IBM WebSphere InterChange Server

Technical Introduction to IBM WebSphere InterChange Server

Version 4.3.0

IBM WebSphere InterChange Server

Technical Introduction to IBM WebSphere InterChange Server

Version 4.3.0

30 September 2004

This edition of this document applies to *IBM WebSphere InterChange Server (5724-178)* version *4.3.0*, and to all subsequent releases and modifications until otherwise indicated in new editions.

To send us your comments about this documentation, email doc-comments@us.ibm.com. We look forward to hearing from you.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this manual

IBM(R) WebSphere(R) InterChange Server and its associated toolset are used with IBM WebSphere Business Integration Adapters to provide business process integration and connectivity among leading e-business technologies and enterprise applications.

This document introduces the infrastructure and other major elements of a system that implements InterChange Server as the integration broker for a WebSphere business integration system.

Audience

This document is for IBM consultants and customers.

Prerequisites for this document

This document is introductory, and is a prerequisite to using other documents in the InterChange Server documentation set.

Related documents

The complete set of documentation available with this product describes the features and components common to all WebSphere InterChange Server installations, and includes reference material on specific components.

You can install the documentation from the following sites:

- For InterChange Server documentation:
  `http://www.ibm.com/websphere/integration/wicserver/infocenter`
- For collaboration documentation:
  `http://www.ibm.com/websphere/integration/wbicollaborations/infocenter`
- For WebSphere Business Integration Adapters documentation:
  `http://www.ibm.com/websphere/integration/wbiadapters/infocenter`

These sites contain simple directions for downloading, installing, and viewing the documentation.

Typographic conventions

This document uses the following conventions:

`courier font`	Indicates a literal value, such as a command name, filename, information that you type, or information that the system prints on the screen.
bold	Indicates a new term the first time that it appears.
italic, *`italic`*	Indicates a variable name or a cross-reference.
blue text	Blue text, which is visible only when you view the manual online, indicates a cross-reference hyperlink. Click any blue text to jump to the object of the reference.
{ }	In a syntax line, curly braces surround a set of options from which you must choose one and only one.

`[ ]`	In a syntax line, square brackets surround an optional parameter.
`...`	In a syntax line, ellipses indicate a repetition of the previous parameter. For example, `option[,...]` means that you can enter multiple, comma-separated options.
`< >`	In a naming convention, angle brackets surround individual elements of a name to distinguish them from each other, as in `<server_name><connector_name>tmp.log`.
`/, \`	In this document, backslashes (\) are used as the convention for directory paths. For UNIX installations, substitute slashes (/) for backslashes. All IBM WebSphere product pathnames in this documentation are relative to the directory where the IBM WebSphere product is installed on your system.
%text% and *$text*	Text within percent (%) signs indicates the value of the Windows `text` system variable or user variable. The equivalent notation in a UNIX environment is *$text*, indicating the value of the *text* UNIX environment variable.
ProductDir	Represents the directory where the product is installed. For the IBM WebSphere InterChange Server environment, the default product directory is "IBM\WebSphereICS". For the IBM WebSphere Business Integration Adapters environment, the default product directory is "WebSphereAdapters".

New in this release

This chapter describes the new features of IBM WebSphere business integration system that are covered in this document.

New in WebSphere InterChange Server v4.3

The IBM WebSphere InterChange Server 4.3 release provides the following features:

- End-to end privacy, which secures messages as they flow starting at the node of origin, through any intermediate nodes, to the destination node. End-to-end privacy provides maceration and asymmetric, dynamic security to the InterChange Server.
- Role-based access control, which authenticates users who want to access the InterChange Server, then limits the access the user has to individual components based on what role the user has been assigned.
- Dynamic service calls, which enables a collaboration to make a service call to a destination that has not been pre-defined and explicitly bound during collaboration development.
- IBM Tivoli License Manager (ITLM) support
- Language-specific behavior support

New in WebSphere InterChange Server v4.2.2

The IBM WebSphere InterChange Server 4.2.2 release provides the following feature:

- InterChange Server (ICS) and other ICS components now use the IBM Java Object Request Broker (ORB) instead of the third-party VisiBroker ORB.

New in WebSphere InterChange Server v4.2.1

For the IBM WebSphere InterChange Server 4.2.1 release, this guide provides high-level descriptions of new System Manager features.

Chapter 1. Overview of IBM WebSphere InterChange Server

This document provides a high-level introduction to the use of IBM WebSphere InterChange Server as the integration broker in an IBM WebSphere business integration system. InterChange Server in a WebSphere business integration system provides a distributed infrastructure for solving cross-application problems, including the capability to

- Move business information among diverse sources to perform business exchanges across the Internet, and
- Process and route business information among disparate applications in the enterprise environment

This chapter provides an overview of the architecture, components, and processing flow of an integration system that uses InterChange Server as the integration broker. It contains the following sections:

Note: Throughout this manual, illustrations are examples only, used to show structure and concepts. They do not necessarily document specific actual components.

InterChange Server and the WebSphere business integration system

The IBM WebSphere InterChange Server and its associated toolset are used with IBM(R) WebSphere(R) Business Integration Adapters in a **WebSphere business integration system**. At the highest level, a WebSphere business integration system consists of an integration broker and a set of adapters that allow heterogeneous business applications to exchange data through the coordinated transfer of information, in the form of business objects.

InterChange Server is not the only integration broker that can be used as the core of a WebSphere business integration system. Others include WebSphere MQ Integrator Broker, WebSphere Business Integration Message Broker, and WebSphere Application Server. All these integration brokers can make use of WebSphere Business Integration Adapters, which utilize connectors for application connectivity and business objects as the containers for business data exchanged between

applications. However, there are some differences between integration systems using the two different brokers, and choosing the right broker for your needs is an important high-level decision.

This guide focuses specifically on concepts for the integration system that is implemented with InterChange Server as the integration broker. For information about implementing an integration system that uses one of the WebSphere Message Brokers (WebSphere MQ Integrator, WebSphere MQ Integrator Broker, or WebSphere Business Integration Message Broker) as the integration broker, see the *Implementing Adapters with WebSphere Message Brokers* guide. For information about implementing an integration system that uses WebSphere Application Server as the integration broker, see *Implementing Adapters with WebSphere Application Server*. Both these implementation guides are part of the WebSphere Business Integration Adapters documentation set.

InterChange Server model

A WebSphere business integration system implemented with InterChange Server uses modular components and application-independent business logic. The InterChange Server approach uses a distributed, hub-and-spoke infrastructure. It enables you to execute business processes—e-business order fulfillment, returns processing, or inventory management, for example—that are distributed across the Internet, across applications on local networks, or both. The system is distributed and flexible, with reusable components and customization features that make it possible to meet site-specific and application-specific needs.

An implementation of InterChange Server utilizes the connectors provided by the WebSphere Business Integration Adapters product, together with several types of modular and customizable components, including collaborations, business objects, maps, and data handlers. The chapters that follow in this guide described how these components are used in a WebSphere business integration system with InterChange Server as the broker.

The IBM WebSphere InterChange Server provides ennoblement for use with IBM Tivoli License Manager. IBM WebSphere InterChange Server provides support for license management functions as well as inventory functions for server components and Toolset. (See *Access Development Guide for Enterprise Java Beans* and *Access Development Guide J2EE Connector Architecture* for more information.

WebSphere Business InterChange Server Toolset

Available with InterChange Server is the WebSphere Business Integration Toolset that enables you to

- Develop and customize components
- Organize your components into projects for deployment in successive stages of development and production
- Monitor a deployed InterChange Server integration system

These are described in Chapter 2, “Tools for use with InterChange Server,” on page 39.

Collaborations, business objects, and connectivity

Note: In this document, "application" refers to an enterprise software product that is being integrated; it does *not* refer to a component of the IBM WebSphere InterChange Server product.

The IBM WebSphere InterChange Server system uses a central infrastructure (InterChange Server) and modular components in a hub-and-spoke design, as follows:

- Business-process logic resides in InterChange Server **collaborations** at the hub.

  InterChange Server collaborations are software modules that contain logic that describes a distributed business process. There are different collaborations for different fundamental business processes—for example, a ContactManager collaboration, or an InventoryMovement collaboration. Collaborations coordinate the functionality of business processes for disparate applications and enable data exchange between them. Collaborations are the hub; through them, data is exchanged in the form of business objects.

- Data is exchanged between the hub and the spokes in the form of **business objects**.

