique for minimizing the number and complexity of interactions between processes
- Middleware: an intermediary between two processes

Middleware comprises a storage medium for messages called a queue, a generic message-passing process called a message broker, or both a queue and a message broker. Communicating through this intermediary enables each process to operate independently of the availability, performance, physical location, or implementation style of another process.

Aspects of the messaging system that is required for e-business integration including:

- Synchronous and asynchronous processing
- Session-based and sessionless communication
- Stateless and stateful communication
- Two-way and one-way communication

Synchronous and Asynchronous Communication

Two processes can communicate through a message either synchronously or asynchronously.

In synchronous processing, the sending process passes control to the receiving process and suspends processing until it receives a reply from the receiving process to confirm that it has received and processed the message.

In asynchronous processing, the middleware queue separates the sending process from the receiving process. The sending process continues processing as soon as the middleware receives the message. Note that a multithreaded program, one of whose threads waits for a message to be processed, is operating synchronously,

even though the program continues to process other threads while that thread awaits a reply.

Session-Based and Sessionless Communication

A message can be created or processed either in encapsulation or in the context of a session. A session is a period of time during which a user or a process establishes a recognized and predefined connection to another process. Messages are either durable or non-durable. Durable messages can be sent independently of a currently running session. Non-durable messages cannot, and they require session-based communication.

Session-based communication between processes takes place within the context of a session. The rules that govern the session affect the way the message is created and processed. Session rules also affect the way the middleware manages the message: how it is routed and stored and retained, whether a reply is sent to the sender, and so on.

Sessionless communication takes place outside of the context of the currently running session. The durable message is processed through a previously established connection to the database, the middleware, or the application. A sessionless message can be managed and processed without the requirement for additional information provided by the currently running session.

Stateless ("Without State") and Stateful ("With State") Communication

The transactional state of a communication is either "with state" or "without state." The transactional state is the state of the communication. It has nothing to do with stages in the life cycle of the message itself: created, ready for processing, processed, and so on.

"With state" communication occurs when messages are created, managed, and processed within the context of a single transaction. The position of the message within the transaction affects the way the message is processed. The process that creates the message, the middleware, and the process that acts on the message all independently maintain the "with state" condition so that the message is processed in the context of the transaction.

"Without state" communication occurs when messages are processed without reference to their position within a single transaction or when the message crosses transactional boundaries.

Two-Way and One-Way Communication

Synchronous communication is two-way communication because the sender of the message waits until it receives a reply indicating that the message has been successfully processed. In general, communication that is "with state" or session-based is two-way communication. A message sent in the context of a session or transaction is almost always associated with a reply sent by the receiving process.

The data contained in a message is its payload. A message sent from one discrete process or application to another independently of a session or transaction contains sufficient information to enable processing. Because further communication with the sender is not required to process the message, it is said to have an autonomous payload.

An asynchronous, "without state," sessionless message with an autonomous payload is a one-way communication. The management and processing of the message have no impact on the sender process, which continues immediately after the message is sent. One-way communication adds another degree of separation between the sender and the receiver of the message.

Message-Based Integration Architectures

Now that you understand the broad technical aspects of messaging, we examine the two primary architectural alternatives in the use of messaging for e-business integration. Depending on your data processing and messaging requirements, you can use one of two possible architectures to organize your solution.

Two alternative message-based integration architectures and their relative benefits and trade-offs

- Point-to-point integration
- Hub-and-spoke integration

Point-to-Point Integration

A point-to-point integration architecture integrates each application directly with each of the other applications with which it needs to communicate. The applications communicate with one another directly on a one-to-one basis. This architecture has historically been used to provide file-based integration between applications.

It requires relatively little architectural infrastructure. In a phased implementation, it speeds implementation of the early phases. However, it has recognized limitations, particularly in terms of ongoing maintenance costs (which can be

exponentially higher than those of hub-and-spoke architecture), and in lack of flexibility and manageability.

Hub-and-Spoke Integration

In contrast to point-to-point integration, most modern integration scenarios are best served by hub-and-spoke architecture. A hub-and-spoke architecture provides a central intermediary (called a hub) through which applications communicate with each other. In this architecture, each application (which represents a spoke) communicates with the intermediary (the hub), which in turn manages communication with the other applications. Applications do not communicate directly with each other.

In the context of a hub-and-spoke integration architecture, the hub provides a central point for a variety of services:

- Message propagation and communication: The hub provides a central facility that manages the routing and propagation of messages among the different applications. If the message, based on its content or topic, must be routed to more than one application, a hub-and-spoke architecture provides greater flexibility than a point-to-point architecture.
- Message management, tracking, and auditing: The hub stores, audits, and tracks messages flowing between the different applications, thus providing a central facility in which all interactions between applications are managed. In a point-to-point integration architecture, information required for auditing is stored within each application, thus complicating analysis of information flows.
- Message and data transformation: The hub provides message and data transformation to convert data from the format of the sending application to the format of the receiving application. In a hub-and-spoke architecture, each application simply transforms the data into a common view from which the data can be converted into the format of the receiving application. When the sending application sends a message to a new application, the hub needs only to apply a new transformation from the common view to the format of the new application. This eliminates the requirement for a new point-to-point map.
- Business process management: The hub provides business process management or workflow services to manage multistep interactions between applications. For instance, one application sends a message to another application, then receives a response from that application, and then sends the information to a third application. By centralizing business process management in the hub, hub-and-spoke architecture simplifies management of workflow among different applications.

Note that because integration architecture is defined at many different levels, the hub-and-spoke structure may not exist at all levels. For instance, the hub and the spoke might all be deployed on the same hardware or potentially on different systems. In general, the hub provides a focus for application interfaces and integration requirements because each spoke communicates only with the hub. This isolates the resource requirement for integration services.

Some people think that the hub-and-spoke approach simplifies the interface to an application by reducing the number of integration points into an application. Actually, the API is as complex as in a point-to-point architecture, but the hub-and-spoke approach focuses the integration points for an application into one place, thus making applications easier to design, develop, and manage.

Benefits and Trade-offs

Consider two primary differences between hub-and-spoke and point-to-point architectures:

- Complexity of Applications: Point-to-point integration architecture is best suited for a relatively simple integration scenario in which a small number of applications are integrated; the applications are rarely replaced, upgraded, or modified significantly; the applications use much the same technology. A hub-and-spoke architecture, though it introduces the complexity of an integration hub, is suited for integrating many different applications and systems. It is more extensible than a point-to-point architecture because new applications need only to be integrated with the hub and not with all of the other applications.
- Complexity of integration requirements: Point-to-point integration architecture is best suited for a scenario that has relatively simple integration requirements:
  - Data is transformed simply between applications.
  - Messages are routed to only a few destinations using simple routing schemes.
  - You do not require a central business process coordination facility.
  - You do not require a common view of all business events.

  In a hub-and-spoke architecture, the hub provides these services to all the applications and thus simplifies how the integration occurs.
- Business-critical nature of integration: A hub-and-spoke architecture is also well suited for integration scenarios that are business-critical:

- Information communicated between applications and business processes is highly sensitive.
- Cost of an integration outage or lost communication is significant.
- Fundamental need exists to retain, audit, and track business event information.

- Manageability of integration environment: Although hub-and-spoke architecture introduces the additional complexity of a coordinating hub, it offers a central point where integration-specific functionality is focused to provide centralized management, maintenance, backup and recovery, tracking, routing, transformation, and so on. Hub-and-spoke architecture provides lightweight spokes where the integration and interface requirements of each application are defined, designed, delivered, and maintained. This simplifies maintenance, impact analysis, upgrade cycles, and the retirement of legacy systems.

Messaging Technology

Now that you understand how you can use asynchronous messaging for e-business integration, we examine some of the key aspects of messaging technology specifically discussing in this section:

- Message Propagation and Routing
- Message Notification Models
- Message and Data Transformation Requirements
- Message and Data Transformation Issues

Message Storage and Management

A fundamental requirement for an asynchronous messaging infrastructure that furnishes a platform for e-business integration is that the infrastructure provide persistent storage and management of messages. As described earlier, the ability to store and manage messages is critical for four reasons:

- Inability to process messages as they arrive: Applications may have to deal with many unprocessed messages arriving simultaneously from clients when they may not have the resources to process all the requests immediately. A messaging system must store the message in a persistent queue and deliver it later when the recipient can process it.

- Message scheduling: Messaging systems also require message persistence so that they can deal with priorities:
  - Messages arriving later may be of higher priority than messages arriving earlier.
  - Messages arriving earlier may have to wait for messages arriving later before you can act on them.

  Such message priorities may also change over time: for instance, messages in a particular queue may become more important than messages in other queues during certain time windows. Message persistence enables messages in high priority queues to be processed first. Low priority messages can be stored and processed later without interfering with high priority messages.
- Message auditing: Message persistence is critical because the control component of the message can be as important as the payload information itself. For instance, the time that a message is received and dispatched can be a critical part of a message. In an e-commerce environment, message persistence stores information on orders from various customers that you can query to understand periods of peak demand or to determine the status of an order. The message, therefore, may remain important even after it has been executed. A persistent messaging store is critical so that such information can be warehoused and queried or audited.
- Communication Failures: The communication links between messaging clients may not be available all the time or may be reserved for some other purpose. If the system cannot process messages immediately, the messaging system must store the message persistently and deliver it when processing resources are available. Such guaranteed message delivery of each message to each recipient exactly once is critical to integration of enterprise applications.

Note that the need for message persistence is enhanced in a business-to-business commerce scenario in which the likelihood of communication failures and the need for auditing and tracking are far more significant than within a single enterprise. Although most messages in an integration scenario must be stored persistently, in some cases volatile messages are sufficient. For instance, if an e-mail is sent to an application to notify it that a message has been sent or received, the e-mail message need not be stored for auditing.

Most messaging systems do not provide automatic message persistence. They simply store the messages in `RAW` format in a file system. As a result, the message headers and payloads are not queryable. If the message does need to be stored persistently, the messaging system requires a two-phase commit operation between itself and a database system.

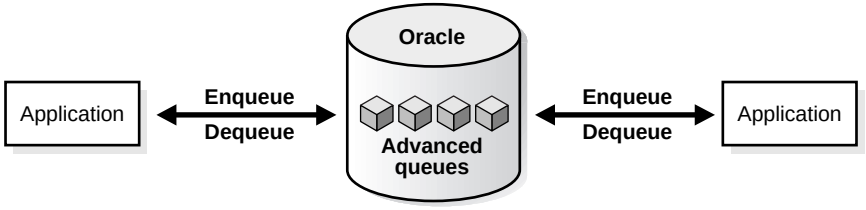
In contrast with such complexity, Oracle Corporation integrates a messaging infrastructure directly with its enterprise database: associating queues with database tables and ensuring that messages placed in these queues automatically persist in the database. You can then archive the messages and execute queries against both message headers and payloads using standard SQL.

Message Propagation and Routing

Because messages must be propagated from one application or business process to another, the messaging infrastructure offers a mechanism to route the message from the sender to the receiver. Message propagation and routing can be performed in one of two ways:

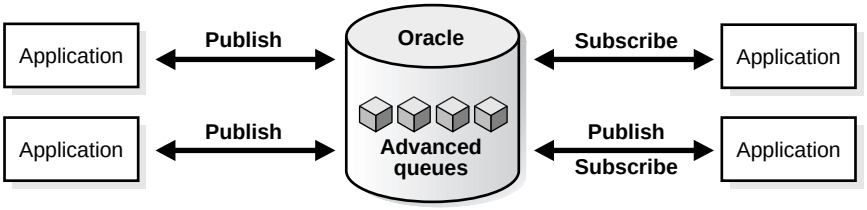
- In point-to-point routing, the sending application knows the destination to which it is sending the message (the receiving application). Point-to-point message routing provides relatively tight coupling between the sending application or business process and the receiving application or process.

Figure 4–3 Point to Point Model



- In publish-subscribe routing, the sending application does not know the destination to which it is sending the message, but instead assigns the responsibility for managing who receives the message to the middleware. The middleware defines a list of topics on which the message creator publishes messages.

Figure 4–4 Publish-Subscribe Model



A process identifies itself to the middleware as a subscriber and defines the topics on which it receives messages. The middleware sends the subscriber any future messages on that topic until the subscriber informs the middleware that it no longer wants messages on that topic or ceases to subscribe. This design technique provides an additional level of separation between the processes. The sender publishes message to middleware without regard to the receiver of the message, the delivery location, or the processing of the message. The middleware is responsible for:

- Registering subscribers and enforcing conditional subscription rules
- Determining recipients for individual messages
- Tracking both message creation by publishers and collection of messages by subscribers

The subscriber registers an interest in receiving messages on particular topics and subscribes without regard to the sender or publisher of the message and where the message is created.

Point-to-point and publish-subscribe based message routing are the two broad categories of message routing. In reality, each of these schemes can be more complex. For instance, in both a point-to-point and a publish-subscribe environment, messages can be routed in three ways: subject-based, topic-based, and rule-based. We now consider each of these and discuss, with examples, how each applies to publish-subscribe routing.

- Subject or topic based: This is the simplest form of publish-subscribe. A process publishes messages related to an appropriately named topic or subject. For example, the subject might be PLACE AN ORDER. Subscribers that have registered with the middleware stating an interest in receiving messages on that topic receive all messages on that topic until they unsubscribe from that topic: “Send me all messages on the topic PLACE AN ORDER.”

- Content Based: This is an enhanced version of topic-based publish-subscribe. The subscriber asks to be sent only a subset of the messages received on a particular topic, depending on the values contained within the message. For instance, “Send me all messages on the topic PLACE AN ORDER in which the message contains UK in the country code field of the message data.”
- Rule Based: Rule-based publish-subscribe offers a further enhancement of the content-based mechanism. The subscriber specifies rules that govern conditions that might or might not relate to the values of the message data. For instance, “Send me all messages on the topic PLACE AN ORDER in which the message contains UK in the country code field of the message and the current time is between 9 am and 5 pm and I have the spare capacity to fulfill the order.”

Message Notification Models

Businesses commonly use two messaging models to facilitate internal communication. A complete communication solution typically requires use of both models:

Event Notification

The event notification model addresses the need to communicate business events throughout the business. A business event is a logical occurrence in a business scenario. It may be simple, as when a customer places an order, or it may be more complex, as when a customer exceeds his credit limit while placing an order. A system event, on the other hand, is a physical point in the execution of a program at which an identifiable computing task takes place, for instance, writing to a file. A single business event is usually associated with a number of system events.

The event notification model defines the communication points between applications and the content of the communication. It uses asynchronous messaging and publish-subscribe routing to deliver messages that represent instances of a business event. Each business event is defined to the middleware as a separate publish-subscribe topic. The application or applications that process the business event also publish a message each time an instance of a business event occurs. An application that must know about the business event subscribes to the topic relevant to that event. The middleware manages delivery of the message to the subscriber and collection of the message by the subscriber.

Typically, each message announcing a business event is autonomous and requires no communication between the applications other than delivery of the message. Usually the subscriber need not respond to the publisher. Once it has acted on the

information received in the message, the subscriber can raise an instance of another business event and can publish messages about this business event as another topic.

Service Requests

The second of the two most commonly used messaging models, the service request model, offers an alternative modeling style to event notification. It is based on the premise that one application can avail itself of the services of another application. For instance, application A requests a service from application B by creating a service request in the form of a message.

The service request model has many more variations than the event notification model. The variant selected depends on the requirements of the individual solution. For instance, the service request model is implemented using asynchronous messaging when no confirmation of service delivery is required. This variant may use publish-subscribe routing. If a confirmation is required, whether it is positive or negative, the service request model is implemented either by using asynchronous messaging or by synchronously using a call-and-reply messaging technology.

In a service request model, the application requiring a service that it cannot itself provide creates a service request message for a named application that provides the service. The middleware routes the message to the named application. The application providing the service has a defined means of invoking the services of queues and of a callable API. Three of the possible variations of the service request model are shown here:

- Non-Publish-Subscribe Service Request Model: In this configuration, the application requiring a service sends a request for service to a named application through the middleware.
- Publish-Subscribe Service Request Model: In this configuration, the application that provides the service registers with the middleware as a service provider. The application requiring the service publishes its request to the middleware, which routes the message to the registered service provider.
- Publish-Subscribe Service Request Model with Positive Confirmation: In this configuration, the application providing the service successfully delivers the service and then returns a message confirming service delivery. The middleware routes the confirmation to the application that originated the request.

Message and Data Transformation Requirements

If you use asynchronous messages to integrate applications or business processes, a fundamental requirement is to transform the message payload sent by the originating application to a different format that can be accepted by the receiving application. To understand the transformation process, consider an Oracle Customer Relationship Management (CRM) application that is attempting to send a purchase order received in its Web store module to an SAP financial application. There are two specific transformation requirements:

Datatype Transformation

The Oracle CRM application stores all its data in SQL format and the SAP application stores and accesses its data in iDOCS, which is a flat file format. SQL datatypes must be converted into the iDOCS datatype at the byte representation level; this is called datatype or data level transformation. When considering data transformation itself, a number of different kinds of transformations can apply:

- Value: Change the data value from 123 to ABC.
- Name: Change the data name from `CUSTNUM` to `customer_id.`
- Type: Change the datatype number to alphabetic.
- Payload Definition Mapping: Concatenate `BranchId`, `CustomerNo`, `AccountType` and `CheckDigit` to give `AccountNumber.`
- Payload Format: Change the actual structure of the message payload, for instance HTML to XML, XML to fixed file format, XML to comma separated values (CSV), name-value pairs to XML, and so on. Do not confuse structural transformation with programmatic transformation.
- Programmatic: Change the message payload format between that supported by a programming language and that supported by a messaging technology, for instance C Struct to XML string, Java Class to stringified object, COBOL picture to ADT (also known as Pro*COBOL).
- Messaging API: Some standard messaging services are implemented differently in different technologies and you may need to transform messages between the formats of the underlying structures, for instance, OMB JMS to OJMS or OJMS to MQSeries JMS.
- Header: Some mapping may also be required in the message properties recorded in the header as you move from one messaging technology to another, for instance, AQ to MQSeries or AQ to TIBrv.

Semantic Transformation

The Oracle CRM application stores a customer name in four fields: Mr./Mrs./Miss, First Name, Last Name, Middle Initial. In contrast, the SAP application stores the same customer name in two fields: the first field specifying Mr./Mrs./Miss and the second field concatenating the First Name and Last Name with a blank in between. In this case, you must concatenate the fields in the customer name of the Oracle application before sending it to the SAP application. Such a transformation is called semantic transformation.

Message and Data Transformation Issues

In addition to understanding these two different kinds of transformation requirements, you must consider four additional issues when thinking about how messages need to be transformed for a particular integration scenario.