  Business objects are the messages used by the IBM WebSphere InterChange Server system for exchanging data. **Data handlers** are used to transform serial application data into business objects, and **maps** are used between a business object that is structured for the data model of a specific application and a business object that is generically structured for use by collaborations at the hub.

- Application and technology **connectors**, which are available in the IBM WebSphere Integration Adapters product, supply connectivity to applications (or to web servers or other programmatic entities) at the spokes.

  Connectors can be configured to interact either within a network, or across the Internet and beyond firewalls.

  Each connector consists of two parts—the **connector controller** and the **connector agent.** The connector controller interacts directly with WebSphere InterChange Server collaboration objects and resides on a server that has implemented the IBM WebSphere InterChange Server system (the hub in a hub-and-spoke relationship). The connector agent interacts directly with an application, and can reside with that application on any server. A remote agent technology can be used to implement communication between a connector controller at a hub site and an agent that resides at another site across the Internet.

  Some connectors are designed to interact with specific applications. Application connectors—for example, the Clarify connector— are intermediaries between collaborations and applications. These connectors transform data from the application into business objects that can be manipulated by the collaborations, and transform business objects from the collaborations into data that can be received by the specific application.

  Other connectors are designed for interactions that conform to specific technology standards. (For example, the XML connector can be used for sending data from InterChange Server collaborations to a web server, even if that web server resides beyond a firewall on a network that is not running the connector agent or other IBM WebSphere software.)

- The **Server Access Interface** makes it possible for remote spoke sites that do not implement WebSphere InterChange Server to use **access clients**, which make calls over the Internet to a hub site that does have InterChange Server.

The Server Access Interface is part of InterChange Server. It is a CORBA-compliant API that accepts synchronous data transfers from either internally networked or external sources. The data is then transformed into business objects that can be manipulated by a collaboration. The Server Access Interface makes it possible to receive calls from external entities—for example, from web browsers at remote customer sites—that do not come through connector agents, but instead come through web servlets into the Server Access Interface.

The Server Access Interface and the connectors both make use of data handlers. In InterChange Server environment, new data handlers can be created from a modular group of base classes called the **Data Handler Framework**. The InterChange Server solution also includes a **Protocol Handler Framework**. These frameworks make it easier to customize solutions and add connectivity for additional data formats and protocols in the future.

Sample implementation solutions

A typical InterChange Server solution includes one or more collaborations and a set of business objects that represent business information relevant to an enterprise. The collaborations and business objects are used with connectors, with the Server Access Interface, or with both.

A connector can interact with one or more collaborations, thereby performing various business processes. And each collaboration can interact with any number of connectors, thereby involving any number of applications.

For example, to automatically update an enterprise resource planning (ERP) application when customer information changes in a customer interaction management (CIM) application, the InterChange Server solution might consist of the CustomerSync collaboration, connectors for the CIM application and the ERP application, and definitions of business objects that represent customer information. Figure 1 illustrates that solution.

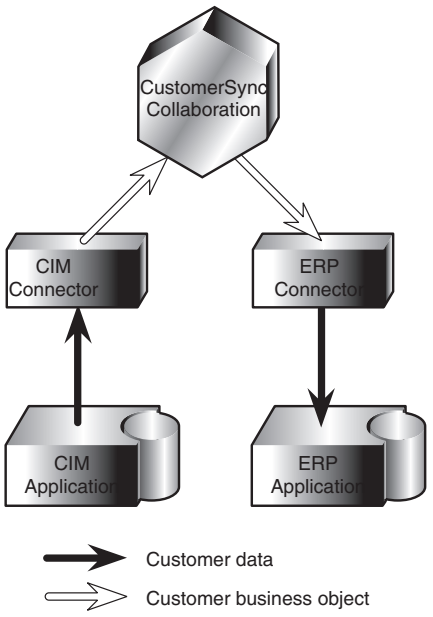

Figure 1. CIM-to-ERP Customer Data solution

The solution in Figure 1-1 could be implemented either locally on a network, or across the Internet.

In Figure 2, at a site that has not implemented either the IBM WebSphere InterChange Server system or a connector, a customer representative might wish to use a web browser to obtain the status of a purchase order over the Internet from an ERP application (SAP in the example) that resides at a site using the WebSphere InterChange Server system. To enable this, the InterChange Server solution uses the Server Access Interface, together with a collaboration (fictional in this example) for purchase-order business logic, an SAP connector, and definitions of business objects that represent purchase-order status information.

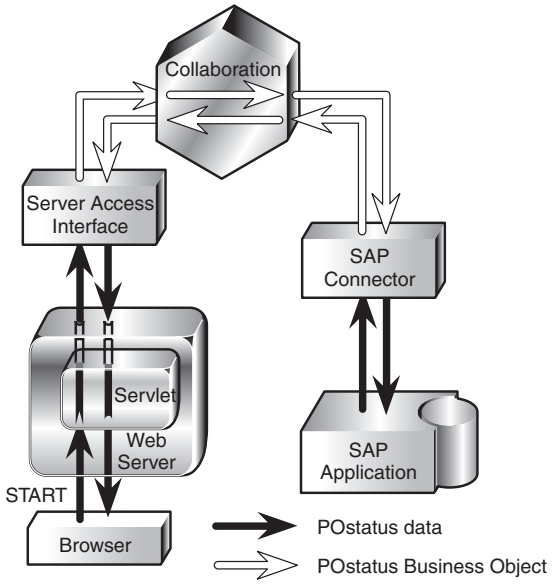

Figure 2. Execution of a call through the Server Access Interface

Multiple server deployment

A single application can interact with multiple InterChange Servers, in a hubs-and-spokes environment. Multiple InterChange Servers can be deployed to work together. Connector agents can be configured to partition and route events from an application to different servers. This makes it possible to balance an application's processing load across multiple InterChange Servers. In addition, InterChange Servers can be configured to interact with each other through connectors and the Server Access Interface.

Connectivity over the Internet

Connectors can enable the exchange of data across the Internet in different ways. These approaches include:

- Application connector agents implemented remotely

  This approach—referred to as remote agent technology—uses JMS, with either SSL in native WebSphere MQ or HTTP/HTTPS in WebSphere MQ internet pass-thru (MQIPT), to implement a connector across the Internet. A single site that has implemented the IBM WebSphere InterChange Server system acts as the hub and performs exchanges across the Internet with spoke sites that have implemented a remote connector agent.
- Technology connectors used with the Server Access Interface

In this approach, the Server Access Interface is used to pass synchronous calls into the IBM WebSphere InterChange Server; connectors that use Internet technology standards are used to send data from the IBM WebSphere InterChange Server:

– Calls from external browsers (or other sources over the Internet) can be received by a web servlet that sends the calls through the Server Access Interface to trigger business processes in collaborations.
– On behalf of collaborations, connectors for specific Internet technology standards (such as XML) engage in request/response interactions with external destinations that understand the specific data format.

The Server Access Interface resides at a hub site on InterChange Server (ICS). When the Server Access Interface receives a call, it sends the data to a handler for that specific data format (such as the XML data handler). The data handler transforms the data into a generic business object. The Server Access Interface then sends the business object to a collaboration. The collaboration performs its processes on the business object and responds, and that response is transformed back into the specific data format that was used at the beginning of the process.

To accept calls from external processes and send the calls as business objects to a collaboration, the Server Access Interface requires the following:

– A web server that uses a servlet or other process to send data to the Server Access Interface. The data must be in a MIME type for which data handlers have been configured in InterChange Server. The servlet uses the Server Access Interface to call a data handler that converts the data into a business object format, and then uses the Server Access Interface to send the business object as a call into a collaboration.
– A collaboration that has been configured to handle requests that are received as calls through the Server Access Interface.

An example of this approach is illustrated in Figure 3.

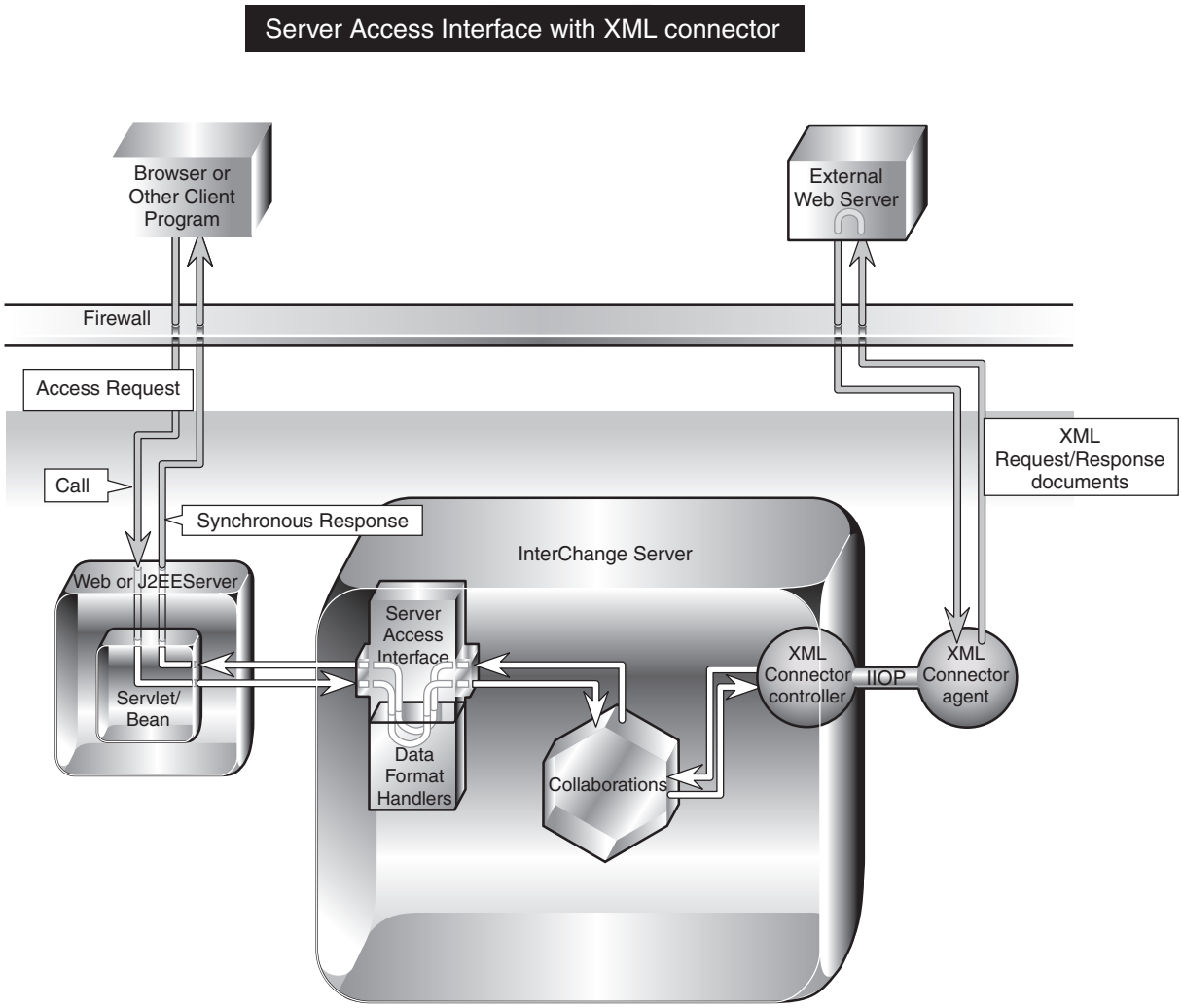

Figure 3. Internet data exchange using Server Access Interface

- Technology connectors designed for Internet exchange

  Some connectors can enable both publish-and-subscribe and request/response interactions that exchange data over the Internet, without using a remote agent configuration. For example, the TPI connector enables data exchange in XML, EDI, or binary format with remote trading partners over the Internet. Similarly, the Email connector exchanges data over the Internet using the SMTP protocol.

  An example of this approach is illustrated in Figure 4.

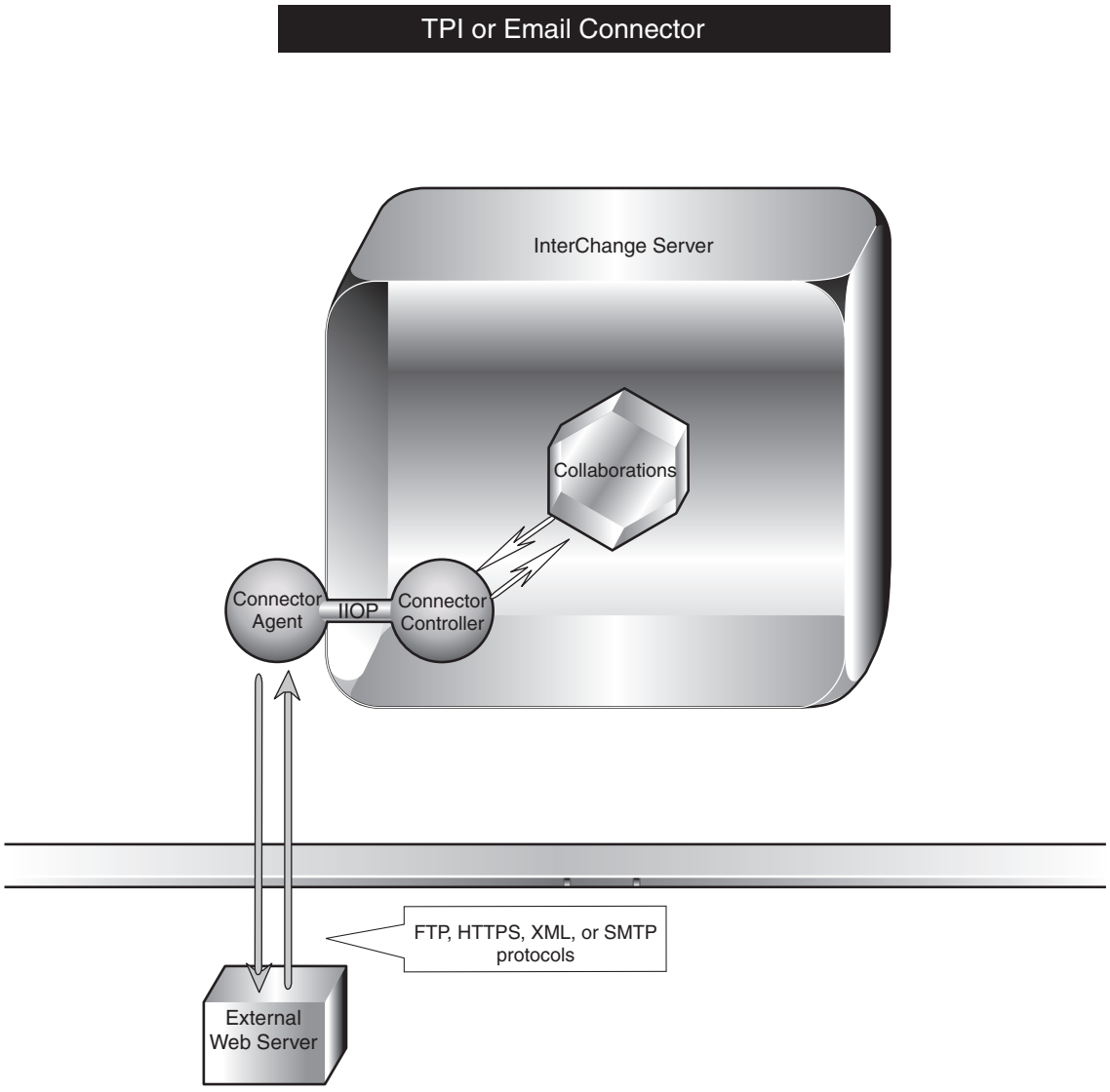

Figure 4. Internet data exchange using TPI or e-Mail Connector

Data flow in an InterChange Server implementation

In an InterChange Server implementation, a data flow—the movement and processing of data from one application or entity to another— can include all of the following:

- Asynchronous or synchronous exchanges between disparate applications on a local network.

- Asynchronous or synchronous exchanges between disparate applications across the Internet.
- Asynchronous or synchronous exchanges between applications on a local network and external web servers.
- Comprehensive business processes that include all of the above.

A data flow can be initiated by either of two types of interactions: publish-and-subscribe interactions or service call interactions. Both types of interaction supply triggers that start the execution of a collaboration's business processes. The collaboration then uses a third type of interaction—request/response —to complete the exchange of data with the intended destination.

Publish-and-subscribe interactions

Connectors and collaborations use a **publish-and-subscribe** interaction to move information about application events into the IBM WebSphere InterChange Server for processing.