Transformation Location

In a point-to-point integration architecture, the sending application transforms the message into the format of the receiving application and then places the transformed message on the message propagation infrastructure, which then passes it to the receiving application.

If transforming data from a sending application to a receiving application in a hub-and-spoke architecture, you must decide whether to transform the message at the spoke or at the hub. For instance, the sending application might behave in the same way as in a point-to-point environment and simply use the hub as a message propagation and routing infrastructure. In contrast, if a single application sends information to multiple receiving applications, it can simply send an untransformed payload to the hub, which then applies the transformation appropriate to the target application receiving the message.

Transformation Mechanism

Additionally, in a hub-and-spoke architecture, the hub can directly transform the payload of each sending application into the format of the receiving application. Alternatively, each application can transform its data from an application-specific view into a common view and then send it to the hub, which converts it from the common view to the target application-specific view. The primary benefit of using a common view for transformation is that when a new application is added or an existing application is upgraded, the integrator needs only to change the mapping from the common view to the application specific view. It need not change the mapping from every single application that communicates with this application. As

a result, using a common view for transformation is more extensible than direct transformation.

Using a common view for data transformation affects the way the data itself gets converted. In typical scenarios, the common view of data is increasingly represented using XML (or in certain cases Java classes). An adapter associated with the sending application converts the data from the application-specific format into the common view representation, places it in an XML document, and then sends it to the messaging infrastructure of the hub. The hub then applies an XSL-T transformation to the XML document based on the necessary semantic information, converts it to an outgoing XML representation, and places it on the message propagation infrastructure. An adapter of the receiving application then receives the XML payload, parses the XML payload, and converts it into the data format of the receiving application.

Note two important benefits of transforming data into XML:

- XML message payloads can be sent over HTTP-S and, as a result, a common infrastructure can integrate applications within a company and between companies using the Internet for message propagation.
- XML is a completely extensible data description language and, as a result, can represent any kind of data; it is ideally suited to describe different kinds of data for transformation purposes.

There are, however, two potential drawbacks to using XML:

- Since it is string-oriented or text-based, the need to parse the data for transformation can introduce overhead.
- Most XML parsers on the market today cannot handle message payloads beyond 10 MB in size through their DOM APIs; the DOM parse trees that are created when the document is parsed are too large. Although you can use the SAX API, it too is frequently inefficient.

Transformation Event Frequency

Finally, consider the frequency of transformation and the business events that necessitate a transformation. There are three kinds of such transformation events:

- Specific: The transformation is specific to a particular business event coming from or going to a particular application.
- Defined: The transformation is defined as a set of maps between standard message definitions, for instance XML to SWIFT, and needs always to be

applied whenever a message of one type is sent to an application or business process that communicates in the other format.

- Generic: The change applies to all messages in a particular adapter, for instance, MQSeries messages to AQ messages.

See Also: Oracle8*i* Applications InterConnect 3.1.3

Message System Interoperability

When considering an e-business integration scenario, the final aspect of messaging that you need to understand is interoperability between different message queuing systems and different types of integration middleware:

- Message System Interoperability: Firstly, how do you get two messaging systems to communicate with each other at the programming interface level and send messages to each other? A related question is: How can an application communicate with multiple messaging systems through a single standard programming interface?
- Message Payload Interoperability: Secondly, how do you get different messaging systems to communicate through a common message payload format or to interpret message payloads of the other system?

In order to ensure interoperability between messaging systems at both the payload and the system level, messaging middleware vendors have adopted two standards:

- Java Messaging Service (JMS) API to ensure message system interoperability
- XML (and a variety of derivatives such as OAG, XML, and OBI) for message payload interoperability

In the next section, we provide a brief overview of these two standards.

Java Messaging Service (JMS)

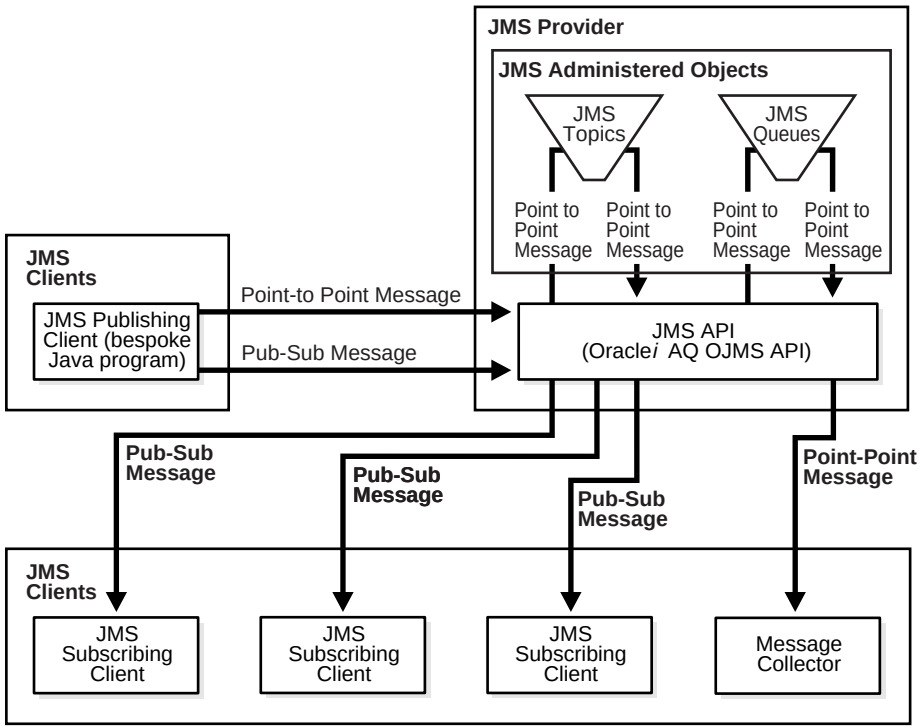
Java Messaging Services (JMS) is the Java industry standard for asynchronous messaging, developed jointly by Sun and other enterprise messaging vendors including Oracle. It complements the synchronous communication model specified by Java Remote Method Invocation (RMI).

The most commonly held misconception about JMS is that it is an application or component that provides messaging services. JMS itself is simply a definition of a standard. Vendors of messaging technologies (such as Oracle) develop and sell software (such as Advanced Queuing) to provide messaging services that conform

to the JMS standard and that can be accessed through a Java API that conforms to the JMS standard.

These JMS compliant services are made available through a published interface that complies with the JMS standard for the programmatic interface (API). The API provides a framework that enables you to develop portable, message-based applications in the Java programming language.

Figure 4–5 Java Messaging Services



The JMS standard defines four important components:

- Clients: Those programs or processes create and consume messages (for instance, a Java Application Program).
- Providers: Those programs or processes provide the JMS service (for instance, AQ).

- Administered objects: Objects are used by the providers to manage and route the messages (for instance, JMS Topics and JMS Queues).
- Messages: These are the messages themselves.

The JMS standard defines two types of message routing:

- JMS Topics that provide a definition for publish-subscribe routing
- JMS Queues that provide a definition for point-to-point routing of a message from one point to another

The JMS standard defines various characteristics about messages including some properties relating to the message itself (stored in the message header) and a number of different styles of payload for the message content. It also defines how content-based publish-subscribe is implemented using filtering conditions on the message header and its properties. The five styles are:

- `TextMessage`
- `BytesMessage`
- `MapMessage`
- `StreamMessage`
- `ObjectMessage`

The Oracle Implementation of JMS

Each vendor who offers a JMS implementation develops a set of services that conform to the standard. The vendors are free to extend their implementation to offer services beyond those defined in the JMS standard.

Oracle8*i* Advanced Queuing offers just such an extended JMS interface, which is called Oracle Java Messaging Services (OJMS). Oracle provides JMS Queues using Advanced Queuing single-consumer queue functionality. Each JMS queue is mapped one-to-one to a single-consumer queue. JMS topics are supported using Advanced Queuing multi-consumer queues. Each JMS topic maps to a multi-consumer queue. Each JMS message type maps to an Oracle Object Type. The JMS connection encapsulates a JDBC connection. This enables the client to combine JMS messaging and other JDBC operations in the same database transaction. JMS Clients executing outside the database JVM can access the JMS API using the “thin” or “OCI” JDBC driver. JMS Clients executing inside the database JVM (JServer) can access the JMS API using the Oracle Server driver. The standard JMS interfaces are available in the `javax.jms` package.

The Oracle extensions to the JMS implementation can be found in the `oracle.jms` package.

See Also: *Oracle8i Application Developer's Guide - Advanced Queuing*

XML

XML (eXtensible Markup Language) is a markup language for documents containing structured information. It provides a mechanism for applying structure to an information set, such as a message or a document, by enabling the data in the set to be labeled, assigned attributes, qualities, or default values and enabling you to verify or validate the information set. It is becoming the standard message format for e-business and is the preferred standard for most major IT vendors.

The main advantage of XML lies in its flexibility and its simplicity. Each message contains within it the information required for the structure of the message to be understood. This is often called a self-describing message. Consequently, any program that knows the rules of XML can understand any message formatted as an XML message without having to store and overlay metadata about the structure of each message.

XML was derived as a subset of the SGML (Standard Generalized Markup Language), which has been the standard, vendor-independent way to maintain repositories of structured documentation for more than a decade. SGML is very powerful but complex to use for e-commerce, so XML was developed as a simplified version of SGML.

The content of a message or document can be thought of as a logical set of information, and that information often needs to be structured so that pieces of the information can be identified, accessed, and manipulated as subsets of the message or document. These subsets are known as elements. Each element can be made up of any number of other elements, and elements can repeat in a group.

XML has syntactic semantics that enable you to define labels known as tags that define where an element starts and where it ends. It is possible to define attributes that describe the characteristics of the element as a whole. These attributes apply to all the sub-elements and to the data contained within the element.

XML also enables you to define a validation template called a Document Type Definition or DTD, which can be used to provide two levels of validation:

- Well formed: To check that the message or document has been formed correctly. If it passes, the message or document is described as well-formed.

- Valid: To check that the message or document is valid against a set of rules defined in the DTD. The validation is performed by a service known as an XML parser, and most vendors including Oracle have software that provides such a service.

  See Also: *Oracle8i XML Reference*

Part II

Products

Part II discusses each of the product components of OIS in the context of integration alone. For complete information on the individual products themselves, you should read product-specific documentation. A link to appropriate product-specific documentation is provided where applicable.

5

Synchronous Application Integration

The Oracle8*i* Java Virtual Machine (JVM) provides a highly scalable and available environment in which to run and execute server-oriented Java applications.

This chapter contains:

Facilities Provided by the Oracle8*i* Java VM

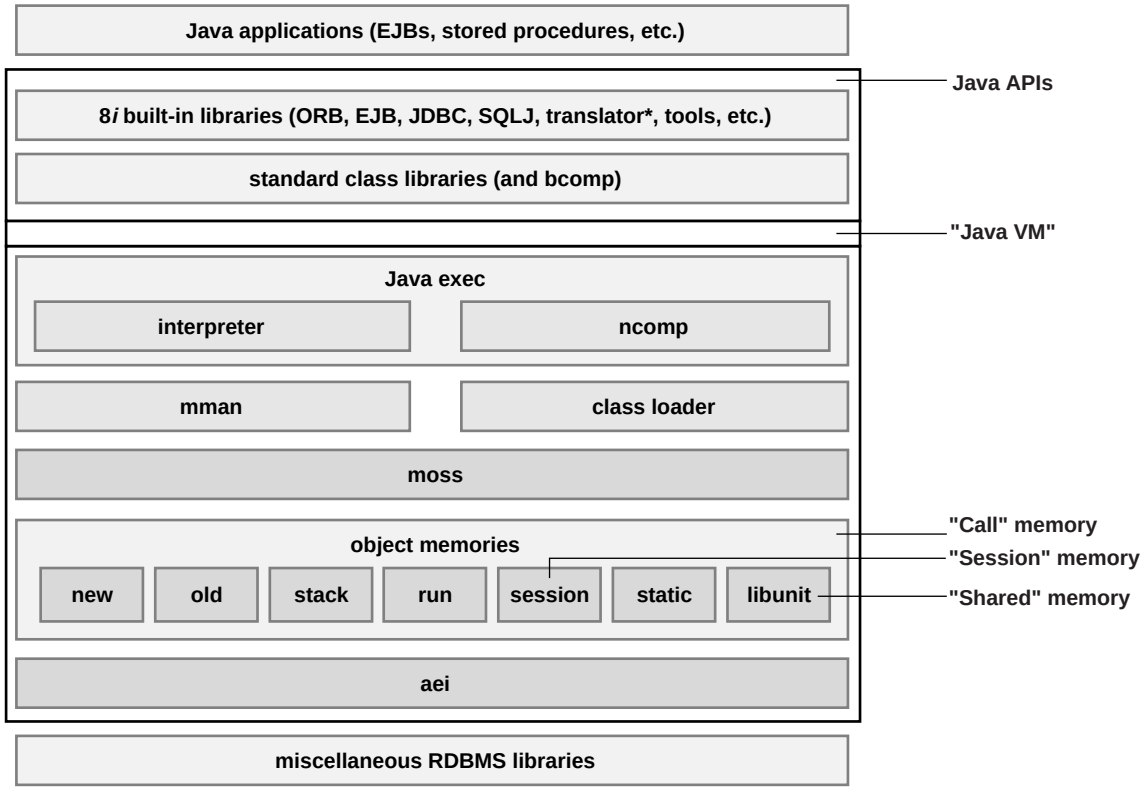
The Oracle8*i* JVM has several key architectural components. This section contains:

- Core Facilities Provided by Java VM
- Core Runtime Facilities Provided by Java VM
- Integration Between Java VM and the Database

The Java VM is a 100% complete Java execution environment, complying with the Java Language Specification and the Java VM specified in the JavaSoft JDK 1.1.6 standard. It is tightly integrated with the database and has a number of components: a bytecode compiler, a garbage collector, an integrated Java class loader, and a Java-through-C compiler, all of which have been designed for optimal performance and scalability within the database environment.

It runs in the same process space and address space as the database kernel itself, sharing a number of the memory heaps and having direct access to the buffer cache of the database to avoid copying memory for optimal performance. The Java VM provides a runtime environment for Java objects. It includes the representation of all Java data structures and supports Java method dispatch. It supports all of the standard Java bytecode operations, including the Java non-recoverable exception model and Java Language-level threads. The VM's dispatch and execution model are also designed to support hybrid interpreted or compiled systems, enabling users to access Java-through-C translation for better performance.

Figure 5–1 Oracle Java VM Architecture



Core Facilities Provided by Java VM

The Java VM provides the following core functions:

- **Object Memory Management:** The Oracle8*i* Java VM allocates and frees memory in standard chunks called **object memories**, which can be specialized in various ways, depending on how they are to be used. Object memories provide users with four important benefits:
  - They are location transparent and can be efficiently relocated across thread, process, host, or temporal boundaries without the inherent costs of object serialization and deserialization, while maintaining the uniform reference semantics of Java. This means the Oracle VM provides sophisticated load

balancing and failover while isolating users from the need to use low-level, error-prone Java threads for scalability.

- Object memories can be specialized for optimizations in the way they are garbage collected: whether they are preinitialized in state or transactional in nature, whether they require rollback and atomic commit semantics.
- Persistent and shared object memories can be archived efficiently to disk.
- Object memories isolate the rest of the Java VM and Java applications from how they are themselves implemented.

- **Memory Manager and Garbage Collector:** The Java VM provides a garbage collector that has been optimized to provide excellent performance and scalability for the Oracle database environment. The garbage collector is responsible for automatically managing the Java memory heaps of the VM and for efficiently allocating and collecting object memories.
- **Java Library Manager:** The Library Manager is the component of the Java VM that provides facilities to load, store, and manage Java programs in the database. Users can interchange (both import and export) Java in three forms: source, binaries and resources, and archives. Java programs are stored as database library units (the equivalent of OS files).
- **Java Class Loader:** The class loader takes requests at runtime from the Java VM and locates, loads, and initializes local DBMS-stored Java classes (in Java binary, or native compiled form).

Core Runtime Facilities Provided by Java VM

The Java VM provides the following core runtime functionality:

- **Bytecode Compiler:** The Java VM embeds the standard Java bytecode compiler from JavaSoft, which translates standard Java source programs into standard `Java.class` binary representation.
- **Interpretation Runtime:** The Oracle8*i* Java VM provides a bytecode interpreter and associated Java runtime that executes standard Java binaries. The interpreter is a 100% implementation of standard Java (currently release 1.1.6) including advanced features such as Java threads and exceptions. Additionally, the runtime provides support for call-in and call-out from the host environment, native methods, and natively compiled Java modules.
- **Standard Libraries:** The Oracle8*i* Java VM supports all of the standard JDK 1.1.6 libraries except for the GUI-related component, Abstract Windowing Toolkit (AWT).

- **Native Compilation:** The Oracle8*i* Java VM provides a native code compiler to increase the execution performance of Java programs to near compiled C-language levels. The native compiler translates standard Java binaries to C executables, which are stored persistently in the database as library units and can be loaded as DLLs or `.sos` by the Java VM.
- **Java Native Interface:** The Java VM supports the Java Native Interface, which provides access to native methods.

.

> **Note:** Oracle Corporation does not externalize JNI to general purpose application developers, in order to prevent you from linking unsafe C libraries into the address space of the database.

However, the Java VM does provide facilities to call out from the database to C and C++ programs.