In the publish-and-subscribe interaction, a collaboration begins its business process when it is triggered to do so by receipt of a business object for a particular type of **triggering event**—for example, Employee.Create—that represents an application operation. The name of the business object (*Employee*) indicates a type of business entity. The verb (*Create*) indicates an operation that occurred on that entity. Therefore, the Employee.Create event reports the creation of an employee entity.

The publish-and-subscribe interaction enables a triggering event to reach a collaboration as follows:

- A collaboration **subscribes** to the event that can trigger its execution. A collaboration subscribes to an event by requesting it, and then waiting for it. A collaboration that subscribes to Employee.Create starts executing when the business object for the Employee.Create event arrives.
- An event occurs in an application, and the event is detected by the application connector's **event notification** mechanism. The connector supplies the event to one or more collaborations by publishing it—making it available as a business object.

Depending on the connector, an event can be published to a collaboration either asynchronously or synchronously. In addition, if the long-lived business process feature of the collaboration is enabled, a collaboration can maintain the event in a waiting state, in anticipation of incoming events satisfying pre-defined matching criteria.

Access requests

A collaboration can be designed to be triggered by direct calls that are sent by an **access client**, received by the Server Access Interface, and sent to the collaboration as business objects. In an InterChange Server implementation, calls sent to collaborations through the Server Access Interface are referred to as **access requests**. Access requests can originate from external sources or from sources that are configured within InterChange Server implementation.

Access request interactions are useful when synchronous communication is important—as when, for example, a customer representative uses a web browser to request inventory status information over the Internet.

Request/response interactions

A collaboration begins processing data when it is triggered to do by receipt of a triggering business object. The triggering business object can be the result of either an access request or an event notification. Once a collaboration has been triggered, it can make requests of connectors to which it has been bound, and receive responses.

The collaboration makes its requests—referred to as **service call requests**—in the form of generic business objects. The connectors transform the generic business objects into data entities that are understood by the specific application or data format for which the connector is designed.

The responses that the collaboration receives from the connectors—referred to as **service call responses**—can be business objects containing business data (in the case of retrieve requests) or status reports (successful or unsuccessful).

Figure 5 shows a simplified view of a business process that is initiated through a publish-and-subscribe interaction and completed through request/response interactions. (In this simplified view, connector controllers and connector agents are shown as a single unit, and no distinction is made between connectors for applications residing on a local network and connectors for applications residing across an Internet firewall.) The example shows a distributed business process that automatically generates an invoice when a customer service representative finishes working on a case. In this example, a hypothetical service billing collaboration uses connectors to exchange business data with three different applications.

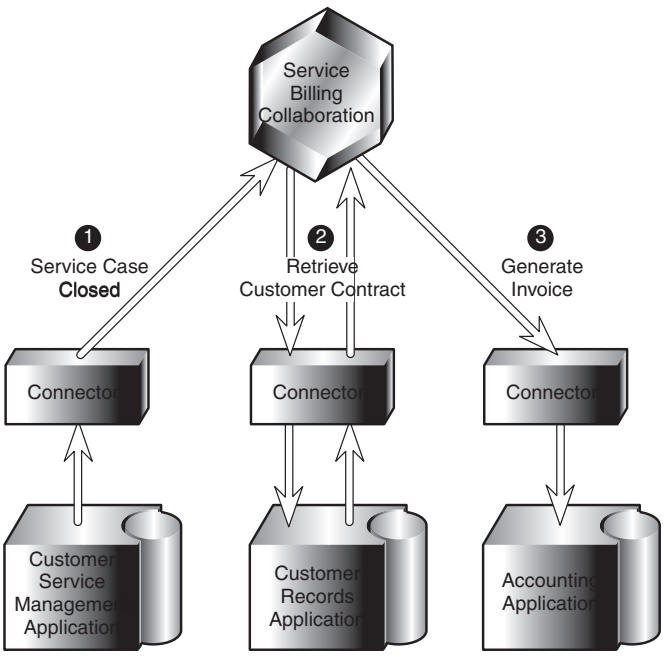

Figure 5. A publish-and-subscribe, request/response interaction

Figure 5 illustrates the following sequence in the business process:

1. A customer service representative completes work on a case. The connector detects the case closure as an event in the customer service management application and retrieves the relevant case data. The connector then publishes the event, making it available to a collaboration that has subscribed to it. These actions comprise a publish-and-subscribe interaction.

2. In order to compute the invoice amount, the collaboration needs the terms of the customer contract. The collaboration sends a service call request to retrieve the necessary data from the connector for the customer records application. The connector responds to the request. These actions comprise a request/response interaction between the collaboration and the customer records application and connector.
3. Using both the case information and the customer contract, the collaboration produces the information needed to generate an invoice. It sends the invoice creation request to the connector for the accounting application, which forwards the request to the application itself, and responds to the collaboration with a notification of success or failure. These actions comprise a request/response interaction between the collaboration and the accounting application and connector.

A site can tune the closeness with which collaborations and connectors are coupled. The collaboration might, for example, execute on a 24-hour basis, sending requests to a connector that communicates with its application only between midnight and 2:00 A.M. The collaboration could be designed and configured to send the requests without requiring a response and simply process the responses when they come.

Alternatively, a collaboration that has been enabled for long-lived business processes can save the flow context of a request, and send the request with a timeout value, specifying the period of time in which a response can cause the saved processing flow to resume.

Sample data flows

Figure 6 shows a high-level view of the IBM WebSphere InterChange Server system components used in typical access request, publish-and-subscribe, and request/response interactions to move data among applications (or other entities, such as web servers and browser).

The data moves in a flow through InterChange Server at the hub, and is exchanged with applications on the local network, applications configured with connector agents beyond an Internet firewall, and external entities such as web servers and browsers.

The flow can be initiated by a access request or by an event—or by any type of external or internal process. Once the flow has been initiated, the collaboration uses the request/response interaction to complete the business process with either local or remote applications or other entities.

In this diagram, one connector is shown publishing an application event to collaborations, and another connector is shown engaging in a request/response interaction between a collaboration and a web server destination. Note that this illustrates only one possible configuration; most connectors can be configured to both publish events and respond to requests from collaborations.

Not shown in this diagram is the possible configuration in which a connector interacts with a locally configured technology server—such as a TPI server or Email server—to exchange data over the Internet.

In addition, note that a connector agent and its corresponding application need not reside locally. Using remote agent technology, connector agents at remote URL locations can both publish events and respond to collaboration requests.

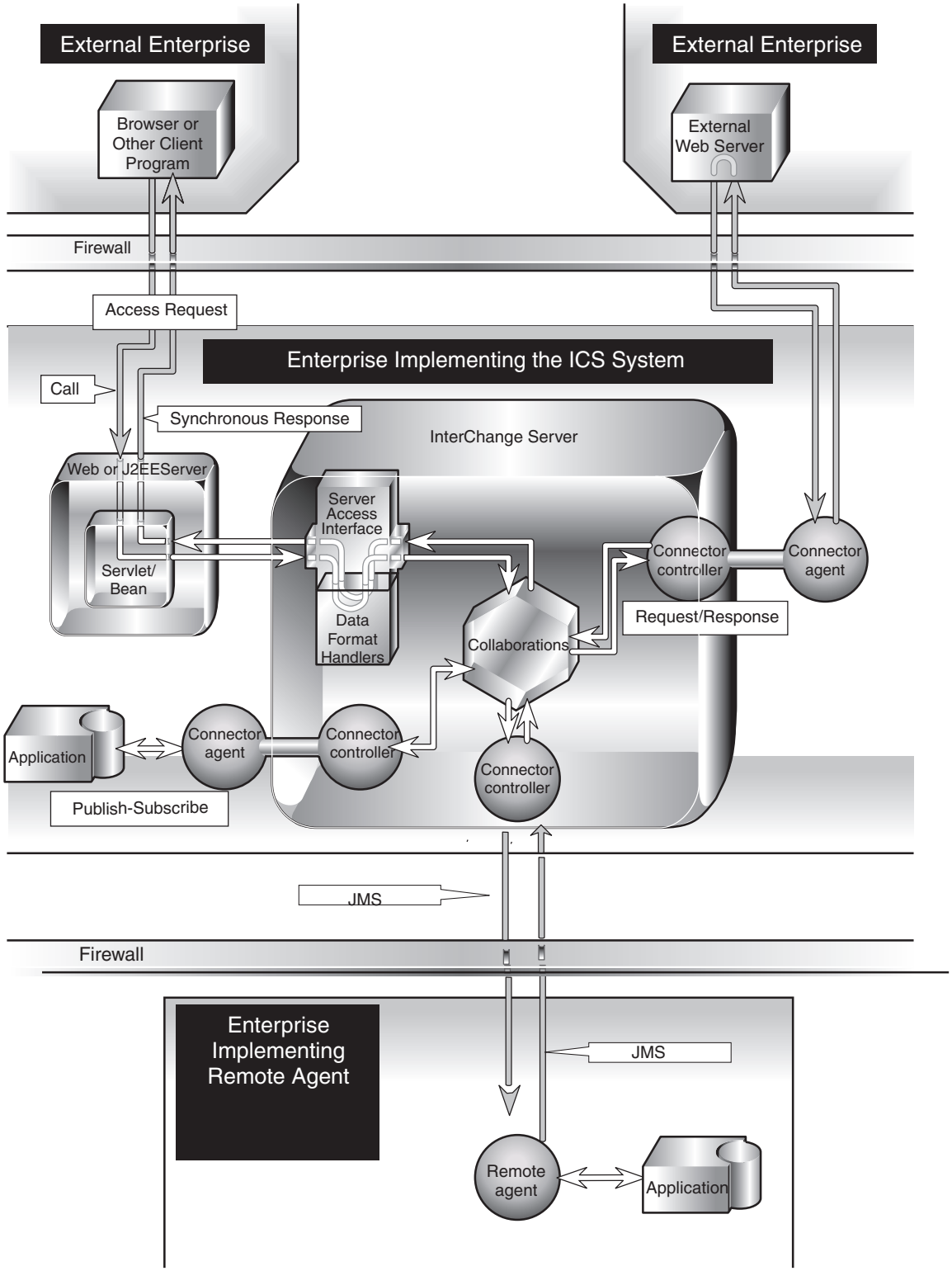

Figure 6. Business data flow

Figure 7 shows some types of data that are exchanged along some of the possible data paths.

The numbers indicate the specific sequence of just one possible data path—in this example, from an external web browser into the collaborations by way of the Server Access Interface, and then, after the collaboration's business processing, to an external web server.

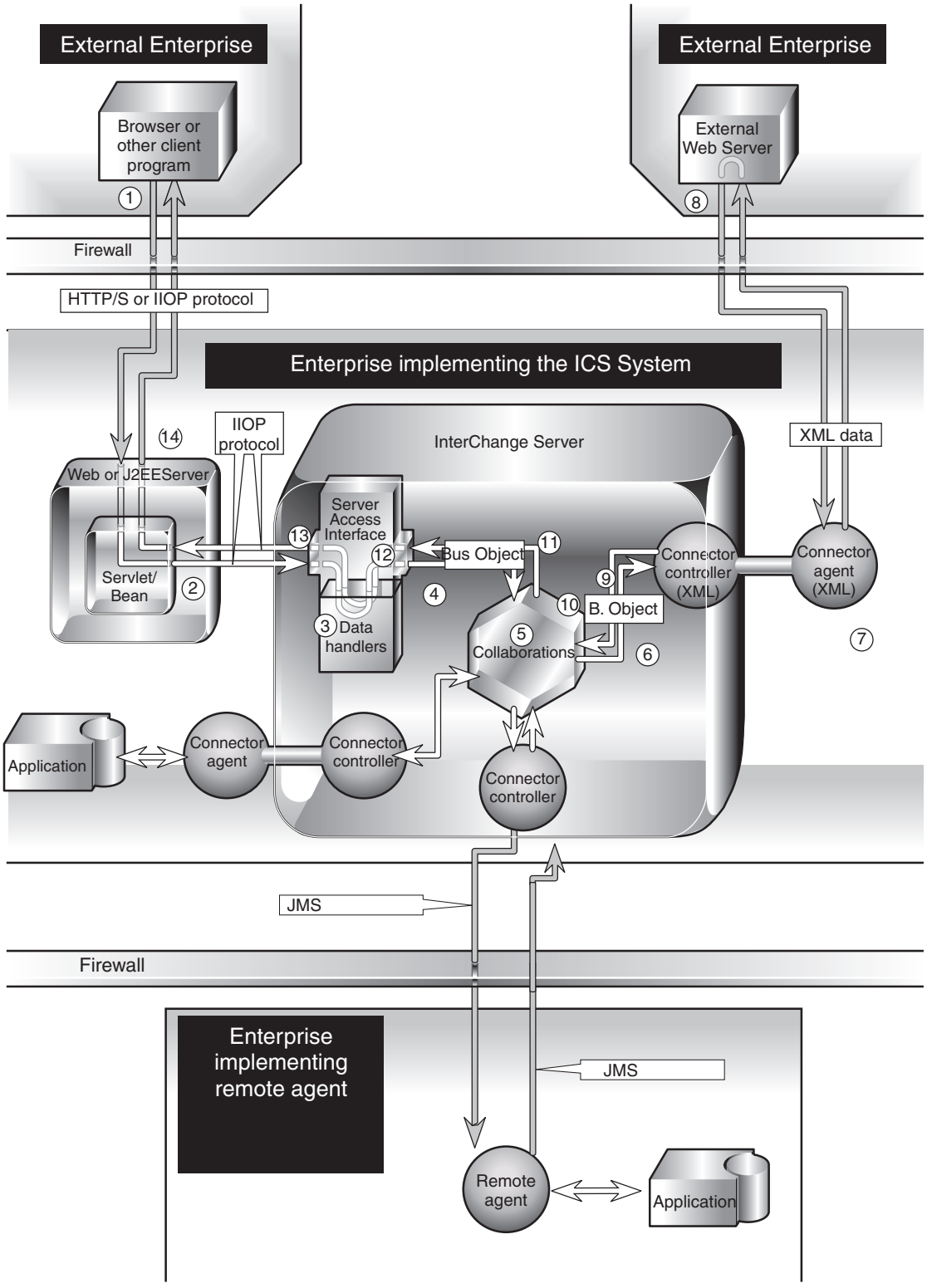

Figure 7. Sample business data flow

In the Figure 7 example, the following process takes place:

1. A client program communicates with a server to send an access request to a collaboration. The client program could be a web browser communicating

with a web server over the HTTP or HTTPS protocol, or a J2EE client making a J2EE method invocation on a J2EE application server using either RMI or IIOP.

2. The call is handled by the appropriate server-side component (a servlet in the case of a web server, and an EJB in the case of a J2EE application server) that has been configured to communicate with the IBM WebSphere InterChange Server system. The component specifies a business object that will be created and a collaboration that will be triggered by the call and sends the call to the Server Access Interface.
3. The Server Access Interface receives the call and passes it to an appropriate data handler (in this example, an XML data handler). The data handler transforms the data into a business object and passes it back to the Server Access Interface.
4. The Server Access Interface passes the business object to the specified collaboration.
5. The collaboration: (a) Performs its business processes on the business object, and (b) synchronously returns a business object to the Server Access Interface, to be passed on to the originating client browser or application. The business object can contain business data that is the result of the processing, or it can be an exception notification.
6. In this example, the collaboration's business logic tells it to send a business object as a request to a technology connector that will be used to send the data across the Internet to an external web server.

   In still other scenarios, the collaboration might instead send a business object to an application connector rather than a technology connector. The application could reside locally or, if the remote agent technology is being used, the application and its connector agent could reside across the Internet, remote from the hub.
7. In this example, the XML connector transforms the business object into an XML document and sends it to a web server.
8. The web server sends a response to the XML connector.
9. The XML connector transforms the response into a business object and sends it to the collaboration.
10. The collaboration performs business processes for that business object.
11. The collaboration sends the business object to the Server Access Interface.
12. The Server Access Interface sends the business object to the appropriate data handler, and receives the data back in that format.
13. The Server Access Interface uses the IIOP protocol to send the data to the servlet.
14. The servlet uses the HTTP or HTTPS protocol to send the data to the originating web browser or other entity.

Connectors

Connectors are supplied as part of the IBM WebSphere Business Integration Adapters product. A connector provides distributed translation services for the IBM WebSphere InterChange Server system, moving data between collaborations and either:

- An application
- A programmatic entity—a remote web server, for example—that understands a technology standard, such as XML, that is handled by a connector

In an InterChange Server implementation, a connector has a distributed structure:

- The **connector controller** interacts directly with collaborations and runs as a component within the InterChange Server process.
- A **client connector framework** runs as a separate process from InterChange Server and, together with an application-specific component, interacts directly with an application or other programmatic entity. In this guide, the client connector framework and the application-specific component are together referred to as the **connector agent**.