Integration Between Java VM and the Database

The Java VM interfaces with the rest of the Oracle8*i* database through four different mechanisms:

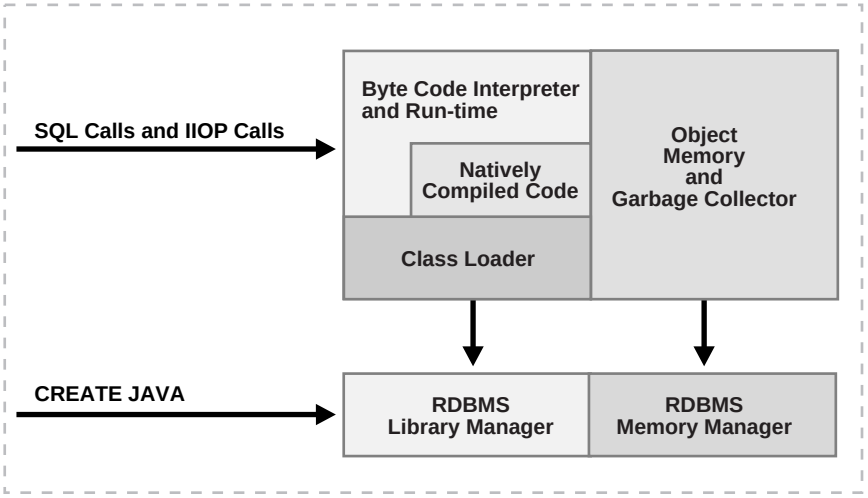
- **JDBC Driver and Embedded SQL in Java Translator:** Persistent data is stored in the Oracle database either as relational tables or as object relational tables defined using SQL. Java programs in the database access SQL and PL/SQL in the database through SQL or JDBC. For static SQL queries, you can write SQLJ applications that are simpler than JDBC. To write dynamic SQL, call JDBC.
- **JDBC Driver:** The VM incorporates a specialized implementation of the Oracle JDBC Driver. It corresponds to the JDBC 1.2.2 specification and to virtually all of the JDBC 2.0 specification, including support for object-relational types, collections, and LOBs. It ports the same programming APIs as the Oracle client-side JDBC/Oracle Call Interface, and Thin JDBC Drivers. This enables application developers to move Java applications onto the database without modifying application code. The only difference between the client-side JDBC drivers and the server-side driver is that Java objects in the server always execute in the context of the database connection under which they were called; as a result, the driver does not permit you to explicitly open connections.
- **Embedded SQL in Java Translator:** The Java VM in the server provides an embedded SQL in Java translator. You can write stored procedures and functions to the higher-level, more productive SQLJ interface. The embedded

SQL in Java translator is transparently invoked by the Java VM in the server, which converts application code with embedded SQL statements into standard Java code with embedded JDBC calls. The resulting code can then be compiled by the embedded bytecode compiler of the Java VM and executed on its runtime facilities. At runtime, calls to JDBC are routed to the SQLJ runtime embedded within the database.

- **Inter-Language Method Services (ILMS):** ILMS is a common component within the database that manages all cross-language method dispatch between SQL, PL/SQL, Java, and with external procedures in C. SQL and PL/SQL call Java stored programs in the database through ILMS, the same mechanism that SQL uses to call PL/SQL in the database.
- **CORBA Object Request Broker:** The Oracle8*i* database integrates a Java-based CORBA 2.0 compliant Object Request Broker (ORB), Visibroker (Visigenics Java ORB), that enables you to call into and out of the database using the CORBA IIOP protocol. Java stored programs and Enterprise JavaBeans in Oracle8*i* can be invoked using IIOP. The database then behaves as a CORBA server. Similarly, IIOP callouts from Oracle8*i* enable the database to serve as a standard CORBA client. The ORB is used efficiently in Oracle8*i*; the Visigenics ORB components are used purely for IIOP protocol interpretation and object activation, and the Multithreaded Server kernel of the database is used for scalability. The database supports a variety of CORBA clients, including direct support for browsers such as Netscape 4.0 and middletier ORBs. Further, the CORBA facilities in the database are Java-friendly with a number of automated tools such as the automated Caffeine Java-to-IDL compiler.
- **Development Tool Support:** The Oracle8*i* Java VM provides a number of command line tools that simplify the use of the Java facilities in the database. Some of the facilities include tools to:
  - Load Java into the database and drop Java stored programs into the database
  - Automate the deployment and registration of CORBA servers in the database
  - Automate the packaging and deployment of Enterprise JavaBeans on Oracle8*i*

  These tools have themselves been integrated within the Oracle JDeveloper 2.0 tool.

Figure 5–2 Oracle Java VM Runtime Components



The Oracle8*i* Java Virtual Machine can be programmed in three specific ways:

- Java Stored Procedures
- CORBA Services Defined in Java
- Enterprise JavaBeans

Developing Java Applications with the Oracle Database

The Java VM in Oracle8*i* can be used to develop three types of Java applications:

- **Database Stored Procedures, Triggers, and Methods:** These support traditional database programmers and SQL-oriented clients of the database.
- **Enterprise JavaBeans:** The Java VM provides a transaction server platform for distributed Java components called Enterprise JavaBeans.
- **CORBA Servers in Java:** Oracle8*i* also enables distributed systems developers to implement CORBA servers in Java on the Java VM of the database.

The support for CORBA provided by the Oracle8*i* Java Virtual Machine creates an ideal platform for applications to communicate with each other in a synchronous, request-response manner. The Oracle8*i* JVM provides standard CORBA services including:

- A fully CORBA 2.0 compliant ORB
- Standard CORBA tools to develop CORBA applications in Java including an IDL compiler

This section contains:

CORBA Facilities in Oracle8*i*

In addition to the IIOP and ORB facilities described earlier in this document, several additional CORBA facilities are integrated with the database:

- **Transactions**: The Oracle8*i* Java VM provides a CORBA Object Transaction Service (OTS) API through the embedded JDBC driver, which has been extended to support OTS-visible transactions. The OTS provides transactional properties to CORBA Servers implemented on the Java VM.
- **Directory/Naming:** The Java VM offers a standard COSNaming interface to the industry standard directory service: LDAP (the Lightweight Directory Access Protocol). You can then access EJB components in the server registered with the CORBA ORB of the database through this standards-based directory service. It also integrates a CosNaming/LDAP compliant name service, which makes it easy to register and locate distributed components.
- **Object Adapter:** Oracle8*i* provides an Object Adapter for persistent CORBA Objects. It provides two important functions: Firstly, it serves as a directory of CORBA Objects published in the database; Secondly, it helps locate and load CORBA Objects upon initial activation by CORBA clients. It is implemented as a standard database table with three pieces of information: the name of the published CORBA Object, the schema name of the publisher, and the name of the Java class implementing the CORBA Object. Since it is a database table, it is hash indexed for scalability and fast access. It can also be split across multiple database partitions to support large numbers of CORBA Objects and Enterprise JavaBeans.
- **Java CORBA Services:** The Oracle8*i* database provides a number of features that make it easy for Java programmers to develop CORBA services. Firstly, it supports Caffeine, a direct Java to IIOP mapping that eliminates the need for IDL. Secondly, it supports standard objects by value and extensible structs, which is enormously useful for Java programmers. Finally, it provides a

number of other tools, including `java2iiop`, `idl2java`, and `java2idl`, which make application development simple.

- **Security**: CORBA services in the database execute in the same highly secure environment in which Enterprise JavaBeans execute, using the same three overlying layers of security: encryption through SSL over IIOP, authentication using traditional database username and password, and access control using roles and privileges.

See Also: *Oracle8i CORBA Developer's Guide and Reference*

The Oracle8*i* Java Virtual Machine also supports the Enterprise JavaBeans component model and a number of standard EJB servers.

Enterprise JavaBeans, an Overview

Enterprise JavaBeans (EJB) is a relatively coarse-grained set of Java application logic defined as a component with clearly specified interfaces that can be efficiently executed on a host system infrastructure provided by an EJB transaction server. EJB are not arbitrary Java classes, but must obey a set of constraints imposed by the hosting server. The EJB component model provides Java application programmers a convenient and highly productive component model for server-side business logic, facilitating application code reuse and multi-tier application development in the following important ways:

No Foreign Interface Definition Language (IDL): EJB defines a high-level infrastructure and programming interface, enabling application developers to specify their component model entirely in Java, without forcing the developer to learn a foreign IDL such as CORBA or D/COM.

No Systems-Level Programming: EJB is designed around the concept of a transaction server that is responsible for providing the system-level infrastructure and services such as scalability, load balancing, connection pooling, failover, and transactions. This frees the application programmer from low-level systems software concerns. The Transaction Server takes care of transaction and state management; as a result, programmers can declaratively specify the transactional behavior of EJB components.

Variety of Execution Environments: The EJB spec has been designed to allow EJB components to be deployed and executed in many different host environments that serve as EJB Transaction Servers, including TP Monitors, Application Servers, and databases. Oracle provides a consistent model for EJB components on the Java VM

integrated with Oracle8*i* and Oracle Application Server and sharing a common set of services between the two environments.

Variety of Communication Infrastructures: EJBs were also designed to isolate Java developers from the specific communications infrastructure that is used between the client, middletier, and database. Specifically, it enables Java developers to design and implement applications as EJB components, which are then portable to a variety of communications infrastructures including CORBA / IIOP, DCOM, and Java's own Remote Method Invocation (RMI) protocol. EJB applications are I-directionally compatible with any underlying infrastructure. Oracle has chosen to support CORBA / IIOP as its communication infrastructure.

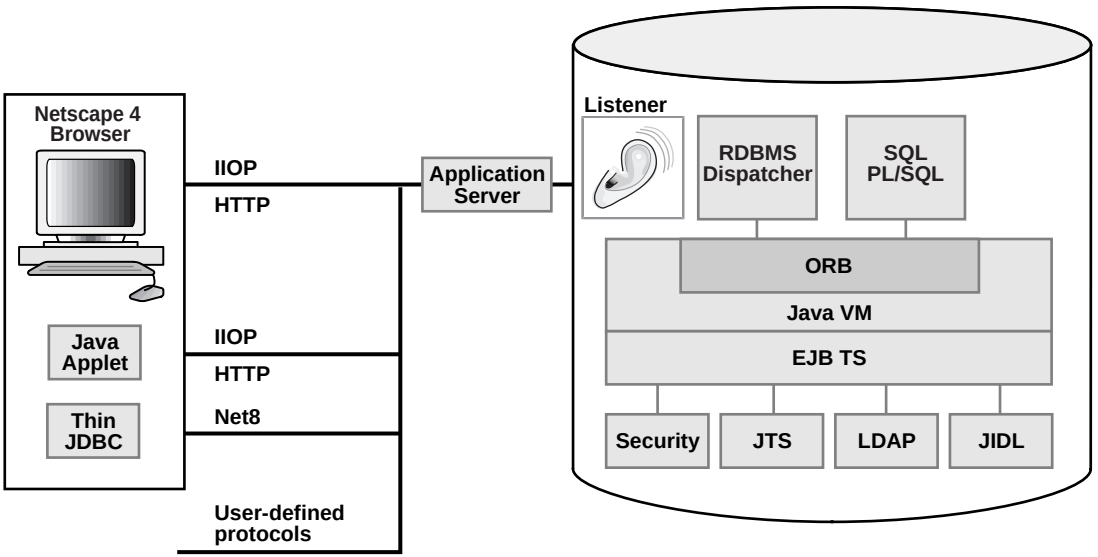
Productivity: EJBs provide a way to declaratively rather than programmatically specify transactional and security properties on a server-side Java component. Java tools, including JDeveloper, are designed to enable Java developers to directly support EJB-specified interfaces, providing an extremely productive environment for users to combine thin clients with server-side EJB components capturing their business logic.

Open Systems Alternative: In contrast with Visual Basic components executed within the Microsoft Transaction Server, the Enterprise JavaBean facilities of Oracle8*i* provide two important benefits: an integrated server that provides both unparalleled data management capabilities with support for Java-based business object logic and an open, portable application solution that does not lock customers into a proprietary, vendor-specific solution.

Supporting JABs on the Oracle8*i* Java VM: an Architectural Overview

The Oracle8*i* database provides a number of facilities to make it a highly productive and scalable EJB Transaction Server. Architecturally, it integrates several different components that have been illustrated in the figure.

Figure 5–3 Oracle8i Distributed Object Architecture



This section contains:

- Session Management Facilities
- Enterprise JavaBeans Services

Session Management Facilities

The session management facilities in the server provide support for the low-level registration of IIOP invocations, provide data-dependent routing of IIOP requests to sessions, and free related resources. These facilities are provided by several different components of the multithreaded server platform of the database including:

- **IIOP Listener:** The listener is a component within the Oracle8*i* database that accesses and translates the messages received by the database across the wire. Based on the messages decoded, it invokes the dispatcher of the database as

needed. The Oracle8*i* Listener has been extended to support the standard CORBA / IIOP binding protocol, enabling you to call into and out of the database using IIOP.

- **IIOP Dispatcher:** The dispatcher is a database component that schedules tasks and sessions within the database. With Oracle8*i*, the dispatcher has been extended to support IIOP-based method invocations. The Dispatcher in turn sends these messages to the CORBA 2.0 ORB for processing and dispatch.
- **CORBA 2.0 ORB Integration:** After the dispatcher invokes the ORB, the ORB marshals the IIOP method invoked, identifies the appropriate EJB component (which had been registered with the ORB) on which the invocation is to be executed, and then dispatches the message to the Java VM. The infrastructure of the ORB can also be used by EJB components in the database to call out to other distributed components, either CORBA components or EJBs on other EJB servers.
- **Multithreaded Server Platform:** The Oracle8*i* Java VM leverages the unmatched set of facilities that the Multithreaded Server provides, including load balancing, failover, data dependent routing, connection pooling, and shared memory management to make it a scalable EJB Server.
- **Java VM Execution Platform:** The Oracle8*i* Java VM provides the necessary execution environment for EJB components. Because they are standard Java application programs, they can leverage all the performance and scalability optimizations of the VM that were described earlier in this chapter.
- **Persistent State Access:** An EJB component in the data server accesses its persistent state: the state that persists across transaction boundaries through the JDBC driver or SQLJ translator integrated in the server. Its persistent state can be stored either as relational tables or as object relational tables.

Enterprise JavaBeans Services

In order to allow seamless interoperability across a multi-tier environment, Enterprise JavaBeans leverages a common set of Network Computing Services. The most important services include:

- **Transactions:** The Oracle8*i* Java VM provides a Java Transaction Service (JTS) API via the embedded JDBC driver that has been extended to support JTS-visible transactions. The JTS provides transactional properties to EJB components on the Java VM.
- **Directory Naming:** The Oracle8*i* Java VM offers a JNDI (Java Naming and Directory Interface) interface to any industry standard LDAP-enabled directory

service. EJB components in the server can be placed in the directory service, from which they can be accessed through JNDI.

- **Security:** The Oracle8*i* database accepts a variety of security mechanisms including Secure IIOP and Secure Sockets Layer (SSL) credentials which is used today by the Netscape 4.0 Browser and is the default standard in the Intranet and Internet environments.
- **Net8 Connection Manager:** The connection manager has been extended to multiplex IIOP connections in addition to Net8 connections.

See Also: *Oracle8i CORBA Developer's Guide and Reference*

6

Data Replication and Gateways

In a distributed database system, Oracle replication is a process that maintains multiple copies of the same data in separate Oracle databases. Replication captures and stores changes made to the data at one location before forwarding and applying them at each remote location. Users at each location see the same consistent view of the data. They need not access data remotely, as they would in a standard distributed database system. This chapter contains:

Oracle Replication, an Overview

Replication copies and maintains database objects, such as tables, in the multiple databases that make up a distributed database system. Oracle replication is a fully integrated feature of the Oracle database server; it is not a separate server.

Replication uses distributed database technology to share data between multiple sites, but a replicated database and a distributed database are not the same. In a distributed database, data is available at many locations, but a particular table resides at only one location.

For example, the `emp` table resides only at the DB1 database in a distributed database system that also includes the DB2 and DB3 databases. Replication means that the same data is available at multiple locations. For example, the `emp` table is available at DB1, DB2, and DB3.

This section contains:

Advantages of Replication

Some common reasons for using replication are:

- **Availability:** Replication improves the availability of applications because it provides them with alternative data access options. If one site becomes unavailable, users can continue to query or even update the remaining locations. In other words, replication provides excellent failover protection.
- **Performance:** Replication provides fast, local access to shared data because it balances activity over multiple sites. Some users can access one server while other users access other servers, reducing the load at all servers. Also, users can access data from the replication site that has the lowest access cost, which is typically the site that is geographically closest to them.
- **Disconnected Computing:** A snapshot is a complete or partial copy (replica) of a target master table from a single point in time. Snapshots enable users to work on a subset of a database while they are disconnected from the central database server. Later, when a connection is established, users can synchronize (refresh) snapshots on demand. When users **refresh** snapshots, they update the central database with all of their changes, and they receive any changes that may have happened while they were disconnected.

- **Network Load Reduction:** Replication can be used to distribute data over multiple regional locations. Applications access various regional servers instead of accessing one central server. This configuration dramatically reduces network load.
- **Mass Deployment:** Increasingly, organizations must deploy many applications that require the ability to use and manipulate data. With Oracle replication, deployment templates enable you to create multiple snapshot environments quickly. You can use variables to customize each snapshot environment for its individual needs. For example, use deployment templates for sales force automation; the template contains variables for various sales regions and salespersons.

Replication is ideally suited to synchronize data between systems in different locations so that each has a consistent view of the data without the necessity of accessing the data remotely.

Uses of Replication

Replication enables you to replicate tables and other supporting objects such as views, database triggers, packages, indexes, and synonyms by organizing logically related objects into **replication groups**. A replication object is a database object existing on multiple servers in a distributed database system. In a replication environment, any updates made to a replication object at one site are applied to the copies at all other sites. Use replication to:

- Replicate changes to data and schema from one master group to a number of snapshot groups
- Replicate changes to data and schema between two or more master groups
- Replicate changes to data from a snapshot group back to the master group
- Create read-only materialized views of data on another database
- Automatically resolve conflicting updates between master groups using pre-built methods
- Replicate and synchronize the running of replica procedures to update copies of datasets

You can ensure transactional integrity by choosing particular replication features and setting up the replication to occur synchronously. (Asynchronous replication is the norm.) Replication has its own set of data dictionary tables and views called the **replication catalog**. A replication management API consists of PL/SQL packages that can be used to administer the replication objects.

Types of Replication

The Oracle database supports three types of replication:

- Multimaster
- Snapshot
- Hybrid Configurations

Multimaster Replication

Multimaster replication (also called peer-to-peer or n-way replication) enables multiple sites, acting as equal peers, to manage groups of replicated database objects. Each site in a multimaster replication environment is a **master site**.

Applications can update any replicated table at any site in a multimaster configuration. Oracle database servers operating as master sites in a multimaster environment automatically converge the data of all table replicas and ensure global transaction consistency and data integrity.

Snapshot Replication

A snapshot contains a complete or partial copy of a target master table from a single point in time. A snapshot may be read-only or updatable. All snapshots:

- Enable local access, which improves response times and availability
- Offload queries from the master site, because you can query the local snapshot instead
- Increase data security by enabling you to replicate only a selected subset of the data set of the target master table

Hybrid Configurations

Multimaster replication and snapshots can be combined in hybrid or mixed configurations to meet different application requirements. **Mixed configurations** can have any number of master sites and multiple snapshot sites for each master.

For example, multimaster (or n-way) replication between two masters can support full-table replication between the databases that support two geographic regions.

Snapshots can be defined on the masters to replicate full tables or table subsets to sites within each region.