The two parts of a connector can run on the same system or on two different systems. The connector controller runs as part of InterChange Server and so resides on that system. However, the connector agent can reside on any system from which it can communicate with both its application and the connector controller.

Connector communication with applications

There is one connector for each version of an application. Each connector is unique, because it communicates with its application according to the application's interfaces. If there is an application programming interface (API), the connector can use it. However, a connector for an application without an API can use whatever method the application provides, such as user exits or email messages.

To detect application events in which collaborations are interested, a connector polls the application or uses the application's event callback notification mechanism, if there is one. A connector can also interact with the application at the command of a collaboration or to verify the results of its previous requests.

Binding between elements

To perform a business process, a collaboration can communicate with connectors, with other collaborations, and with external processes from which it receives access requests through the Server Access Interface. **Binding** is used to set up communications between the collaboration and these elements when the collaboration is configured.

Binding a trigger

To allow a collaboration's business processes to be triggered, you must bind the collaboration at configuration time to an element that will supply the triggering event or call.

Binding is done between the collaboration and any element that will participate in that collaboration's business process, but only one element can be bound as the trigger.

Binding for events

When you configure a collaboration at your site to be triggered by events, you bind it to a connector capable of publishing the triggering event. For example, you might specify that the collaboration's Employee.Delete event comes from the PeopleSoft connector.

To look more closely at this relationship, recall that the connector actually consists of a connector controller (which, like the collaboration, resides in InterChange Server) and a connector agent (which includes the client connector framework and an application-specific component, and is separate from InterChange Server). The

connector controller maintains the binding information and provides its connector agent with a list of events to which collaborations have subscribed.

When relevant application operations occur, the connector agent publishes the events on that list to the connector controller. The connector agent sends an event to the connector controller without knowing anything about its ultimate destination at a collaboration.

The connector controller is therefore an intermediary between the connector agent and the collaboration, as Figure 8 illustrates.

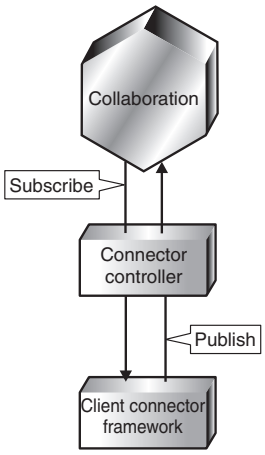

Figure 8. A connector providing a triggering event

Multiple collaborations can subscribe to the same event. When the connector controller publishes the event, it can publish simultaneously to all subscribers.

Binding to receive access requests

Instead of binding the collaboration to a connector for triggering, you can specify that the collaboration will receive access requests from external processes as triggers.

Binding destinations

In addition to binding collaborations to triggering elements, you also bind collaborations to the destination elements with which the collaborations will engage in request/response interactions. Destination elements can be either connectors or other collaborations. A single collaboration can be bound to multiple destination elements.

Business objects

Collaborations and connectors interact by sending and receiving **business objects** through InterChange Server.

A business object reflects a data entity—a collection of data that can be treated as an operative unit. For example, a data entity can be equivalent to a form, inclusive of all of the form's fields. The form might typically be used in an application, or over the Web, to contain business information about customers, or employees, or invoices.

Business objects are cached in memory during collaboration execution for fast access, and also stored in a persistent transaction state store to provide robust recovery, rollback, and re-execution of collaborations upon server restarts after failures.

The IBM WebSphere InterChange Server system creates business objects that reflect the information contained in entities. In this manual, a data entity is often referred to in the context of the kind of business information it contains—for example, an *employee entity* or a *customer entity*.

This section provides a first look at business objects. A later chapter in this guide contains more information on business objects, their contents, and the way that the IBM WebSphere InterChange Server system manages and handles them.

Roles of a business object

A business object can act as an event, a request, or a response.

Event

A business object can report the occurrence of an **application event**, an operation that affected a data entity in an application. The application event might be the creation, deletion, or change in value of that collection of data. When a connector detects an application event and sends a business object to an interested collaboration, the business object has the role of representing the event, and so it is called an **event** in the IBM WebSphere InterChange Server system.

For example, a connector might poll an application for new employee entities on behalf of a collaboration. If the application creates a new employee entity, the connector sends an event business object to the collaboration.

Request

Requests are typically generated in one of two ways:

- A collaboration can send a business object as a **request** to a connector, instructing the connector to insert, change, delete, or retrieve some data in an application. For example, in the service billing collaboration illustrated in Figure 5, the collaboration sends two business objects to connectors, one to retrieve a contract and one to create an invoice. Both are requests.
- The Server Access Interface can send a business object as a request to a collaboration, if that collaboration has been designed or customized to accept the Retrieve verb as a trigger.

Response

When a connector finishes processing a request, it usually returns a **response**. For example, after a connector receives a request to retrieve employee data from an application, it sends a business object containing the employee data.

Structure of a business object

A business object is a self-describing unit that contains a type (its name), processing instructions (a verb), and data (attribute values). Figure 9 is an example of a simple business object, showing its type, verb, and attribute values.

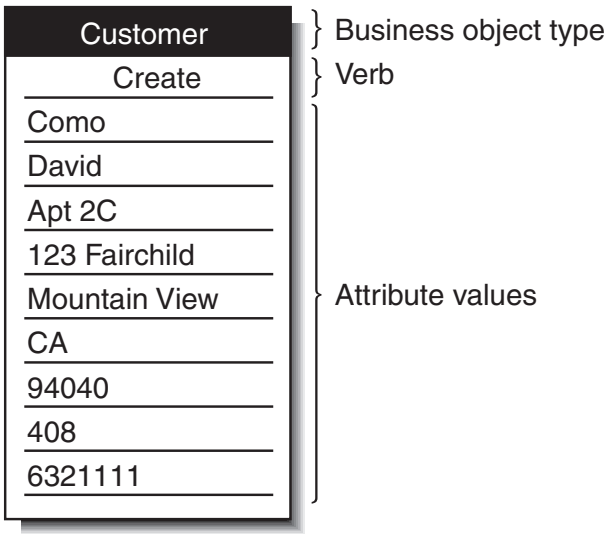

Figure 9. Business object components

The next sections describe these components.

Business object type

Each business object has a type name that identifies it within the IBM WebSphere InterChange Server system. This type is defined by the business object definition. For example, the type might be Customer, Employee, Item, or Contract.

Business object verbs

A business object verb specifies an action in relation to the attribute values. The verb can indicate various types of actions, depending on the role of the business object. Table 1 lists the three business object roles and describes the meaning of the verb in a business object that has each role.

Table 1. Meanings of business object verbs

Role of business object	Meaning of verb
Event	Describes what happened in an application. For example, in an event, the Create verb indicates that the source application created a new data entity.
Request	Tells the connector how to interact with the application in order to process the business object. For example, the Update verb is a request to the connector to update the data entity.
Response	Provides the results of a previous request. For example, in a response, the Retrieve verb indicates that the connector obtained the attribute values from the application.

Note: The naming convention is to use the format *business-object-type.verb* to indicate a particular type of business object with a particular verb. For example, Customer.Create is a Customer business object with the Create verb.

Business object attribute values

A business object contains **attribute values** that represent data fields associated with the data entity, such as Last Name, First Name, Employee ID, or Invoice Status.

Some attributes, instead of containing data, contain **child business objects** or **arrays of child business objects**. Figure 10 illustrates the structure of a Contract business object. The Line Item information in the contract is in an array of child business objects.

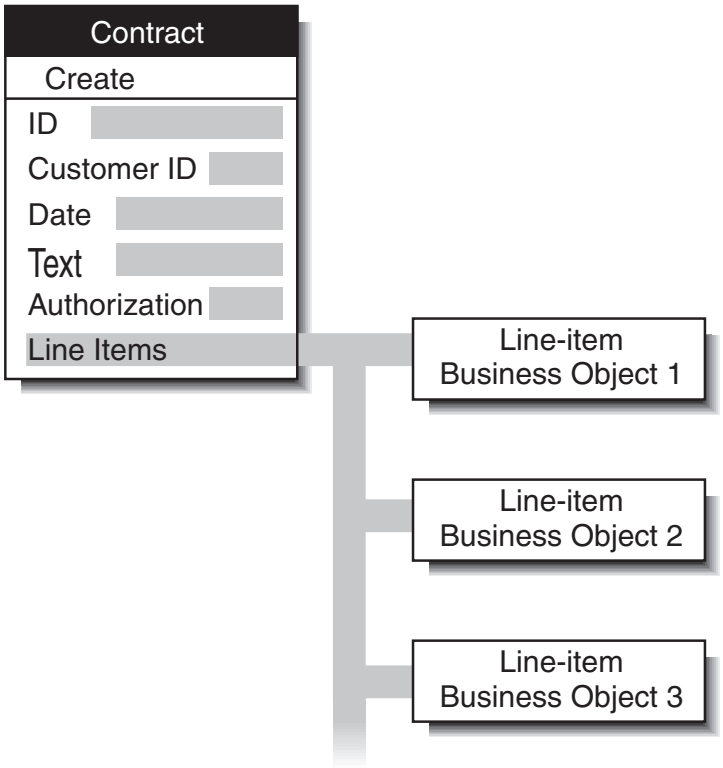

Figure 10. Business object with child business objects

A business object that contains child business objects or arrays of child business objects is a **hierarchical business object**. One whose attributes contain only data is a **flat business object**.

Application-specific and generic business objects

The IBM WebSphere InterChange Server system includes two kinds of business objects: application-specific and generic.

- An **application-specific business object** reflects data entity attributes and the data model of a specific application or other programmatic entity.
- A **generic business object** contains a set of business-related attributes that are common across a wide range of applications, not tied to any specific application's data model

When the connector agent (through its application-specific component) detects an application event such as an update, it retrieves the appropriate data entity from the application and transforms it into an **application-specific business object**.

Note: When this document refers to a business object whose name includes an application name, such as Clarify_Contact or Oracle_Customer, it refers to an application-specific business object. A Clarify_Contact business object, for example, contains the set of information that the Clarify application stores about a contact. In another application, a contact entity might store a

somewhat different set of information, store the information in a different order or format, or have a different name.

After the connector agent has built an application-specific business object, it sends the business object to the connector controller in InterChange Server.

The connector controller exchanges business objects between collaborations and the connector agent. Collaborations are generally application-neutral, so the business object that a connector controller exchanges with a collaboration must be a **generic business object.** The use of generic business objects enhances the reusability of the collaboration because its business logic is not bound to specific versions of specific applications.

Note: Generic business object names do not include a company name or product name. Examples include Contact, Employee, and Customer.

Figure 11 shows where the two kinds of business objects fit within the IBM WebSphere InterChange Server system: the collaboration interacts using generic business objects and the connector agent supports business objects designed for specific applications.

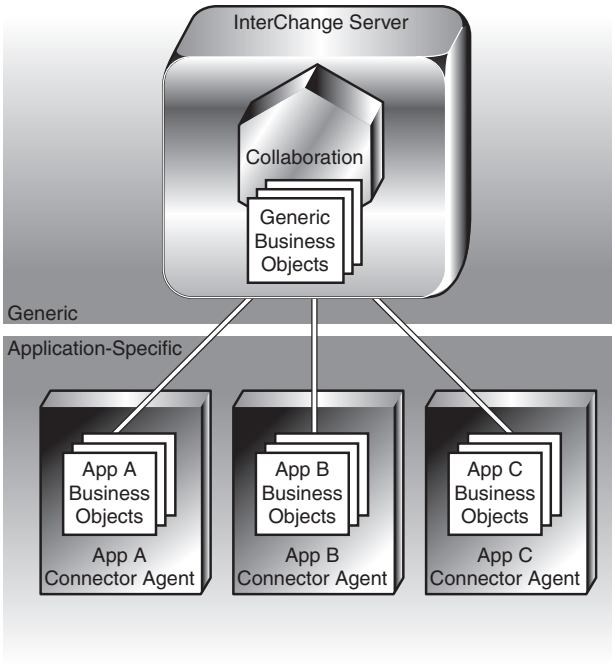

Figure 11. Generic and application-specific business objects

You can use the same connector to run multiple types of collaborations, if the connector supports the business objects used by those collaborations.

Data mapping

Figure 11 shows different types of business objects at each application and at the collaborations. The IBM WebSphere InterChange Server system, therefore, must convert business objects into like forms so that it can send events and data across applications and collaborations. **Data mapping** is the process of converting business objects from one type to another. Data mapping is required whenever the IBM WebSphere InterChange Server system sends data between a source and a destination that does not exactly share the source's data model.

Unlike custom application integration solutions that map data directly from one application to another, InterChange server collaborations generally use the generic business object between the application-specific data models. The generic business object serves as a common, cross-application data set. If you change applications in the future, you need only get a new connector and map the new application-specific business object to the generic business object. Collaborations then continue to work as they did previously.

Whenever a collaboration transfers a business object across dissimilar applications, mapping transforms the business object to and from the common data set. Business object transformation takes place:

- From application-specific to generic when business objects pass from connectors to collaborations
- From generic to application-specific when business objects pass from collaborations to connectors

For example, in a collaboration that synchronizes Clarify_BusOrg data with SAP_CustomerMaster data, mapping occurs twice:

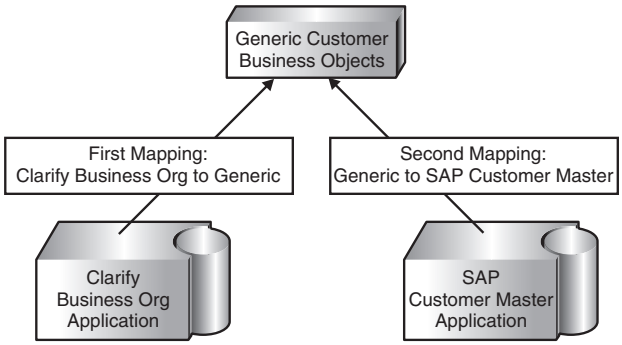

Figure 12. Mapping transformations

Each connector controller manages the mapping of business objects that pass between its connector agent and InterChange Server. To actually perform data mapping, however, the system invokes the use of the mapping tools—Map Designer and Relationship Designer. These tools let you create and modify detailed mapping specifications and execute mapping at runtime.

A connector controller invokes the mapping function when it receives business objects that require mapping. Figure 13 illustrates the invocation of mapping from the connector controller.

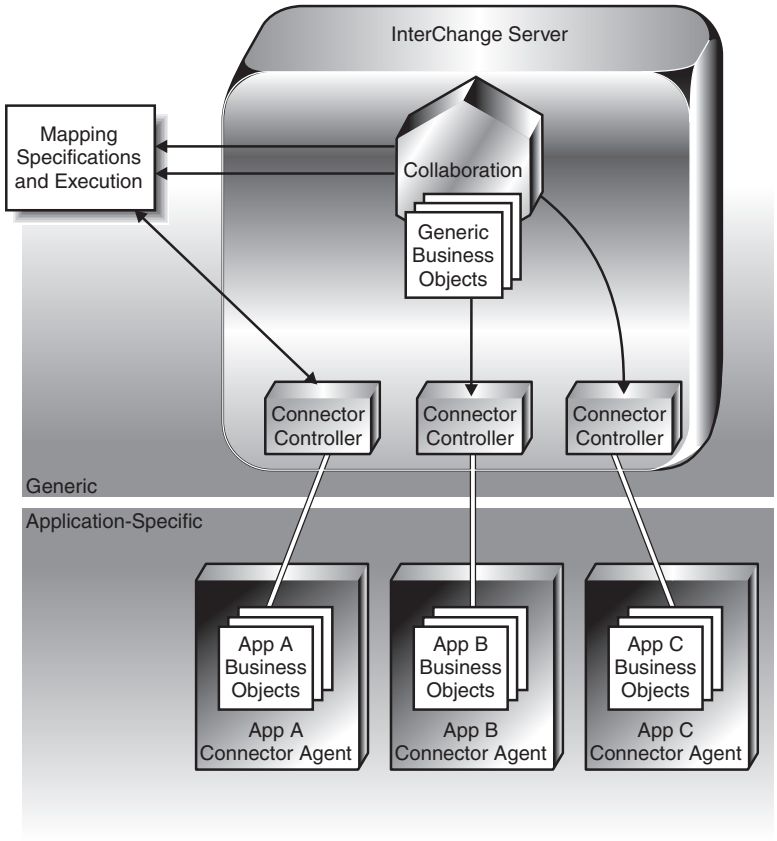

Figure 13. Mapping control and execution

InterChange Server

InterChange Server is a multi-threaded, Java-based execution framework for collaborations. InterChange Server runs within its own Java Virtual Machine (JVM). InterChange Server runs on Windows and UNIX systems. This section describes the following the services and features of InterChange Server:

- "Event management service"
- "Connector controllers" on page 24
- "Repository" on page 24
- "Database connectivity service" on page 24
- "Database connection pools" on page 24
- "High availability" on page 25
- "Transaction service" on page 25
- "Recovery features" on page 27

Event management service

InterChange Server persistently stores every business object that it receives during the execution of a collaboration. This allows InterChange Server to recover from an unexpected termination or from the failure of a collaboration without losing event notifications or calls.