See Also: *Oracle8i Concepts*

Data Access Gateways

Oracle Corporation offers a number of products that you can use to facilitate communication with non-Oracle products and technologies. These are grouped under Data Access Gateways and fall into three broad categories: Oracle Transparent Gateways, Oracle Procedural Gateways, and Oracle Access Managers. Both the Transparent Gateways and the Procedural Gateways enable you to access non-Oracle technology from an Oracle environment. The Oracle Access Managers enable you to access Oracle technology from a non-Oracle environment.

This section contains:

Oracle Transparent Gateways

Oracle Transparent Gateways gives applications access through transactional SQL to non-Oracle data stores such as DB2, IMS, and DB2/400.

Oracle Transparent Gateways provides:

- Access to data as if it resided in a single, local, relational database
- Dynamic SQL access to the data
- The ability to join tables from different sources with a single SQL statement

OTG provides two major IBM-oriented transparent gateways for DRDA and EDA/SQL, a transparent gateway for ODBC, and transparent gateways for Sybase, Infomix, Ingres, Microsoft SQL Server, HP IMAGESQL, Digital RMS, and Rdb.

The Oracle Transparent Gateway for IBM DRDA is the Oracle implementation of the IBM DRDA architecture. It provides Oracle applications with read and write

access to DRDA server databases, including DB2 OS/390, SQL/DS, DB2/400, and DB2/UDB.

With the Transparent Gateway for EDA/SQL running on OS/390, you can access the majority of non-Oracle databases and file systems, including VSAM, IMS, ISAM, Teradata, ADABAS and DB2.

Oracle Procedural Gateways

The Procedural Gateways offer programmatic access to non-Oracle transactions from within the context of an Oracle transaction. They are IBM-oriented and fall into two types: Procedural Gateway for APPC and Procedural Gateway for MQSeries.

Because the Procedural Gateway for MQSeries is message-oriented, we cover it in Chapter #, "Advanced Queueing" (hotlink).

Oracle Procedural Gateway for APPC

The Oracle Procedural Gateway for APPC uses native IBM APPC / LU6.2 mechanisms to execute mainframe transactions through a transaction monitor such as IBM CICS, IBM IMS/TM, or CA–IDMS/DC.

These transactions access a variety of databases and file systems, including:

- IMS
- DB2
- VSAM
- CA–IDMS/DB
- ADABAS
- Model 204

Procedural Gateway for APPC enables:

- Sharing of transactions between Oracle and non–Oracle applications
- Support for multiple transaction monitors and multiple operating systems from a single UNIX–based technology:
  - MVS: CICS, IMS/TM, CA–IDMS/DC, and APPC/MVS
  - CICS/VM
  - DOS/VSE: CICS

- AS/400: CICS/400

Oracle Access Managers

Used in combination with the Transparent Gateways, Access Managers enable mainframe applications to access virtually any data store. The Access Managers let you use existing OS/390 and AS/400 applications by providing them with access through SQL to data in the Oracle Server. The Access Manager for AS/400 enables AS/400 applications written in RPG, COBOL, and C to access data stored in Oracle.

The Oracle Access Managers for CICS and IMS/TM are part of the Oracle for MVS Client Solution, which enables you to access data in Oracle from TSO, batch, CICS, and IMS/TM programs:

- You can start a TSO session and have direct interactive access to any Oracle database server anywhere in the enterprise.
- Batch processes update remote Oracle servers directly, eliminating the need to set up complicated file-transfer mechanisms.
- CICS and IMS/TM transactions update Oracle databases and other recoverable resources. The transaction is protected by a two-phase commit using SYNCPOINT processing.

Uses of Oracle Replication and Gateways

An original method of integrating applications was to use **data level integration to** maintain multiple copies of the data in the different applications and to periodically synchronize the copies. If the requirements for integration are relatively basic, then data level integration provides a simple, cost-effective, and manageable solution.

Solutions that successfully employ data level integration have most or all of the following characteristics:

- All applications and data are owned and controlled by a single organization.
- There are a relatively small number of application instances (less than 20).
- Applications are mostly based on relational technology.
- Applications are predominantly based on the Oracle database.
- Data Structures (`customer`/`account`/`transaction` tables) are broadly similar in all applications.
- Each data item is created and changed by only one application.

- Ratio of Read Access to Data Change Access is relatively high.
- Extensive real-time or near-real-time synchronization of data is not required.
- The architecture and applications to be integrated are not continually changing.

Classic solutions that use this style of integration include:

- Mobile applications that can be synchronized infrequently with back-end systems
- Web-based applications requiring read-only data to be cached in the middle tier
- Centrally-managed List-of-Values reference data that must be distributed to OLTP applications

The Oracle products best suited to providing this level of integration are Oracle Replication and Oracle Data Access Gateways.

Oracle Replication enables asynchronous synchronization of data across distributed Oracle databases using **journal logs** to capture both changes made to the master copy of table structures and row-level data changes. The logs then apply the changes to the snapshot copies of the tables.

Oracle Replication is a mature product that contains numerous advanced features that extend this basic functionality to:

- Manage and synchronize multiple master copies
- Resolve conflicting data changes
- Provide real-time replication of data changes as part of the instigating transaction
- Synchronize the running of replicated stored procedures

In a data-level integration solution, Oracle Replication is the mainstay of integration between applications based on the Oracle database.

Interoperability

Oracle Corporation provides a number of transparent gateways that enable seamless integration to other databases in Oracle-controlled transactions. For transactions that are not controlled by the Oracle transaction manager, use the Access Managers to enable data in Oracle databases to be accessed by non-Oracle-controlled transactions.

7

Oracle Advanced Queuing and JMS

This chapter contains:

A Brief Review of the Products

This section examines several Oracle products that are involved in Oracle Integration Server. It contains:

Advanced Queuing

Advanced Queuing (AQ) is a database-integrated message queuing component of the Oracle8*i* Enterprise Edition database management system. It provides an infrastructure that simplifies the task of passing messages within an application or between applications.

Its functional strong points include:

- Store and forward capability
- Simple message management
- Recovery to point-of-failure
- Transactional integrity
- Guaranteed message delivery between databases
- Message mining
- Message tracking
- A choice of programming interfaces (APIs) to place messages in and remove them from queues

Components of Advanced Queueing

Advanced Queuing has several key components:

Message

A message is the smallest unit of information inserted into and retrieved from a queue. A message consists of:

- Control information (metadata)
- Payload (data)

The control information represents message properties used by AQ to manage messages. The payload data is the information stored in the queue and is transparent to Oracle AQ. A message can reside in only one queue. A message is created by the **enqueue call** and consumed by the **dequeue call**.

Queue

A **queue** is a repository for messages. There are two types of queues: **user queues**, also known as normal queues, and **exception queues**. The user queue is for normal message processing. Messages are transferred to an exception queue if they cannot be retrieved and processed. Queues can be created, altered, started, stopped, and dropped by using the Oracle AQ administrative interfaces.

Queue Table

Queues are stored in queue tables. Each **queue table** is a database table and contains one or more queues. Each `queuetable` contains a default exception queue.

Agent

An **agent** is a queue user. This could be an end user or an application.There are two types of agents:

- Producers who place messages in a queue (enqueuing)
- Consumers who retrieve messages (dequeuing)

Any number of producers and consumers may be accessing the queue at a given time. Agents insert messages into a queue and retrieve messages from the queue by using the Oracle AQ operational interfaces.

An agent is identified by its `name`, `address` and `protocol`.

- The `name` of the agent may be the name of the application or a name assigned by the application. A queue can itself be an agent, enqueuing or dequeuing from another queue.

- The address field is a character field of up to 1024 bytes that is interpreted in the context of the protocol. For instance, the default value for the protocol is 0, signifying a database link address. In this case, the address for this protocol is of the form

```
queue_name@dblink
```

where queue_name is of the form [schema.]queue and dblink can be either a fully qualified database link name or the database link name without the domain name.

Recipient

The recipient of a message may be specified by its name only, in which case the recipient must dequeue the message from the queue in which the message was enqueued. The recipient may be specified by name and an address with a protocol value of 0. The address should be the name of another queue in the same database or another Oracle8*i* database (identified by the database link) in which case the message is propagated to the specified queue and can be dequeued by a consumer with the specified name. If the recipient's name is NULL, the message is propagated to the specified queue in the address and can be dequeued by the subscribers of the queue specified in the address. If the protocol field is nonzero, the name and address field are not interpreted by the system and the message can be dequeued by a special consumer.

Recipient and Subscription Lists

A single message can be designed for consumption by multiple consumers. There are two ways to do this:

- The enqueuer can explicitly specify the consumers who may retrieve the message as recipients of the message. A recipient is an agent identified by a name, address and protocol.
- A queue administrator can specify a default list of recipients who can retrieve messages from a queue. The recipients specified in the default list are known as **subscribers**. If a message is enqueued without specifying the recipients, the message is implicitly sent to all the subscribers.

  Different queues can have different subscribers, and the same recipient can be a subscriber to more than one queue. Further, specific messages in a queue can be directed toward specific recipients who may or may not be subscribers to the queue, thereby overriding the subscriber list.

Rule

A rule is used to define the interest of one or more subscribers in subscribing to messages that conform to that rule. The messages that meet this criterion are then delivered to the interested subscribers. Put another way: a rule filters for messages in a queue on a topic in which a subscriber is interested.

A rule is specified as a Boolean expression (one that evaluates to true or false) using syntax similar to the `WHERE` clause of a SQL query. This Boolean expression can include conditions on the following:

- Message properties (currently priority and correlation identifier)
- User data properties (object payloads only)
- Functions (as specified in the where clause of a SQL query)

Rule-Based Subscriber

A rule-based subscriber has rule associated with it in the default recipient list. A rule-based subscriber is sent a message that has no explicit recipients specified if the associated rule evaluates to `TRUE` for the message.

Queue Monitor

The queue monitor (`QMNn`) is a background process that monitors the messages in the queues. It provides the mechanism for message delay, expiration, and retry delay. The `QMNn` also performs garbage collection for the queue table and its indexes and index-organized tables.

It is possible to start a maximum of 10 multiple queue monitors at the same time. You start the desired number of queue monitors by setting the dynamic `init.ora` parameter to `aq_tm_processes`. The queue monitor wakes up every minute, or whenever there is work to be done, for instance, if a message is to be marked as expired or as ready to be processed.

General Features of Advanced Queueing

AQ uses the strength of the Oracle8*i* database management system (DBMS) to provide these general features:

SQL Access

Messages are placed in normal rows in a database table, and so can be queried using standard SQL. This means that you can use SQL to access the message

properties, the message history, and the payload. All available SQL technology, such as indexes, can be used to optimize the access to messages.

Integrated Database Level Operational Support

Standard database features such as recovery, restart, and enterprise manager are supported. Oracle AQ queues are implemented in database tables, hence all the operational benefits of high availability, scalability, and reliability are applicable to queue data. In addition, database development and management tools can be used with queues. For instance, queue tables can be imported and exported.

Structured Payload

Users can use object types to structure and manage message payloads. RDBMSs in general have had a far richer typing system than messaging systems. Since Oracle8*i* is an object-relational DBMS, it supports both traditional relational types as well as user-defined types. Many powerful features are enabled as a result of having strongly typed content, such as content whose format is defined by an external type system. These include:

- **Content-based routing:** An external agent can examine the content and route the message to another queue based on the content.
- **Content-based subscription:** A publish and subscribe system built on top of a messaging system which can offer content based on subscription.
- **Querying:** The ability to execute queries on the content of the message enables message warehousing.

Retention and Message History

Users of AQ can specify that messages be retained after consumption. The systems administrator can specify the duration for which messages will be retained. Oracle AQ stores information about the history of each message, preserving the queue and message properties of delay, expiration, and retention for messages destined for local or remote recipients. The information contains the `ENQUEUE/DEQUEUE` time and the identification of the transaction that executed each request. This enables users to keep a history of relevant messages. The history can be used for tracking, data warehouse and data mining operations.

Tracking and Event Journals

If messages are retained, they can be related to each other. For example, if a message `m2` is produced as a result of the consumption of message `m1`, `m1` is related to `m2`. This allows users to track sequences of related messages. These sequences represent

event journals, which are often constructed by applications. Oracle AQ is designed to let applications create event journals automatically.

Integrated Transactions

The integration of control information with content (data payload) simplifies application development and management.

Queue- Level Access Control

With Oracle8*i*, an owner of an 8.1 style queue can grant or revoke queue- level privileges on the queue. DBAs can grant or revoke new AQ system level privileges to any database user. DBAs can also make any database user an AQ administrator.

Non-Persistent Queues

AQ can deliver non-persistent messages asynchronously to subscribers. These messages can be event-driven and do not persist beyond the failure of the system (or instance). AQ supports persistent and non-persistent messages with a common API.

Publish-Subscribe Support

A combination of features enable a publish-subscribe style of messaging between applications. These features include rule-based subscribers, message propagation, the listen feature, and notification capabilities.

Two Contexts for Developing Queueing Operations

Oracle AQ offers two development contexts:

- Native AQ, which you can access by means of three different programmatic environments:
  - From PL/SQL using the `DBMS_AQ/AQADM` PL/SQL Packages
  - Visual Basic using Oracle Objects for OLE (OO4O)
  - Java using the `oracle.AQ` Java Package
- Java Messenging Service (JMS)

  Java using the `oracle.jms` Java Package. This implementation of a public standard extends the defined W3C interfaces so that developers operating in the JMS context have the same means as those working within native AQ.

A comprehensive graphical interface supports the administration of Advanced Queuing objects through the Enterprise Manager DBA Studio (Schema Management), part of the DBA Management pack, which is in the 8.1.7. database release.

See Also:

- *Oracle8i Application Developer's Guide - Advanced Queuing* for a full definition of the features of Advanced Queueing and examples of how to use many of them
- *Oracle8i Supplied PL/SQL Packages Reference*
- *Oracle8i Supplied Java Packages Reference*
- Online Help for Oracle Objects for OLE

Oracle Java Messaging Service (OJMS)

Oracle Corporation has extended the JMS API to leverage those Oracle Advanced Queuing features that cannot be accessed through a standard JMS API. We are working with the standards body to include these extensions in the JMS standard.

We have extended the standard JMS API in the following areas:

Agents

The standard JMS definition expects each message to be delivered to a single recipient. Because AQ can record, manage, and track the delivery of a message to more than one recipient without having to copy the message, the OJMS API supports the use of **agents.**

An agent is associated with the message for each recipient of the message. The agent is an object that has a name, a transport protocol, and an address that is specific to the transport protocol.

The agent model is particularly important for the publish-subscribe capability of AQ and for the interoperability of AQ with other messaging technologies.

Additional Message Control Properties

To use standard AQ features that are not supported by the standard JMS message properties definition, OJMS has extended the message control property set to include:

- **Delay interval**: The creator of the message can specify that the message is not to be made available until a specified period of time (the delay) has elapsed. When you use this with the standard JMS expiration feature, you can define a window during which the message must be **consumed** (successfully collected by all the recipients). This is particularly useful for events that must be recorded.
- **Exception handling**: AQ automatically moves messages that expire or that cannot be consumed within a specified number of attempts into an exception queue. The exception handling property enables you to specify the exception queue, which extends the standard JMS expiration feature.

Additional Message Type

AQ is unique among messaging technologies in supporting strongly typed message payloads that you can access using standard SQL while they are in the queues and that support content-based routing. AQ uses Oracle Object Types (also known as ADTs) to do this. To support this feature, OJMS has an additional message type called an ADTMessage.

Transactional Session

AQ messages can be created within the context of an Oracle database `COMMIT` transaction. The same transaction can include SQL operations such as table inserts and `SAVEPOINT`s without requiring a two-phase or XA-compliant commit.

We extended the JMS Transacted Session feature to support this unique Oracle feature, which is useful in developing adapters to interface to Oracle-based applications.

Administration

The JMS standard includes a concept of **administered objects.** Chapter 4, "Key Integration Concepts", describes Oracle JMS implementation in depth. The Oracle JMS implementation has an API for creating and managing these objects.

The Session interface now supports administrative functions such as the creation of `Queue Tables` and `Queues` and `Topics` and the modification of their properties.

The Destination interface now supports administration of `Queues` and `Topics` and AQ propagation of messages between destinations. This has significance to a hub and spoke architecture that is used for the hardware configuration, security model, or persistence schema.

Restrictions

The first release of the OJMS interface (with release 8.1.6 of the database) has a small number of standard JMS features that are not supported or are supported in a non-standard manner:

- Temporary queues
- Non-durable subscribers
- Message filtering on JMS Queues (AQ single-consumer queues). However, OJMS support for filtering with durable subscribers is actually richer than the standard JMS definition.
- Access to administered objects through JNDI. The OJMS extensions to the administration interface provides this functionality.

Oracle Procedural Gateway for IBM MQSeries

Oracle Procedural Gateway for IBM MQSeries enables Oracle applications to create MQSeries messages directly or to convert AQ messages into MQSeries messages and vice-versa. It has a graphical interface called Visual Workbench that you can use to define the mapping of Oracle object types to MQSeries structures and to import Oracle Object Types and COBOL Copybook definitions.

You then complete the configuration by generating PL/SQL packages that provide a seamless transactional API to the MQSeries queue that you can access from SQL*Plus, PL/SQL packages, Pro*Series languages, and so on.

TIB Adapter for Oracle

TIBCo sells an adapter that you can use to transform AQ messages into TIB Rendezvous messages and vice-versa. The product was developed in collaboration with Oracle and provides a code-free transformation solution between Oracle Advanced Queuing and TIB Rendezvous (TIBrv).

The software is designed to convert messages from AQ to TIBrv, to transport them across the TIB bus, and to convert them back into AQ without losing any of the AQ message properties. Messages are converted transactionally using a version of the TIB Rendezvous Certified Messaging and the Oracle commit model.

Applying the Products in an Integration Solution

This section examines the ways in which these products interface in an integration solution. It contains:

- Advanced Queuing
- OJMS
- Procedural Gateway for MQSeries
- Interoperability

Advanced Queuing

Business Event Integration

Within an integration solution based on business events, the AQ message queuing functionality provides the backbone for the solution and supports business event requirements not previously expected of messaging technologies such as:

- Tracking the recipients' consumption of messages by keeping a history
- Recording messages and their history persistently so that they can be recovered in the event of a system failure
- Retaining messages and history after consumption to form a provable audit trail
- Providing simple query access to messages for message mining
- Delivering an architecture that performs consistently even when messages are retained
- Enabling you to define and apply sophisticated routing rules

Advanced Queuing supports business events by using the Oracle8*i* database to provide a repository for messages, transactional support, and recovery. Most of the other products in the Oracle Integration Server can be accessed directly through a delivered AQ interface. This reinforces the position of AQ as the backbone of the solution.

AQ keeps a history by determining and recording the recipients of the message when the message is created. Each time a recipient attempts to collect the message, the attempt is recorded automatically. This guarantees that a message is collected only once by each recipient and that you always know when a message has been collected.

AQ messages have three main components: a header, a payload, and a history. All three are stored in queue tables that consist of a set of regular database objects in the database. Provided that the database is protected by a suitable backup and recovery routine, messages can be recovered to a consistent state in the event of a system failure.

AQ messages are accessed through an API that prevents the message header, payload, or history from being changed directly by the user accessing the message. This design feature, when combined with the retention of messages after they have been consumed, enables you to construct an audit trail.

Although AQ uses a sophisticated set of database objects to implement this rich functionality, it also includes a set of views for each queue table so that you can easily interrogate the header, payload, and history of the messages using SQL.

If the payload structure is defined as a simple object type, SQL can access the attributes directly. However, if a generic object type containing a CLOB is used to store the payload in, for example, XML format, then the character string must be translated into a more strongly typed structure before it can be interrogated easily.

As already mentioned, AQ uses a set of regular database objects to implement the logical construct of a queue table. We have engineered this set of objects to ensure that retaining consumed messages does not adversely affect the creation or collection of other messages.

The ability to determine the recipients of messages according to predefined rules is comprehensively supported by Advanced Queuing.

See Also: Chapter 4, "Key Integration Concepts"

Multiconsumer queues provide publish-subscribe routing for durable subscribers:

- Subject-based routing (Chapter 4, "Key Integration Concepts") is provided by defining a queue for each subject and a subscription list of recipients for the queue.
- Content-based routing (Chapter 4, "Key Integration Concepts") is provided for queues with object type payloads through the use of subscription rules in the subscription lists. The subscription rule can include references to the payload by prefixing attribute names with the constant value `tab.user_data.`
- Rule-based routing (Chapter 4, "Key Integration Concepts") can be achieved by specifying a subquery in the rule.

You can also use propagation to route messages to groups of recipients within a publish-subscribe context as shown in the following scenario:

1. Recipients receive messages about red cars by subscribing to Queue `Red Cars`.
2. Queue `Red Cars` subscribes to Queue `Sports Cars` with a rule of `tab.user_data.color = RED`.
3. Publisher creates a message in Queue `Sports Cars`.
4. If a car has the attribute color set to `RED`, Queue `Red Cars` is recorded as a recipient of the message.
5. AQ propagates the message to the Queue `Red Cars` and records all the recipients on the subscription list as recipients of the propagated message.

Advanced Queuing is the most appropriate mechanism for routing if the subscribers are durable but change frequently (daily or more often).

If the subscription is static and the rules for routing are complex, Workflow (Chapter 10, "Workflow") provides a better alternative to develop your solution.

Data integration

If Oracle Replication does not provide the required functionality, use Advanced Queuing technology to develop a predefined replication environment.

You can do this in two ways:

- Define database triggers to create AQ messages when changes are made to rows in database tables or to the propagation used to transfer the messages to the destination databases. When they are located in queues in the destination databases, you can remove the messages and apply them to copies of the table.
- Use non-persistent queues to capture the DML and DDL changes as database events. The messages must then be processed using OCI.

Developing your own replication software using AQ requires a special understanding of replication issues and a commitment to maintain and upgrade predefined software.

OJMS

If you use Java as the development language for the programmable aspects of the solution and if you employ JMS as a standard for communicating with messaging technologies, then OJMS offers an effective API to Advanced Queuing.

In a B2B-oriented solution, you can use AQ features additional to those included in the JMS standard, such as object type payloads, transacted sessions, and the agent model. The OJMS extensions to the JMS standard make it possible to use these extra

AQ features. If you use these features, OJMS offers an excellent alternative to the JMS implementation of Oracle Message Broker.

Procedural Gateway for MQSeries

The Procedural Gateway for MQSeries is the only gateway to another messaging technology offered by Oracle.

Use it to transform messages between MQSeries and Advanced Queuing. In its current form, this requires you to develop a simple PL/SQL bridging procedure to call the MQSeries dequeue package or the AQ enqueue API. It supports only the conversion of simple payloads and does not support collections or LOBs.

The message throughput across a single queue bridge varies depending on system specification and message size. Do not choose Procedural Gateway for MQSeries if you anticipate hundreds of messages per second.

Interoperability

The products that make up the Oracle Integration Server interface smoothly with Advanced Queuing to provide asynchronous services comparable to those provided by many third-party application integration tools.

However, some applications require interaction with other messaging technologies, such as MQSeries and Rendezvous from TIBCo, because of a decoupling requirement, an application platform, an existing interface, or a previous choice of strategic technology.

If the application has an interface that extracts messages to a messaging technology other than AQ, you must decide whether to integrate the messaging technology to AQ, OMB, or Workflow directly by using a programming language or through specialized software.

Alternative methods for integrating to Advanced Queuing are:

Programmatic 1: If both message queuing products have an API for a particular language, write a program that collects the message from one product, transforms the message properties, and creates the message in another product.

Programmatic 2: Choose two languages that can communicate with each other and that have APIs to message queuing products.

Software: Choose a specialized product that enables conversion between the two messaging products.

MQSeries Example

The example given here is used solely to illustrate the process and should not be taken as a recommendation of any one solution over any other.

Scenario

One of the applications included in a hub-and-spoke integration solution has a ready-made message-based interface that places messages in a set of MQSeries queues and that expects business events in the form of MQSeries messages. The application interface is written in C and creates and consumes messages using the native MQSeries API.

The objective is to transport messages from MQSeries to AQ and from AQ to MQSeries.

The solution depends upon the architectural principles, the language or languages used in the integration solution, the approach to modularization, the standards used, and so on.

If the second message queuing product is TIB Rendezvous from TIBCo rather than MQSeries, then the scenarios are similar except that TIBCo and not Oracle is the software provider and the software is called The TIB Adapter for Oracle.

Business Intelligence and Message Warehousing

As discussed earlier in this chapter, Advanced Queuing can manage two types of queues: persistent queues and nonpersistent or volatile queues.

This section contains:

Persistent Queues

Persistent queues are ideally suited to manage messages that flow between applications (when integrating applications with an enterprise) or between two

business processes (when integrating businesses for B2B commerce). There are two reasons why such messages must be stored persistently:

- **Auditing and tracking:** Messages that are sent from one application or business to another must be stored so that they can be audited and tracked for dispute resolution. Typically, three kinds of queues are run for message auditing:
  - Queues against the message destinations to determine what information flows where
  - Queues against the message headers (also called subjects or topics) to determine the type of information sent
  - Queues against the message payloads or contents to determine the actual information sent

  Particularly for B2B commerce, auditing and tracking of messages is a critical requirement.
- **Guaranteed once-only, in-order delivery of messages:** A critical requirement for B2B and EAI messaging is that the message infrastructure guarantee delivery of messages in order and only once.

The message payload of AQ can also be an unstructured data type such as `RAW` or `BLOB` for flat-file-oriented data such as SAP TDOCs and XML-based business object documents. Consider three issues when warehousing message payloads:

- **Storing message headers:** You can store message headers from AQ payloads in relational or object relational tables. If a message is received in XML format, you can selectively parse it and convert its header information to SQL format and store it in a table. In determining whether to parse and store and how to do so, keep two trade-offs in mind:
  - Converting message headers to SQL format expedites frequent querying of messages for analysis.
  - Storing untransformed headers and payloads expedites frequent auditing.
- Storing message payloads: You can store structured message payloads either as relational tables or as object relational tables and query them using standard SQL. You can store unstructured payloads in either `RAW` or `BLOB` format. If you receive B2B messages as XML payloads, you must decide how completely to parse the payload. If you must perform analytical operations frequently, you must parse the payload. If the payload is large, construct a DOM parse tree; if the payload is small, use a SAX API. (Typically a payload that is greater than 100 MB is considered large.) If only a small set of fields within the XML document require analysis, then you extract only those fields from the parse

tree and store them in tables in SQL format. Retain the rest of the payload in RAW or BLOB format.

For more information on the Oracle XML facility, read the Oracle8*i* XML Reference Guide Release 3 (8.1.7)

- **Use of partitioning and rolling windows:** Because a message warehouse looks similar to a data warehouse, features applicable to data warehousing are also applicable to message warehousing. Most importantly, in a B2B commerce environment, you can store and manage messages received each day in a separate database partition. Two benefits of such partitioning include:
  - You can create and execute analytical calculations such as aggregates and moving averages, as well as indexes, at the partition level. For instance to analyze total volume of B2B messages on a daily basis, simply run SORT and SUM operations on the data in the partition for that day.
  - Partitioning messages on a daily basis enables you to move individual messages online and offline. You maintain a rolling window in which you add new partitions and take existing partitions offline.

The messaging infrastructure must maintain a persistent message store to guarantee message delivery even when message propagation or transport fails.

Volatile Queues

In contrast to a persistent messaging environment, volatile queues are ideal for best-effort delivery of messages. For instance, a message sent to confirm that a B2B message has been received does not need to be stored persistently.

Basic Principles of Message Storage

As previously described, each message passed through AQ is stored in a queue. You can store the contents of single or multiple queues in a queue table. Four basic principles govern the storage of messages for business analysis and intelligence:

- Message payload structure
- Message logging facility
- Message warehousing architecture
- Query, analysis, and reporting facilities

Business Intelligence Tools

The Oracle Business Intelligence Tools are a set of three Oracle products that provide simple, intuitive access to business information in Oracle databases. Use them to analyze business intelligence stored in message warehouses.

Oracle Reports, Oracle Discoverer, and Oracle Express have graphical interfaces that can be accessed from anywhere through a standard Internet browser. They deliver decision support information to business users (usually from a data warehouse or data mart). Recently, we've enhanced them to enable operational monitoring for throughput analysis and service level monitoring in an asynchronous messaging integration solution.

The tools can be used to generate and view dynamically created reports, to perform dynamic queries against virtual and materialized data structures, and to analyze and forecast trends.

Reports

Oracle Reports is a graphical tool for developers of sophisticated, high-quality reports. Its rich functionality enables reports with complex hierarchic structures and matrices to be defined quickly and easily, using the full power of SQL.

Users can then view these dynamically generated reports through a standard web browser.

Discoverer

Oracle Discoverer is an easy-to-use browser-based tool, primarily aimed at business users, that can be used to perform dynamic queries, to provide reports and charts, and to enable Web publishing.

It employs the End User Layer, an abstraction layer over database objects, to simplify queries and enable predefinition of queries. It provides drag-and-drop functionality that removes the need for the user to be versed in the grammatical constructs of SQL.

Powerful, integrated charting illustrates trends and exceptions and enables users to drill down through charts to view specific data more closely.

Some of its more sophisticated features include:

- **Advanced Query Prediction**: Users are informed of the likely execution time of submitted queries.

- **Intelligent Summary Redirection**: The tool automatically optimizes queries to make use of any pre-built summary tables that can be used to reduce the execution time.
- **Designer Generation**: The End User Layer can be maintained and version-controlled using the Oracle Designer.

Express

Oracle Express is a decision-support tool with an online analytical processing (OLAP) engine. It employs a multidimensional data model that expands rows and columns into multiple categories of data called dimensions.

This type of data model is optimized for decision support, uses array arithmetic to provide quick access to the data for “slice and dice” type queries, and offers built-in functions for analysis, forecasting, modeling, and what-if scenario playing.

The data model can be interrogated using intuitive tools that are accessible through a standard Web browser. The OLAP engine can use a data cache to store dimensional data as an aid to query performance.

8

Oracle Message Broker and JMS

This chapter introduces the Oracle Message Broker and contains these sections:

Overview

Oracle Message Broker is a Java-based message management subsystem that provides a message brokering facility to major message queuing systems including AQ, the IBM MQSeries, and the TIBCo Rendezvous. Its drivers provide a consistent, open, JMS-compliant API for these message queuing systems.

OMB provides its own proprietary volatile queuing and propagation. You can store its administrative metadata in a local or a remote LDAP server so that it operates independently of an Oracle database instance. OMB supports the JMS standard publish-subscribe, topic-based routing and supports both durable and non-durable subscribers. You can apply filters to the message control properties. OMB supports JMS transacted sessions.

The Oracle Message Broker consists of these components:

- Oracle Message Broker Core
- Drivers
- Administrative Components and the LDAP Directory
- Client Programming Interface
- Adapter Developers Toolkit

Oracle Message Broker Core

The Oracle Message Broker Core acts as a **JMS provider.** See *Chapter 4, "Key Integration Concepts"* as defined in the JMS standard.

Oracle Message Brokers in different locations communicate with one another and coordinate message transmission between brokers.

The Oracle Message Broker Core provides:

- Support for pushing messages to **JMS Clients.** See *Chapter 4, "Key Integration Concepts"*.
- Polling of message queuing systems that do not support message notification
- Core-based administrative tasks
- Multiplexing of JMS Client connections onto the underlying message queuing systems
- Standard drivers that provide a JMS-based abstraction layer over the message queuing systems APIs

Drivers

Oracle Message Broker provides five drivers that provide a JMS API to underlying message queuing systems. It uses the message queuing systems to provide persistence and message management.

The drivers also coordinate the translation of the messages into the native storage formats required for the message queuing systems to store the messages.

The drivers are:

- **Oracle Advanced Queuing Driver,** an alternative to the OJMS API into Oracle8*i* Advanced Queuing
- **Oracle Volatile Driver,** fast delivery of JMS messages using lightweight, in-memory, communication facilities. The Volatile Driver is useful for high throughput of messages if the messaging system does not require persistent message storage.
- **MQSeries Driver**, support for most of the features of this widely used commercial messaging system
- **Oracle Multicast Driver,** fast delivery of JMS messages using lightweight, multicast communication facilities. The Multicast Driver uses the Oracle Application Server Multicast Communication libraries.
- **TIBCO Driver,** fast delivery of transient messages based on lightweight multicast communication facilities. The TIB Rendezvous (TIBCO) Driver is written to work with TIB Rendezvous release 5.x, or TIB/Rendezvous Pro release 5.x.

Administrative Components and the LDAP Directory

The Oracle Message Broker uses an LDAP directory for storing and accessing administrative information. It includes a command line tool and a graphic tool for the administration and monitoring utilities. It also has a performance monitoring tool based on the ORB Dynamic Monitoring Service (DMS) of the Oracle Object Request Broker.

Client Programming Interface

The client programming interface is the set of services offered by the Oracle Message Broker to Java client programs and is often referred to as the JMS API. A set of C++ class libraries offers a similar programmatic interface for C++ programmers.

Adapter Developers Toolkit

Oracle Corporation provides a Java-based developers toolkit as part of Oracle Message Broker version 2 that you can use to develop adapters. The kit provides a framework that lets you develop programs of a standard construction and that supports three types of access to applications:

- Through the application API
- Through SQL
- By collection from interface files

The toolkit enables you to link the application and queuing technology through JMS `Queues/Topics`, providing transaction management and basic exception management. The developer extends the framework by adding application API-specific calls, file-based access, or a JDBC connection, depending on the type of access to the application that is required.

The framework for SQL access converts XML-formatted messages dequeued from a JMS `Queue/Topic` to Business Components for Java (BC4J). SQL that is defined within the BC4J object applies changes to the application over a JDBC connection.

Uses of OJMS and OMB

Oracle Corporation offers Java Messaging Services (JMS) through two products: Oracle Java Messaging Services and Oracle Message Broker. You must understand the differences between the two products to ensure that you make the best use of the products in the context of the specific requirement.

This section contains:

OJMS offers an AQ-only implementation of JMS with AQ extensions, and consequently it is inherently closely coupled with the database.

OMB offers non-extended Java Messaging Services over all the major messaging technologies including AQ and the OMB proprietary queuing mechanism. OMB is designed to use minimal space and to use LDAP services for its metadata. Its

proprietary queuing mechanism (called volatile queues) enables it to offer non-durable subscription to JMS Topics.

The AQ driver that enables OMB to use AQ as a message store includes no extensions that enable you to use AQ features that do not appear in the JMS standard. However, OMB offers publish-subscribe routing using JMS Topics and applies message filtering at the point of message collection. The OJMS driver does not offer message filtering.

Consider using Oracle Message Broker if these factors are functionally or architecturally important:

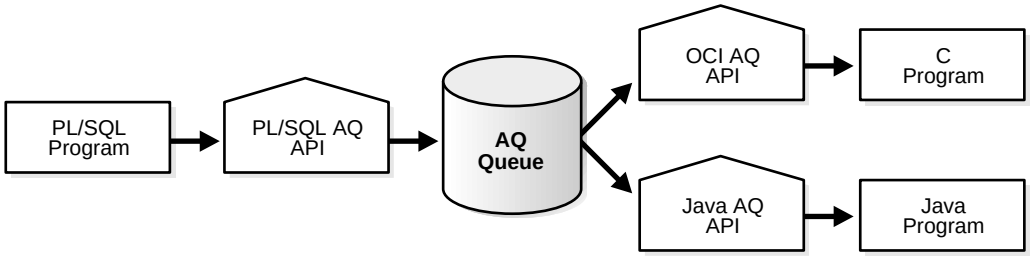
- JMS services are deployed on a computer that is does not have an Oracle 8.1.6 Enterprise Edition database installed.
- Non-durable subscribers subscribe to JMS Topics.
- You use another messaging technology, such as MQSeries, as the underlying message store on a particular platform.
- You do not require AQ extensions to JMS.
- An adapter toolkit is beneficial.

AQ API Compatibility

If you use either OMB, OJMS, or both in an integration solution, look carefully at these compatibility considerations:

- If, in a mixed programming environment (for instance, PL/SQL, C, and Java), you use AQ as an interchange between the languages, then using the native AQ APIs is probably a better alternative than using a JMS API.
- The compatibility of the native AQ APIs means that messages can be inserted using one native AQ API and removed using another without necessitating any transformation. For instance, you can enqueue a message to Queue1 using PL/SQL AQ API and dequeue the message from queue1 using the OCI AQ API for one recipient and the Java AQ API for another recipient, as shown below.

Figure 8–1 AQ API Compatibility



Unfortunately, the different JMS implementations are incompatible, both with each other and with the native APIs, because they use different attributes in the AQ repository to record the JMS message properties and payload. Consequently, messages created using one JMS API (for instance, the OMB JMS AQ Driver) must be transformed before they can be collected by another JMS API (for instance, the OJMS API). You must dequeue the message using one of the native AQ APIs, manipulate the message properties and JMS properties, and sometimes separate the JMS properties from the payload before enqueuing the message in the new format to a compatible queue.

Interoperability with Other Messaging Technologies

OMB has a number of drivers that enable the Oracle AQ, the TIBCo Rendezvous, and the IBM MQSeries to act as JMS message servers in a Java programming environment. You can use these message servers to create OMB-specific, JMS-format messages in AQ, MQSeries, and Rendezvous, and ordinary messages in MQSeries and Rendezvous.

By defining propagation between queues that use these different message formats and message servers, you can use OMB as a message translation service between the queues, transforming the message, for example, from a native MQSeries message to an OMB-specific, JMS-format AQ message.

Some restrictions apply to the types of messages supported. Read "Drivers" on page 8-3, *Oracle8i* Message Broker Administration Guide *2.0.1.0* for details.

In OMB, you can define client-side callouts in Java, C, and C++. These can be used to connect to Mercator Enterprise Broker.

You can connect directly from OMB to the Workflow, Java APIs, or, through PL/SQL calls, to the PL/SQL API only through predefined programming in Java.

MQSeries Example

We can use the same MQSeries example that we used in the AQ chapter to demonstrate some of the scenarios for integrating to OMB.

The example given below is used solely to illustrate the process and should not be taken as a recommendation of any one solution over any other.

Scenario

One of the applications included in a hub-and-spoke integration solution has a ready-made message-based interface that places messages in a set of MQSeries queues and that expects business events in the form of MQSeries messages. The application interface is written in C and creates and consumes messages using the native MQSeries API. The objective is to transport messages from MQSeries to AQ and from AQ to MQSeries.

Our objective is simply to get the messages in MQSeries into OMB and vice-versa.

The solution depends upon the types of messages we send or receive from MQSeries, the architectural principles, the language or languages used in the integration solution, the approach to modularization, the standards used, and so on.

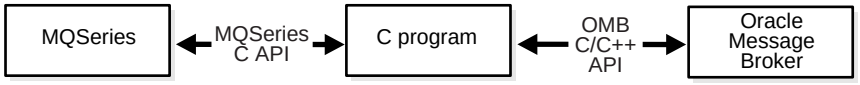
Figure 8–2 MQ queue-MQ driver-OMB

See Also:

- This solution is simple, but is subject to the constraints and limitations described in the Chapter 7, "Oracle Advanced Queuing and JMS".
- The same is true for communicating to Rendezvous. It is a simple solution. However, see the Chapter 7, "Oracle Advanced Queuing and JMS"

Programmatic

Figure 8–3 MQSeries-C program-OMB



Developing a predefined Java, C, or C++ program that communicates with OMB through the standard client-side OMB APIs is an alternative means of indirectly connecting OMB queues to other messaging technologies.

Workflow Example

In certain application integration scenarios, it may be desirable to have applications interact with other messaging technologies such as MQSeries and TIBCo Rendezvous because of a decoupling requirement, application platform, existing interface, or previous strategic technology choices. It is important to understand how these messaging technologies can interact effectively with the Oracle Integration Server products.

If you have an interface to the application that extracts messages to a messaging technology other than AQ, your first task is to identify your objectives and to establish the routes by which you can get from the messaging technology to these objectives. The most common mistake at this point is to assume that the most direct route from A to B is the fastest, the simplest, the easiest, or the most effective.

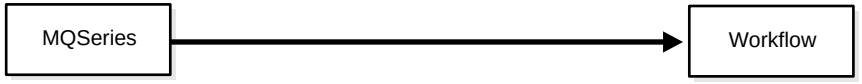
The process is most easily shown by example. The scenario that follows is used solely to illustrate the process and should not be taken as a recommendation of any one solution over the other.

Scenario

One of the applications included in a hub-and-spoke integration solution has a ready-made message-based interface that places messages in a set of MQSeries queues and that expects business events in the form of MQSeries messages. The application interface is written in C and creates and consumes messages using the native MQSeries API. One of the objectives is to get a message in MQSeries that represents a business event to instantiate an Oracle Workflow business process.

The solution depends upon the architectural principles, the language or languages used in the integration solution, the approach to modularization standards used, and so on.

Figure 8–4 MQSeries - Workflow



It is not possible to cover all the possible alternative scenarios and permutations of solution, but a few examples based on variations of the scenario illustrate the point.

Variant 1

You take a code-based adapter development approach, using the Java programming language, and have not adopted the JMS standard. The designer prefers a coarse-grained approach to modularization (that is, a smaller number of bigger program units).

There are no requirements to:

- Retain the message in an audit trail
- View unconsumed messages using SQL
- Receive the message from other non-MQSeries enabled publishers

In this variant, you write a simple, predefined Java program that gets the message through the MQSeries native Java API and transforms the message payload into the required parameter format before invoking the Java Workflow method that instantiates the business process.

Figure 8–5 Variant 1

Variant 2

You take a code-based adapter development approach using the Java programming language. You adopt the JMS standard for all message interfaces and the designer prefers a coarse-grained approach to modularization (that is, a smaller number of bigger program units).

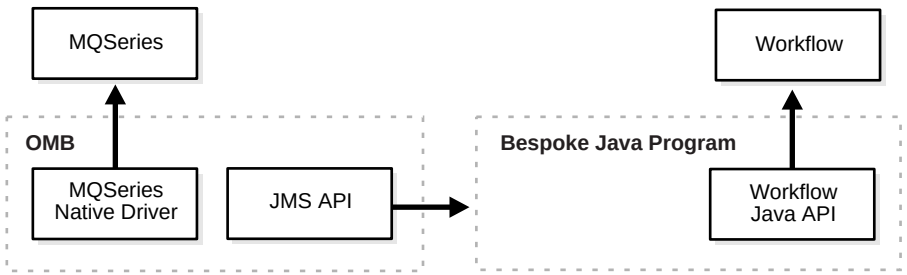
The MQSeries queue and the Workflow instance reside on different computers and the Oracle database is not installed on the application computer. You may need in the future to receive the same business event from publishers using TIBCo Rendezvous messages and AQ messages.

There are no requirements to:

- Retain the message in an audit trail
- View unconsumed messages using SQL

In this variant, you write a simple predefined Java program that gets the message through Oracle Message Broker and transforms the message payload into the required parameter format before invoking the Java Workflow method that instantiates the business process.

Figure 8–6 Variant 2



Variant 3

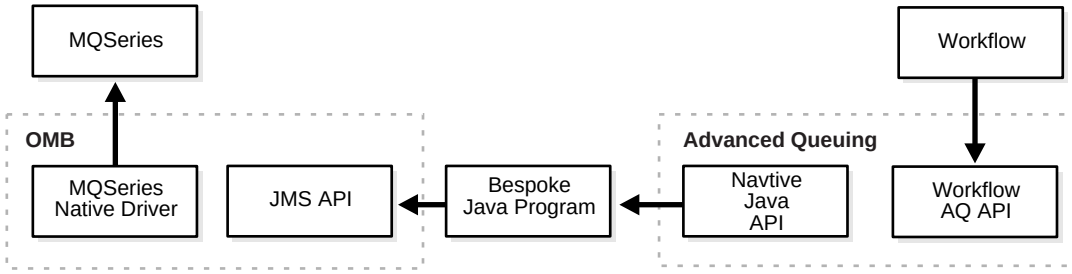
The variant is the same as for Variant 2, but this time you asynchronously inform the MQSeries-enabled application of a business event that has occurred during a step or activity in an Oracle Workflow business process.

There are no requirements to:

- Retain the message in an audit trail
- View unconsumed messages using SQL

In this variant, you develop an adapter that uses the Workflow AQ API to record the asynchronous message. You develop a simple, predefined Java program that gets the message through the native Java AQ API, sets the relevant JMS properties, and passes the message to Oracle Message Broker, which records the message in the MQSeries queue.

Figure 8–7 Variant 3



Variant 4

Take a graphical approach to developing adapters and connectors, thus minimizing the coding effort. No specific language is mandated and XML is the preferred format for external interfaces. The MQSeries queue and the Workflow instance reside on different computers and the Oracle database is not installed on the application system.

Messages must be retained as an audit trail. In the future, it may be necessary to receive the same business event from publishers using TIBCo Rendezvous messages and AQ messages.

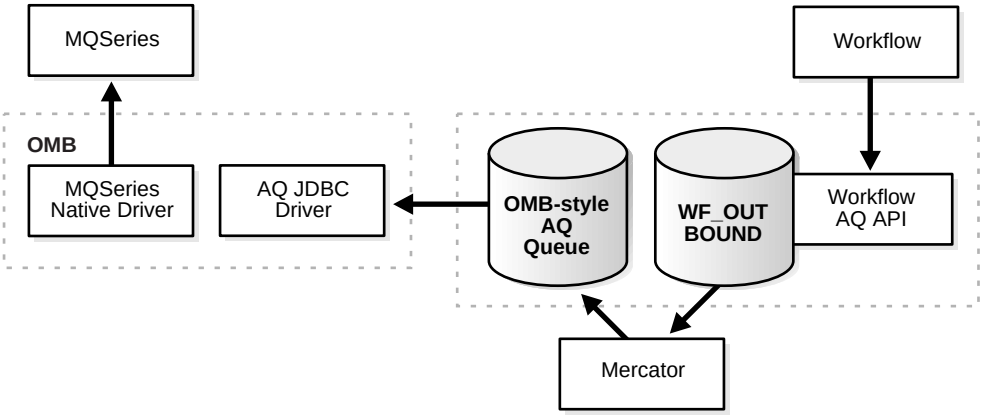
The business event is the same as for Variant 3; you are asynchronously informing the MQSeries-enabled application of a business event that has occurred during a step or activity in an Oracle Workflow business process.

This time, the coding effort is reduced but more components are involved. The Workflow step performs an asynchronous callout by writing to the `WF_OUTBOUND` queue.

Mercator subscribes to the `WF_OUTBOUND` queue, transforms the message into an OMB-style JMS format, and places it in an OMB-style AQ queue. OMB propagates the message between the AQ queue and the MQSeries queue.

The increased number of transformation steps may cause the end-to-end delivery time of an individual message to be slightly increased, but message throughput should not be adversely affected.

Figure 8–8 Variant 4



Variant 5

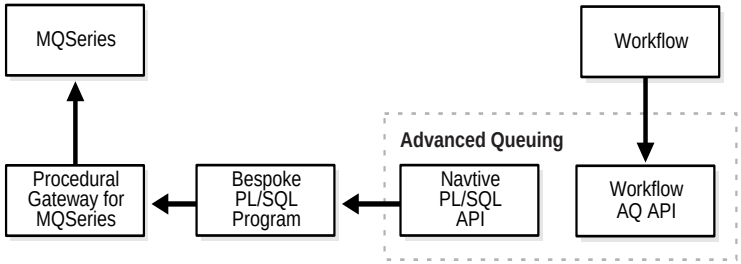
For this variant, you take a code-based adapter development approach, but with no specific language mandated. The applications are predominantly Oracle-based and the application using MQSeries is one of the few running on a computer without an Oracle database.

Messages must be retained as an audit trail. In the future, it may be necessary to receive the same business event from publishers using TIBCo Rendezvous messages and AQ messages.

The business event is the same as for Variants 3 and 4; you are asynchronously informing the MQSeries-enabled application of a business event that has occurred during a step or activity in an Oracle Workflow business process.

The MQSeries messages are simple and the expected message volumes are expected to be approximately 200 messages per minute.

Figure 8–9 Variant 5



The Workflow step performs an asynchronous callout by writing to the `WF_OUTBOUND` queue.

The predefined PL/SQL program subscribes to the `WF_OUTBOUND` queue, transforms the message data into the format required by the application, and calls the Procedural Gateway package that puts the message on MQSeries. You then use MQSeries to propagate the message to the application computer where the application can collect it.

Enabling Tools

This section introduces several tools including:

- Programming Languages
- Transformation Engines
- Message Transformers

Programming Languages

You can write predefined programs to connect any messaging technologies to any of the Oracle Integration Server products, provided both the messaging technology and the OIS product have an API for that language.

The amount of programming required depends on the extent of the differences in message format between the two products, the standard on which the API is based, and the extent to which the API supports the standard. Remember that, to remain simple, predefined programs are designed to connect only one topic to one service call or topic. You must also maintain the code.

Programs that facilitate generic communication between any queue in a messaging product and any aspect of the API of another product are time-consuming to develop, difficult to test thoroughly, tend not to be robust, and rarely scale linearly.

Effective programs are normally developed by the product vendors as a collaborative effort, consume fairly large R&D budgets, and use complex multithreaded programming to enable an acceptable level of scalability.

Transformation Engines

Transformation tools such as Mercator Enterprise Broker can transform messages between one messaging technology and another. The requirement to migrate messages between messaging technologies is rarely sufficient justification in itself to warrant the purchase and implementation of such specialized products. Often these products take up a lot of storage space and some are transactionally weak or offer no commit model when moving between technologies.

Message Transformers

A number of products available from Oracle and other vendors simplify the programming effort required to transform messages from one technology to another.

The three most significant message transformation tools are:

- **Procedural Gateway for MQSeries:** An Oracle product that enables PL/SQL and AQ connection to MQSeries
- **Oracle Message Broker:** An Oracle product that provides drivers that enable the Oracle AQ, TIBCo Rendezvous, and the IBM MQSeries to act as JMS message servers in a Java programming environment. Some restrictions apply. Read Oracle8*i* Message Broker Administration Guide 2.0.1.0 for more information.
- **TIB Adapter for Oracle:** This TIBCo product was developed in collaboration with Oracle Corporation and provides a code-free transformation solution between Oracle Advanced Queuing and TIB Rendezvous (TIBrv). The design enables messages to be converted from AQ to TIBrv, transported across the TIB bus, and converted back into AQ, without losing any of the AQ message properties. Messages are converted using the Oracle commit model.

9

Directory Services (LDAP)

Oracle Message Broker (OMB) determines message routing information for point-to-point messaging by looking up the destination address in a Lightweight Data Access Protocol (LDAP) directory. OMB accesses the LDAP directory through a standard Java Naming and Directory Interface (JNDI) driver.

By storing message routing information in an LDAP, OMB provides two important benefits:

- Centralizing the administration of message routing information
- Enabling dynamic modifications to message routing simply by changing entries in the LDAP directory

This chapter tells you all about directory services provided through the Oracle LDAP directory, Oracle Internet Directory (OID). It contains:

Java and Directory Service Integration

This section contains:

Directory Services - An Introduction

A directory service is a centralized, network-based repository that stores and provides access to information that must either be shared between applications or is highly distributed. Directory services play a vital role in developing intranet and Internet applications by helping you share information about users, systems, networks, services, and applications throughout the network.

Directory services have been used in a variety of forms for a number of years. For instance, e-mail systems include directories to store user information and to route messages between senders and receivers. Large corporations also use directories of various forms as repositories of employee information: centrally storing names, departments, managers, social security numbers, and other organizational information.

Although the evolution of directory services has been piecemeal in most companies, the value of directory services has increased sharply with the growth of Internet computing. Internet applications are by nature highly distributed and need an efficient mechanism to share information across a network.

The Problem

For instance, consider an expense-reporting application that migrates from a client-server application to an Internet application. A few human resource professionals enter the expense reports for an entire company to an Internet self-service application on which each employee also enters his expense reports. The volume of users accessing the self-service Internet application is far greater than the volume who accessed the client-server system. As a result, the cost of managing Internet users becomes a much greater part of the total cost of administering the application.

Similarly, modern Internet applications are developed as modular Enterprise JavaBean or CORBA components, each of which is accessed by a particular name that identifies its location and its public interfaces. Each time a component is moved from one node in the network to another, it must provide a new name that all of its clients must be able to find. With many different components distributed across an

intranet, the cost of administering these components becomes a significant part of the total cost of ownership of the system.

Finally, using an Internet architecture to develop applications simplifies how applications are deployed: they are no longer deployed on many client computers but are deployed on a few centralized, professionally managed servers. However, a number of new infrastructure components are required for Internet applications: ORBs, Web servers, Java Virtual Machines, databases, and application servers, to name just a few. The cost of administering these infrastructure components also must be managed.

The Solution

All Internet applications have a common problem: how to centralize information in a single repository from which it can be efficiently accessed and shared by users or applications. The solution is Directory Services. You can use directory services to store and share a variety of different kinds of information.

For instance, a directory service can administer Internet, intranet, or extranet users who access OLTP applications and databases. Users are represented by their security certificates and their access control privileges, which can be stored centrally in a directory. An administrator then changes the user's information and privileges in a single location rather than making such changes in each of the applications and databases that the user accesses. Additionally, the directory service serves as a central repository to manage distributed network resources such as Net8 and IIOP listeners, dispatchers, and connection managers in an intranet environment.

Further, you can also use a directory service to store the names of CORBA and Enterprise JavaBean application components in a similar manner to storing Net8 names in a `TNSNames` file.

Directory services are, therefore, now recognized as a critical component of the management infrastructure required to lower the total cost of ownership of Internet applications. Directory services do this in two ways:

- By centralizing the administration of distributed information in a single place
- By letting information stored in the directory be easily shared across users and applications within a corporate intranet and across extranets

Directory Services and LDAP, a Technical Overview

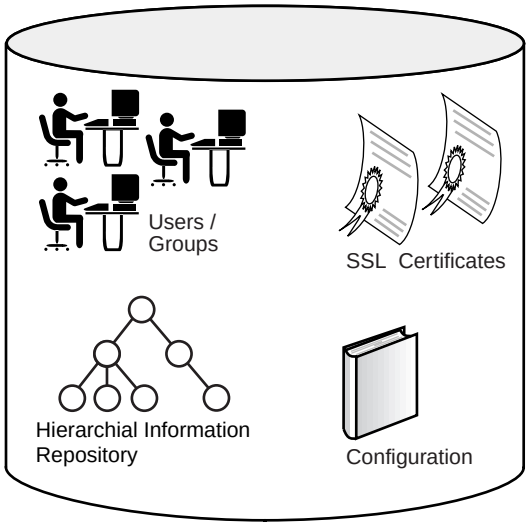
A directory service requires a robust, fast, and highly scalable data store for the information stored in it. As a result, these services are typically built on an enterprise database. A directory service is, therefore, essentially a specialized

database with a hierarchical schema that can be easily extended to store a variety of different kinds of information.

Most directory services are accessed through the Lightweight Directory Access Protocol (LDAP), which defines an open, vendor-neutral, industry standard network protocol and set of access methods to a directory. LDAP has become the standard access method for directory information, much as the Domain Name System (DNS) is used for IP address look-up on almost any system on an intranet and on the Internet. LDAP is currently supported in most network operating systems, groupware, and even shrink-wrapped Internet applications.

LDAP evolved from the X.500 OSI directory service standard. X.500 incorporated many useful ideas, but was too heavyweight and complex to be useful for Internet applications. LDAP was designed to provide 90% of the functionality of the full X.500 specification at 10% of the cost. It radically simplifies the format in which messages are transported across the wire, representing data elements as simple strings, and leaving out many of the little-used but redundant operations in X.500. The efficiency of the LDAP on the wire and its aptness for Internet applications have catapulted it into the forefront of Internet directory services.

Figure 9–1 LDAP Directory Service



LDAP defines four basic information models that fully describe its operations, what type of information can be stored in LDAP directories, and what can be done with the information. These four models are:

- **Information Model**: The LDAP Information Model defines what kinds of information can be stored in an LDAP Directory.
- **Naming Model**: The LDAP Naming Model defines how information in an LDAP directory can be organized and referenced.
- **Functional Model:** It defines what you can do with the information in an LDAP directory and how it can be accessed and updated.
- **Security Model:** It defines how you can secure information in an LDAP directory and what kinds of privileges users and applications require to access the directory.

Let us look at each of these in greater detail to understand how you use them in developing Internet applications.

LDAP Information Model

The LDAP Information Model is centered around the concept of an **entry**. Entries are created in the directory to hold information about some object or concept in the

real world (for example, a person, an organization, or a printer). Entries are composed of attributes that contain the information to be recorded about the object. Each attribute has a type and one or more values.

The type of an attribute has an associated syntax that defines what kind of information can be stored in the values of the attribute and how those values behave during searches and other directory operations. For example, the attribute `cn` (short for common name) has syntax called `caseIgnoreString` that implies three things:

- Attributes are sorted by lexicographic ordering.
- Case is ignored during comparisons.
- Values must be character strings.

Attribute types can also have various constraints associated with them, both limiting the number of values that can be stored in the attribute to one or many and limiting the size of those values. For example, an attribute to hold a person's identification number might be single-valued. An attribute to hold a photograph might be constrained to a size of no more than 10 KB to prevent unreasonable uses of storage space. The attributes required and permitted in an entry are controlled by content rules defined on a per-server basis, or by a special attribute in every entry called an `objectclass`. The values of this `objectclass` attribute identify the type of entry (person, organization, and so on) and determine which attributes are required and which are optional.

For example, the `objectclass` person requires the `sn` (for surname) and `cn` (for common name) attributes. This is the LDAP equivalent of the rules associated with a schema.

LDAP release 3 adds the ability to do away with schema enforcement altogether and add any attribute you choose to an entry. This is useful in environments where the overhead of adding new schema entries and making all the clients and servers aware of the new elements is prohibitive. To facilitate this, the LDAP release 3 protocol adds a special `objectclass` called `extensibleObject`. If the `objectclass` attribute of an entry contains the value `extensibleObject`, any other attribute is permitted in the entry regardless of the schema rules in place.

LDAP Naming Model

Though it is not a protocol requirement, LDAP directory entries are usually arranged in a hierarchical tree structure, for instance, following a geographical and organizational distribution. Entries are named according to their position in the hierarchy by a distinguished name (DN). Each component of the `DN` is called a

relative distinguished name (RDN) and is composed of one or more attributes from the entry. To anyone familiar with a UNIX or a Windows file system, this concept is analogous to pathnames and filenames: the RDN is analogous to the name of a file and the DN to the absolute path name to the file. As with a file system, sibling entries (entries with the same parent) must have different RDNs. However, LDAP has two important differences from a file system:

- **Sort Order:** Unlike filenames, LDAP components start with the least significant component (that component that names the entry itself) and proceed to the most significant (that component just below the root). In contrast, filenames usually start at the root and proceed down to a file or a directory. This difference is only syntactic and not semantic.
- **Name separators:** LDAP entries are named using a format called a distinguished name. A **distinguished name** is a sequence of RDN components, separated by commas or semi-colons. Note that in contrast to filenames, LDAP name components are separated by commas and not by forward or backward slashes. For instance, a filename has the structure

  ```
  /usr/local/ldap/include/ldap.h
  ```

  but an LDAP entry has the structure

  ```
  cn = Steve Harris, o=Software Industry, c=India
  ```

As shown, alias entries that point to other entries are permitted to circumvent the tree-like hierarchy. Further, note that a hierarchy is supported but is not required by LDAP. In many applications, the ability to organize and search information hierarchically is useful, however, in other cases, it may be inconvenient. The LDAP model can handle both cases because, in non-hierarchical case, you can use a one-level hierarchy as a flat namespace.

LDAP Functional Model

After you place information in an LDAP directory, LDAP defines nine different operations to authenticate clients so that they can access the directory, search the directory and retrieve information from it, and update information in the directory.

The search operation selects information from a defined area of an LDAP tree based on selection criteria. For each matching entry, a requested set of attributes (with or without values) can be returned. The searched entries can span a single entry, the immediate children of an entry, or the entire sub-tree of an entry. Alias entries can be followed automatically during a search and clients can specify the size and time limits of the search. LDAP release 3 directories also retrieve the results from a query

in the form of a scrollable result set. That is, search results can be retrieved a page at a time, and clients can scroll up and scroll down to view the results they require.

LDAP Summary

LDAP provides three important facilities that makes it an ideal way to access Directory Services.

Separable Naming Contexts: First, it provides a namespace that can be partitioned into a number of different naming contexts into which information in the directory can be organized. Each of these naming contexts can be hosted on a separate physical server node in order to provide clients with the perception of a single logical directory. This enables a directory service administrator to speed performance by placing frequently accessed information on a high performance server computer. For instance, in managing many different suppliers accessing a set of e-commerce applications, you may partition them into a set of namespaces based on the priority of the supplier: placing top priority suppliers in one `namespace`, next tier suppliers in a different `namespace`, and so on.

Additionally, since LDAP `namespaces` are physically separable, administrators can eliminate the risk of single points of failure by placing directory information in a mirrored CPU configuration.

Hierarchical Information: Secondly, LDAP-based directory services organize information in a hierarchical pattern. This makes a directory service a natural way to represent hierarchical information, such as inventory lists for supply chain applications and price lists and catalogs for e-commerce applications. In the inventory example, for instance, each product is represented as a single product entry in the directory with child entries that represent the inventory levels for each subcomponent of the product itself.

Information in the directory service can itself be queried dynamically to find the root node of a naming context. You can access the child entries by sequentially navigating the hierarchy of the directory.

Security Enforcement: Finally, an LDAP directory service provides a number of stringent security mechanisms to protect the information stored in it. For instance, directory users must first authenticate themselves to the directory using either a username and password or an SSL / X.509 release 3 certificate (through a bind operation). Once the user has been authenticated, the information he can access is still further constrained by using an access control list.

Access control lists specify the users who can read information within the directory. Since entries in the directory are stored hierarchically, you can easily apply access

controls to both individual entries and entry attributes, as well as to subtrees of entries. Further, access control applied to the product entry is automatically inherited by all the children. As a result, directory services enable you to control access to information in the directory in a detailed manner. This makes directory services ideal mechanisms to share information.

Oracle Corporation recommends that you use a directory service as a centralized administration facility to manage any information or facility that is distributed across the intranet, extranet, or the Internet. This includes users, application components, security, and business processes or workflow. We also recommend the use of a directory to manage information that is shared between applications within a corporate intranet and across an extranet.

Oracle Internet Directory

Oracle Internet Directory is an application running on the Oracle8*i* database server that implements directory services. It is LDAP release 3 compliant.

Because it is based on the Oracle8*i* server, Oracle Internet Directory leverages the standard features of the database to offer unparalleled scalability in terms of size, performance, reliability, availability, and management by using:

- Database backup and recovery features
- Multithreaded model of the Multi-Threaded Server
- Connection pooling feature
- Advanced symmetric replications to enable the implementation of geographically disturbed LDAP servers

Access to the LDAP server can be provided securely through the support of OID for Secure Socket Layers (SSL) release 3.

> **See Also:** *Oracle Internet Directory Administrator's Guide* for more information on Oracle Internet Directory and its capabilities

10

Workflow

This chapter introduces Oracle Workflow and contains these sections:

- Overview
- Key Workflow Components
- Key Workflow Features

Overview

Oracle Workflow was originally developed as a traditional workflow tool to manage complex user-focused workflows within Oracle Applications. As Oracle Corporation recognized Oracle Workflow as a powerful tool for managing complex user-based business processes, we made subsequent releases available as a stand-alone product.

The latest releases include functionality that enhances the ability of the product to provide fully automated business processing by enabling synchronous and asynchronous callouts through Advanced Queuing.

Oracle Workflow is made up of these components:

- Oracle Workflow Builder
- Workflow Engine
- Workflow Definitions Loader
- Workflow Monitor

Key Workflow Components

Oracle Workflow Builder

Oracle Workflow Builder lets you create, view, or modify a business process with simple drag-and-drop operations. Using the Workflow Builder, you can create and modify all workflow objects, including activities, item types, and messages.

At any time, you can add, remove, or change workflow activities, or set up new prerequisite relationships among activities. You can easily work with a summary-level model of your workflow, expanding activities to greater levels of detail within the workflow as needed. And, you can operate Oracle Workflow Builder from a desktop PC or from a disconnected laptop PC.

Workflow Engine

The Workflow Engine embedded in the Oracle8*i* database server monitors workflow states and coordinates the routing of activities for a process. A change in workflow state, such as the completion of workflow activities, is signaled to the engine through a PL/SQL API or a Java API. Based on flexibly defined workflow rules, the Engine determines the activities that are eligible to run, and then runs

them. The Workflow Engine supports sophisticated workflow rules, including looping, branching, parallel flows, and subflows.

Workflow Definitions Loader

The Workflow Definitions Loader is a utility program that moves workflow definitions between database and corresponding flat file representations. You can use it to move workflow definitions from a development to a production database or to apply upgrades to existing definitions. In addition to being a standalone server program, the Workflow Definitions Loader is integrated into Oracle Workflow Builder, enabling you to open and save workflow definitions in both a database and a file.

Workflow Monitor

Oracle Workflow queue APIs can be called by an application program or by a workflow function in the runtime phase to handle workflow Advanced Queue processing. In Oracle Workflow, an outbound and an inbound queue are established. A package of data on the queue is referred to as an **event** or a **message**. A message in this context is different from the messages associated with notification activities.

Events are enqueued in the outbound queue for agents to consume and process. These agents may be any application that is external to the database. Similarly an agent may enqueue a message to the inbound queue for the Workflow Engine to consume and process. The outbound and inbound queues facilitate the integration of external activities into the workflow processes.

Note: Background engines use a separate or deferred queue.

Key Workflow Features

This section contains:

- Complete Programmatic Extensibility
- Electronic Notifications
- Electronic Mail Integration
- Internet-Enabled Workflow

- Monitoring and Administration
- Business Event System
- Uses
- AQ API
- PL/SQL Callout Functionality
- Instantiating Business Process Instances Using PL/SQL and Java
- Interoperability

Complete Programmatic Extensibility

Oracle Workflow lets you include your own PL/SQL procedures or external functions as activities in your workflow. Without modifying your application code, you can have your own program run whenever the Workflow Engine detects that the prerequisites of the program are satisfied.

Electronic Notifications

Oracle Workflow lets you include users in your workflow to handle activities that cannot be automated, such as approvals for requisitions or sales orders. Electronic notifications are routed to a role, which can be an individual user or a group of users. Any user associated with that role can act on the notification.

Each notification includes a message that contains all the information a user needs to make a decision. The information may be embedded in the message body or attached as a separate document. Oracle Workflow interprets each notification activity response to decide how to move on to the next workflow activity.

Electronic Mail Integration

Electronic mail (e-mail) users can receive notifications of outstanding work items and can respond to those notifications using their choice of e-mail applications. An e-mail notification can include an attachment that provides another means of responding to the notification.

Internet-Enabled Workflow

Any user with access to a standard Web browser can be included in a workflow. Web users can access a Notification Web Page to see their outstanding work items, then navigate to additional pages to see more details or to respond.

Monitoring and Administration

Workflow administrators and users view the progress of a work item in a workflow process by connecting to the Workflow Monitor using a standard Web browser that supports Java. The Workflow Monitor displays an annotated view of the process diagram for a particular instance of a workflow process, so that users can view a graphical depiction of their work item status. The Workflow Monitor also displays a separate status summary for the work item, the process, and for each activity in the process.

Business Event System

The Business Event System is an application service that uses the Oracle Advanced Queuing (AQ) infrastructure to communicate business events between systems. The Business Event System consists of the Event Manager, which lets you register subscriptions on events that are significant to your systems, and Event Activities, which let you model business events within workflow processes.

When an event occurs, the subscribing code is executed in the same transaction as the code that raised the event. Subscription processing can include sending event information to other queues or systems, executing custom code on the event information, and sending event information to a workflow process.

Workflow Monitor

The Workflow Monitor represents graphically the current state of a particular instance of a workflow as it is executed. This is particularly useful for manually tracking long-running message flows as they move through the integration solution. The Monitor is a Java-based applet that you can access through a browser.

Uses

You can use Oracle Workflow Standalone Edition release 2.5.2 and later as an integration flow manager to coordinate business event communication between applications by instantiating, controlling, and synchronizing the flow of instances of business events.

Use the Workflow Builder graphical interface to define the order and conditions under which the business process steps execute. Implement automated business process steps using the PL/SQL API and the AQ API to request instantiations of an integration flow by the Workflow Engine. The Engine requests execution of externally defined process steps (for instance, to perform payload transformations).

Oracle Workflow strongly supports manual intervention by users. You can define flows to detect abnormal, suspicious, or problematic instances of business events and to route these events to e-mail, Lotus Notes, MS Exchange, UNIX SendMail, Worklist User Interface, and other products for user approval, correction, or rejection.

> **Caution:** Oracle Corporation strongly recommends that the instance of Oracle Workflow employed within the integration solution be dedicated only to integration flow management.

This restrictive approach prevents you from trying to enable the workflow of applications through the integration solution. This ensures that you can upgrade the integration solution independently of the applications that it integrates.

Asynchronous communication between Oracle Workflow and Advanced Queuing is possible both through the Workflow AQ API and through the PL/SQL callout functionality.

AQ API

The Workflow AQ API provided with Standalone Workflow release 2.5.2 is simple and includes three queues, all of which:

- Have the same Object Type payload (`wf_payload_t`)
- Do not permit message retention
- Employ single consumer queues

The object type for the payload has a number of attributes required by Workflow that enable it to associate the queued message with a particular flow. The object type also has a text attribute that you can format to contain parameters in a `name/value` pairs format proprietary to Oracle Workflow.

Because the queues are single-consumer, the routing cannot use the AQ agent to determine the recipients of outbound messages. Use a combination of `itemtype` and correlation to identify the business process or a range of business processes.

The three queues have these purposes:

- `WF_OUTBOUND`: The Workflow Engine places a message in this queue when the flow reaches a step defined as an external callout. External processes collect the

message from the queue using a PL/SQL package called WF_Queue.DequeueOutbound or WF_Queue.DequeueEventDetail.

- WF_INBOUND: After placing the message in the outbound queue, the Engine waits for any reply from the inbound queue. When the message is processed by the external process, you place results in the inbound queue using the PL/SQL package WF_Queue.EnqueueInbound. The Workflow Engine then associates the results with the appropriate flow.
- WF_DEFERRED: This queue is used by the foreground Workflow Engine to defer processing of flows to a background Workflow Engine.

All Oracle Workflow queue APIs are defined in a PL/SQL package called WF_QUEUE. You must execute these queue APIs from the same Oracle Workflow account because the APIs are account dependent.

Note: In using these APIs, we assume that you have prior knowledge of Oracle8*i* Advanced Queues concepts and terminology.

See Also: *Oracle8i Application Developer's Guide - Advanced Queuing* for more information on Advanced Queue APIs

Queue APIs

- EnqueueInbound
- DequeueOutbound
- DequeueEventDetail
- PurgeEvent
- PurgeItemtype
- ProcessInboundQueue
- GetMessageHandle
- Deferred_queue
- Inbound_queue
- Outbound_queue

Developer APIs for the Inbound Queue

The following APIs enable developers to write to the inbound queue by creating messages in the internal stack rather than using `WF_QUEUE.EnqueueInbound()`. The internal stack is purely a storage area and you must eventually write each message that you create on the stack to the inbound queue.

For efficient performance, you should periodically write to the inbound queue to prevent the stack from growing too large.

- `ClearMsgStack`
- `CreateMsg`
- `WriteMsg`
- `SetMsgAttr`
- `SetMsgResult`

Payload Structure

All Oracle Workflow queues use the data type `system.wf_payload_t` to define the payload for any given message. The payload contains all the information that is required about the event. A description of `system.wf_payload_t` follows:

Table 10–1

Column Name	Type	Description
ITEMTYPE	Varchar2(8)	The item type of the event
ITEMKEY	Varchar2(240)	The item key of the event
ACTID	Number	The function activity instance ID
FUNCTION_NAME	Varchar2(200)	The name of the function to execute
PARAM_LIST	Varchar2(4000)	A list of `value_name=value` pairs. In the inbound scenario, the pairs are passed as item attributes and item attribute values. In the outbound scenario, the pairs are passed as all the attributes and attribute values of the function (activity attributes).
RESULT	Varchar2(30)	An optional activity completion result. Possible values are determined by the Result Type of the function activity or can be an engine-standard result.

See Also: *Oracle8i Application Developer's Guide - Fundamentals* and *Standard API for an Oracle Workflow 2.5.2*

PL/SQL Callout Functionality

It is possible to define a process step as a PL/SQL callout. When the step is reached, the PL/SQL procedure named in the step is called and parameters specified in the step are passed to the called procedure. When the procedure completes, it passes back a result to indicate success or failure.

You can use the PL/SQL procedure to create a message in an AQ queue or to collect messages from an AQ queue. In doing so, consider:

- Procedures must conform to the standards defined in *Workflow Builder Help.* (cross-reference to "Standard API for PL/SQL Procedures Called by Function Activities"
- The transactional implications. Workflow uses save points to separate flow steps from each other. You can create or collect AQ messages as part of the `savepoints` transaction. However, the creation or collection is not visible until the commit is issued.
- To dequeue a JMS message from an AQ queue using the PL/SQL AQ API and translating the JMS properties and payload in PL/SQL, you must understand how OMB/OJMS formats its messages in AQ. (Oracle8*i* Message Broker Administration Guide 2.0.1.0)

Instantiating Business Process Instances Using PL/SQL and Java

A PL/SQL package called `WF_ENGINE` contains two procedures that you can use to create a business process (`wf_engine.CreateProcess`) and to start that process (`wf_engine.StartProcess`).

To ensure that the process is executed asynchronously, the cost associated with the first step in the flow must be greater than the threshold of the Workflow Engine.

The Oracle Workflow Java interface offers the `WF_ENGINE` and `WF_NOTIFICATION` packages as Java methods that can be called by any Java program to communicate with Oracle Workflow. The Java methods directly reference the `WF_ENGINE` and `WF_NOTIFICATION` PL/SQL package procedures and views and communicate with the Oracle Workflow database through JDBC.

Interoperability

Previously, we have used an MQSeries integration scenario to demonstrate the types of solution that you can use to integrate an application that uses non-Oracle messaging technology to the Integration Server products.

The scenario can be examined again here to good effect.

If an interface to an application extracts messages to a messaging technology other than AQ, you must decide whether to integrate the messaging technology to Workflow directly using a programming language or indirectly through specialized software.

The alternative methods for integrating to Workflow are:

- **Programmatic 1:** If both the message queuing product and Workflow have an API for a particular language, you can write programs that collect the message from the messaging technology and pass the parameters to Workflow. Or you can write programs that instantiate a new workflow OR callout from Workflow to create messages in the messaging technology.
- **Programmatic 2:** Choose two languages that can communicate with each other, one of which has an API to the messaging technology and the other of which has an API to Workflow.
- **Mixed Programmatic and Software:** Use software such as Procedural Gateways for MQSeries that provide an API in a language to which Workflow has an API.

Part III

Reference

Part III describes several interoperability solutions between OIS and various third-party products.

This part contains the following chapters:

- Appendix A, "Mercator Enterprise Broker and OIS"
- Appendix B, "Front-End and Back-End Integration"
- Appendix C, "Autonomous and Pointer Payloads"
- Appendix D, "Business Events and System Events"

A

Mercator Enterprise Broker and OIS

Mercator Software is a vendor of a set of enterprise application integration tools including the Enterprise Broker. The Enterprise Broker provides a rich set of data transformation services that you can use with the Oracle Integration Server, particularly for non-XML data transformations in a legacy systems environment. This appendix provides an overview of how to use Mercator Enterprise Broker with Oracle Integration Server. This appendix contains:

Introduction

The Enterprise Broker performs application-to-application integration and is particularly useful for transforming data and messages. The software is designed around a transformation flow in which an object is transformed from one form or state to another by one or more transformation steps.

Each transformation step has these properties:

- A trigger or triggering event
- An input object or objects
- Transformation rules
- An output object or objects

The Enterprise Broker itself is made up of the Engine and four graphically based editors:

- System Editor
- Type Tree Editor
- Database Editor
- Mapping Editor

System Editor

The System Editor defines transformation flows. It links together a number of transformation steps to model the multistep transformation of a message from one form or state to another.

You can use the System Editor to define a trigger or a triggering system event that precipitates a transformation. Typical triggers include:

- Arrival of a message in a queue
- Creation of a file in a directory
- Insertion of a row in a table
- Changing of a last-modified timestamp of a file

Type Tree Editor

Mercator Software products use a construct called a Type Tree to internally represent the data structures of external objects. A Type Tree takes one of three forms:

- **Predefined:** Mercator provides a set of predefined Type Trees that represent common standard message formats (such as EDI and HL7) and the public interface objects of common ERP packages (such as SAP / R3).
- **Imported:** You can generate Type Trees to represent database objects in most relational databases using the Database Editor.
- **Definable:** You can define Type Trees manually by using the Type Tree Editor.

Type Trees must be defined for each source that provides input to the transformation, and for each target that receives output from the transformation.

Database Editor

Use the Database Editor to generate Type Trees for given database objects in a relational database. The Editor must connect to the database as a database user. The Editor thus assumes the privileges, permissions, and view of the objects defined for that user.

In Oracle, Type Trees can be generated for:

- **Tables**: Columns become fields in the Tree Type.
- **Procedures**: `IN`, `OUT` and `IN/OUT` parameters become fields.
- **Queues**: If the payload of the queue is defined as an object type, then each attribute becomes a field in the Type Tree. If the payload is of `RAW` type, then the Type Tree is represented by a single binary string (`BLOB`).

Mapping Editor

Use the Mapping Editor to configure individual transformation steps by defining input cards, output cards, and maps.

A transformation step is analogous to a phase in a manufacturing process:

- **Input cards** define suppliers and the specification of raw materials.
- **Maps** define the manufacturing process.
- **Output cards** define the product specification and the delivery addresses of customers.

Input Cards

I**nput cards** define:

- Rules for collection, such as the type of connection to the source, transactional behavior, number of rows per fetch, and retry rules
- Format for the incoming messages by specification of all or part of a Type Tree

You can define multiple input cards in a single step, and multiple iterations of an input card are possible. For example, you define an input card for `order` and an input card for `order lines`. For each `ORDER`, there are multiple iterations of `order lines`.

Output Cards

O**utput cards** define:

- Rules for delivery, such as the type of connection to the destination, transactional behavior, number of rows per push, and retry rules
- Data format for the outgoing messages by specification of all or part of a Type Tree

You can define multiple iterations of multiple output cards within a step.

Maps

M**aps** are the mechanisms that tie the input and output cards together. They define the rules for transforming data from one form or state to another. Maps defined to perform a particular function on a defined subset of data are called **functional maps** and can nest within other maps.

The basic objective of the map is to transform the data from the input cards to the output cards. Maps achieve this in a number of ways:

- Dragging and dropping a field from the input card to another field on the output card
- Defining a static value for a field on the output card
- Defining the field in the output card as null
- Setting the field on the output card to a system value, such as a date/time stamp
- Providing input card fields to procedural logic, functions, or SQL lookups and then applying the result to the output card

Enterprise Broker Engine

After you configure the transformation processes using the editors, use the Engine to execute the transformation steps. These steps can be instantiated from the command line, from the System Editor, or from within programs written in C or Java (through the Mercator C and Java APIs).

The Engine provides logging and tracing facilities that capture information about each transformation step. This helps you debug errors, but it incurs a performance overhead.

The Engine has a number of performance-enhancing features that improve its scalability:

- It treats large messages as a data stream. This enables the Engine to start processing a message without waiting for the whole message to arrive.
- It reuses maps and database connections.
- It switches off logging and tracing.

Uses of Enterprise Broker in the Oracle Integration Server

Oracle Corporation recommends that you use the Mercator Enterprise Broker as the transformation engine for particularly complex transformations. It is especially useful when you require an "XML to anything" or an "anything to XML" transformation.

The AQ adapter implementation eases transformation between AQ queues in the same database if the input and output cards use the same connection string. This implementation enables the Oracle commit model to deliver the dequeue-transform-enqueue process as a single transaction, thereby guaranteeing that messages are retained.

Defining the data transformation as a distinct step in the process is consistent with the component-based approach we recommend.

Hints and Tips

- You can use Mercator Enterprise Broker to transform messages between AQ and MQSeries. However, you must track transactions to ensure that messages are not lost.

- The AQ adapter supports CLOBs and regular Oracle data types, but you must use care in mapping to BLOBs/RAW AQ queues because you must determine the length of the binary string before constructing the message.

Note: Release 2.1 of the Enterprise Broker includes no functionality to assist you with interpretation of JMS messages. You must understand the interrelationship of the OJMS and OMB AQ drivers before you attempt this, and must define the Type Trees for the JMS messages with care.

B

Front-End and Back-End Integration

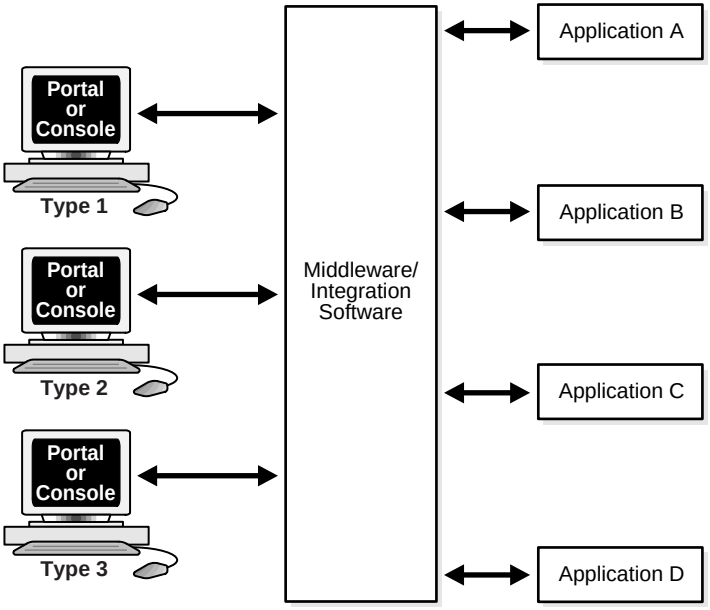
The requirements for front-office and back-office integration are fundamentally different and, as a result, they require fundamentally different integration approaches. This appendix contains:

- Front-End Integration
- Back-End Integration

Front-End Integration

In this implementation, middleware or a similar integration solution intercepts interactions between applications. The middleware acts as a middle tier application server: transparently marshaling, managing, and directing interapplication requests and responses. Typically, the requesting application waits while the middleware requests services of another application.

Figure B–1 Front-End Integration



Front-end integration solutions generally display some, or all, of these characteristics:

- Interaction between the integration layer and the applications is:
  - Synchronous, or simulates synchronous processing
  - Two-way communication based on a service request model
- Transactions processed are of short duration.

- The application (user) is not aware of using the services of multiple disparate applications.
- Middleware behaves like a midtier application server: marshaling, managing, and directing requests and responses and managing transactional and recovery issues.

Front-end integration has both advantages and disadvantages.

Advantages

- You can develop a number of presentation layers for different user, customer, and supply channels.
- All presentation layers share the same services, thus ensuring consistent interactions regardless of channel.
- Each presentation layer requires only one signon to access all required functionality.

Drawbacks

Applications must meet these criteria to participate in the interactions:

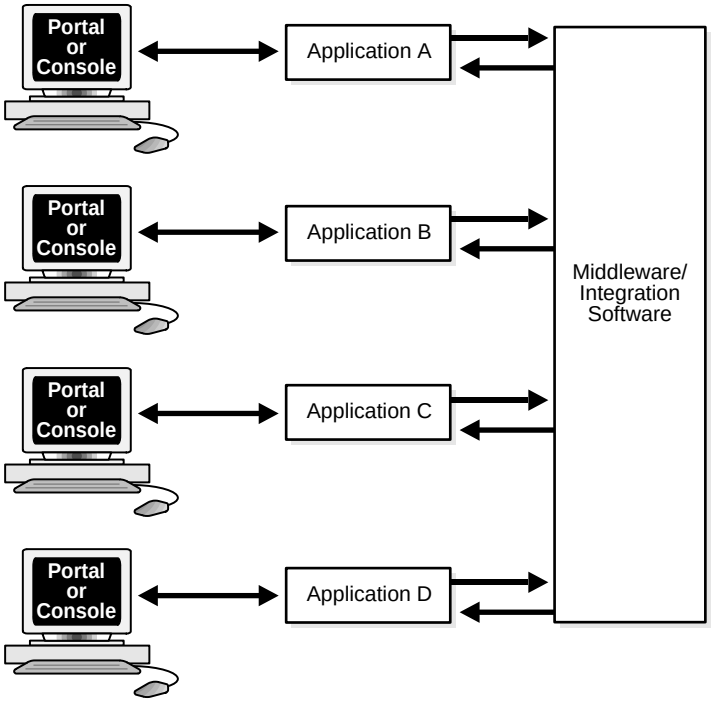
- All functionality must be offered through a service-based API.
- All business and data logic must be separate from the presentation layer.
- The transaction / commit model must act as a resource manager in the chosen multiphase commit processing.
- Few legacy and currently available ERP / CRM packages meet these criteria, so you probably must develop a custom application.
- The presentation layer of packaged applications is unlikely to be usable. This may lead to difficulties in providing support, lengthy implementation time, and so on.
- It is difficult to manage a development and testing environment because of the need for a complex architecture, the lack of suitable strong tools, and the dependence on program-oriented delivery.

Back-End Integration

In this implementation, users interact with only one application at a time in a manner determined by their user roles.The application notifies other applications as necessary of the significant aspects of the user interaction.

Application-to-application interactions can be based on further sequential notifications.

Figure B–2 Back-end integration



Back-end integration solutions display some or all of these characteristics:

- Interaction between the integration layer and the applications is:
  - Predominantly asynchronous
  - One-way communication
- Service-request and event-notification models are both used.
- Business process definitions closely mirror the business, and consequently transactions take a long time.
- The user interacts with a single application or with a small number of applications.

- The integration solution acts transparently as a sophisticated distribution and synchronizing agent.

Advantages

- The presentation layer interface to the applications need not employ the same architecture as the applications.
- You can use the prebuilt presentation layer delivered with packaged applications.
- The business and data logic in an application need not be separate from the delivered presentation layer.
- It is easy to manage the development and testing environments because asynchronous breakpoints provide natural testing points.
- Graphical tools are provided to ease development.

Drawbacks

- Each application has a different look and feel and a different signon, which makes applications appear to be only loosely integrated.
- The user interface is narrowly constrained, thus restricting users who act in multiple and diverse roles.

C

Autonomous and Pointer Payloads

The amount of information in the payload of the message is an important aspect of message design. In this appendix, we consider two alternative strategies for handing payload size, as well as a compromise implementation.

If you count the compromise implementation as a strategy, you can choose from three payload strategies:

- Pointer Payload
- Autonomous Payload
- Hybrid Payload

Pointer Payload

Payloads based on this strategy contain only sufficient information to enable other applications to request further information about the event from the publisher of the message. Use this style of payload if it is important to limit the message size because shipping large volumes of data is impractical.

Some characteristics that suggest that this payload style might be appropriate include:

- The payload does not need to be recorded immediately because a copy exists in some other persistent, secure form.
- Information can be sent to the consumer on an as-needed basis.
- A persistent copy of the data is not required after the consumer has processed it.
- The information is not required for routing, tracking, or business intelligence, or is available elsewhere.

Example 1: Video Film

In video-on-demand, there is little point in recording all the data in memory or to disk because the data is a pure, binary, digital, data stream. Little is gained by interpreting the data before it reaches its final destination.

Similarly, if there is a strong possibility that only the initial part of the data stream is actually required by the consumer, there is no need to deliver all the data.

Example 2: Changes to a Name and Address Database

Name and address changes are normally recorded in operational systems during the day when network traffic is high and disk I-O is intense.

A notification to a non-operational system, such as a marketing database, that a change has occurred enables that database to mark affected data as obsolete in real time. At a later time, when network and disk usage is lower, data can be updated with the details of the change.

Autonomous Payload

The payload strategy includes sufficient information that the message can be processed without reference to the originator of the message. Autonomous payloads reduce the interdependence of applications. This strategy is essential if you need to deploy publish-subscribe routing.

Including all the pertinent information relating to a business event facilitates:

- Content-based routing
- A complete audit trail
- Business intelligence use
- Message tracking

In an asynchronous, one-way messaging environment, if the payload is relatively small (less than 2 MB) and the message is processed in near real-time, using the autonomous payload strategy is your obvious choice.

Example 1: Share Trading

Banks responsible for managing stock portfolios for customers must ensure that details of sale or purchase requests are captured in an audit trail. The information about the trade, while relatively small, must be placed on the market within seconds and must be routed to different markets based on information contained in the payload.

Example 2: Stock Control

A warehouse stock control application subscribes to an order application during the day and uses the captured events to determine reorder levels during the night. The messages are relatively small. The stock control application must operate independently of the order application, which is operational only during normal business hours.

Hybrid Payload

This strategy is similar to pointer payload in that the payload must contain sufficient information to enable other applications to request further information about the event from the publisher of the message. The message must also include information sufficient for the subscriber to determine whether or not the event requires processing.

A hybrid payload is useful if the message is large, and if subscribers filter the messages after receiving them. Parts of the message payload enable integration functionality. For instance, they indicate whether or not content-based routing is required.

Example: Marketing

A e-mail broadcast distribution must contain enough information (sale price, sale dates) to interest a customer in downloading a product pack containing video clips, digital pictures, and full details of the product being offered.

If the take-up rate on the mailing is expected to be low, sending the full pack with the initial broadcast is inefficient. However, sending customers only a pointer to a Web site is insufficient to pique their interest in the product.

D

Business Events and System Events

Business Events

A business event is a definable occurrence in a business scenario. It can be a common high-level occurrence, such as a customer placing an order. Alternatively, it can be a more specialized event, such as a customer exceeding a credit limit while placing an order. Some events can be triggered by changes in values, such as the share price for a particular stock exceeding a given value.

Business events define significant happenings that different parts of the business must register and act upon by using different applications. Oracle Integration Server integrates all applications that register and act upon a business event.

This appendix contains:

System Events

A system event is a point in the execution of a program at which an identifiable computing task takes place. Examples of system events include: writing to a file, inserting data into a database table, calling a subroutine, and instantiating a program thread.

A program that implements a particular business process usually includes many system events, and more than one of these system events can provide a suitable trigger to capture a particular business event.

Example: Raising an Order

Many integration applications contain the business event, `Raise an Order`. Several applications must subscribe to such an event because they must know that an order has been placed. A program in another application that captures orders contains program steps to perform these system events:

- Get Accounting Date
- Get Customer Address
- Calculate Order Total
- Insert into Order Header

Each of these system events occurs once per order. Any one of them can provide a suitable point at which to trigger the '`Raise an Order`' business event.

Distinctions between Business and System Events

There are three important distinctions between a business event and a system event:

- The business event is a logical occurrence in a business scenario; the system event is the mechanism you use to recognize that the business event has occurred.
- The business event is not influenced by the manner in which it is implemented in an application; the system event is a part of that implementation.
- Messages that represent the business event must be constructed to provide information about the business event, not information about the system event that triggers it.

To emphasize business events in the integration solution, use a message-based strategy. To emphasize system events, use a strategy that implements data replication using messaging technologies.

Example: Emphasizing System Events

To integrate order processing, identify a business event called `PlaceanOrder`. If you have a system event named `CallCreateOrderHeade`r, you can instantiate message creation immediately prior to or immediately after the call.

The message created contains all the information about the order, including order lines, customer details if necessary, the order total, and some or all of the information in the order header, which is created as part of the `CreateOrderHeader` subprogram.

The message is then published to the middleware as an instance of the `Place an Order` subject or topic. The middleware takes responsibility for routing the messages to those subscribing to the `PlaceanOrder` business event.

Example: Emphasizing Business Events

You identify a set of business events that relate to customer interactions, including one called `RegistrationofaNew Customer`.

Identify a system event that always occurs when you register a new customer. The system event is the insertion of a row into the `customer` table of the customer relationship management application.

The system event can raise the business event by creating a database trigger that starts when a row is inserted into the `customer` table. The database trigger calls a PL/SQL procedure or Java function that constructs a message to represent the registration of the new customer, by gathering information from other relational tables such as the `address` table and by calling other functions such as a credit rating calculation.

Index

A

B

C

D

E

F

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W