Connector controllers

A connector controller is an interface between the client-side of a connector and InterChange Server. A connector controller routes business objects as they traverse the IBM WebSphere InterChange Server system, linking the client-side of connectors to collaborations and managing the mapping process.

Through the connector controller, an administrator can:

- Trace interactions between InterChange Server and the connector agent.
- Enable and disable interactions between InterChange Server and the connector agent.
- Specify the type of mapping to perform for each business object arriving from or departing to the connector agent.

Repository

InterChange Server maintains configuration information and definitions of all objects in a persistent store called the **InterChanger Server repository**, which consists of a set of tables in a relational database. The tables store object definitions and configuration information in the form of XML documents.

Database connectivity service

The **database connectivity service** manages interactions between InterChange Server and the repository. The database connectivity service interacts with the repository by means of the Java Database Connectivity API (JDBC), as Figure 14 illustrates.

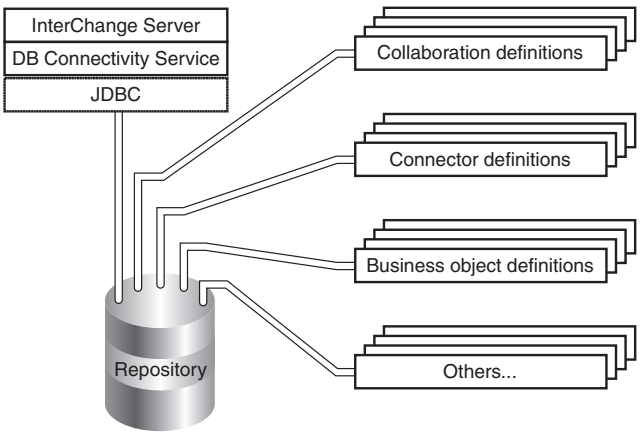

Figure 14. Database connectivity service and the repository

Note: Some databases have Open Database Connectivity (ODBC) drivers but do *not* have native JDBC drivers. For those databases, you can use an available JDBC-to-ODBC bridge.

An InterChange Server implementation supports a number of database vendors and continues to certify additional databases. For the current list of supported databases, refer to the *System Installation Guide for Windows* or *for UNIX*.

Database connection pools

You can use the System Manager tool of the IBM WebSphere InterChange Server system to define database connection pools in InterChange Server. User-defined

database connection pools make it possible for developers to directly access relational databases from within a collaboration or map. This feature provides support for

- Automated database connection life-cycle management
- Simplified APIs for SQL statements and stored procedure execution
- Container-managed transaction bracketing

High availability

The IBM WebSphere InterChange Server system can be configured to provide high availability (HA) for InterChange Server (ICS).

On Windows systems, the HA configuration runs under Microsoft Cluster Server (MSCS) software. The HA configuration requires two Microsoft-certified cluster server machines. The servers are set up with identical InterChange Server system configurations and are designated in the MSCS software as a cluster system. One machine is configured in MSCS as the primary (the active server until a failure occurs), and the other as the backup. The two machines share a cluster name and a cluster IP address that external processes use for accessing the active server. Both the primary and backup servers have access to a shared RAID storage system, which is controlled by the active server.

When a hardware failure is detected, the HA configuration provides shutdown of the ICS, and automatic migration and restart of the ICS (and the third-party software, and HA-supported connectors) on the cluster backup system. The cluster backup server becomes the active server, assuming the cluster name and IP address of the primary server, and automatically takes over system processing until such time that you correct the failure on the primary server and initiate a failback (that is, manually return processing to the primary server)

For an overview of the HA environment, see the *System Administration Guide*. For information on how to configure ICS for use in an HA environment, see your *System Installation Guide*.

Transaction service

Any application integration solution encounters risks as it moves data from one application to another. When data leaves the protected realm of one application and its database and travels across a network to another application, any number of problems can occur. For example:

- The network might go down.
- The destination application might crash before receiving the data.
- An error might occur in the destination application before it has a chance to process the new data.

Consider a collaboration that involves three applications: human resources, payroll, and product cost accounting. Each stores employee salaries. Figure 15 is a high-level view of the cross-application business logic for handling an employee's salary increase.

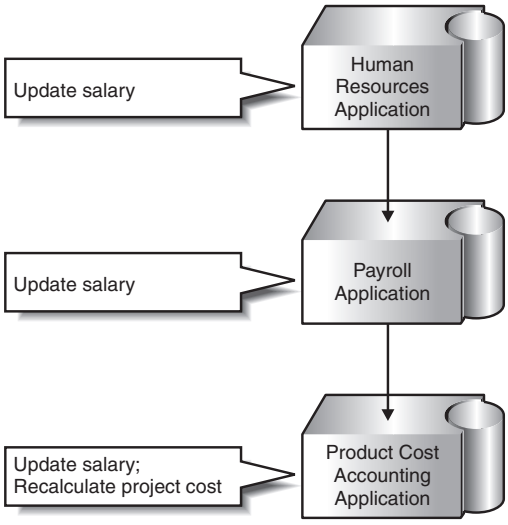

Figure 15. Three applications process employee salaries

When an employee gets a raise, someone enters a new salary into the human resources application. The collaboration automatically updates the employee salary in the payroll application, and then adjusts the project cost in the product cost accounting application.

If a system error prevents the salary update from reaching the product cost accounting application, data inconsistency can result. The value for the employee's salary is now different in the various applications, and the product cost is incorrectly computed.

Traditionally, painstaking cross-checking was required or the data remained faulty until someone noticed the problem. The IBM WebSphere InterChange Server system, however, offers a higher level of service with strict controls to which you can trust enterprise data, with services that can run a collaboration as if it were a type of **transaction**.

Transactional collaborations

Transactional qualities are desirable and achievable for collaborations where data consistency is important across applications. Like other transactions, a transactional collaboration involves a set of steps. If an error occurs, InterChange Server can undo each completed step, performing a transaction-like rollback.

However, collaborations are different from traditional transactions in some important ways:

- Collaboration actions are distributed, and there is no centralized control over the participating databases.
- Collaborations that respond to events (as in the publish-and-subscribe model) are long-lived; they execute asynchronously, because to isolate application data while they execute would adversely affect application users.
- Applications save data changes caused by collaborations, thereby providing a distributed, cross-application form of durability. However, if a collaboration needs to roll back, it might need to undo previously saved operations.

The techniques that InterChange Server uses to support transactional collaborations, therefore, differ from those that support traditional transactions. The

transaction levels associated with collaborations define the rigor with which InterChange Server enforces transactional semantics.

Recovery features

An InterChange Server implementation provides features for improving the time it takes ICS to reboot after a failure, for making ICS available for other work before all flows have been recovered, and for controlling the resubmission of failed events:

- Asynchronous recovery

  InterChange Server does not wait for collaborations and connectors to recover before it completes boot-up; collaborations and connectors are allowed to recover asynchronously after InterChange Server has booted. This makes it possible to use System Manager troubleshooting tools, such as System Monitor and the Administer Unresolved Flows dialog, while the connectors and collaborations are still recovering.

- Deferred recovery

  Use of this feature is optional and is configured through the use of collaboration object properties. If you enable this feature for a collaboration, when an ICS failure occurs, the recovery of the collaboration's WIP flows is deferred until after the server has rebooted, thereby saving the memory usage associated with those flows. After the server has rebooted, you can use the Administer Unresolved Flows dialog in System Manager to resubmit the events.

- Persist service call in-transit state

  You may want to prevent a recovery from automatically resubmitting all service calls that were in transit when a failure occurred, to avoid the possibility of a nontransactional collaboration sending duplicate events to a destination application. This is accomplished by configuring the collaboration (prior to the server failure) so that it will persist any service call event in the In-Transit state when a failure and recovery occurs. When InterChange Server recovers, the flows that were processing service calls remain in the In-Transit state, and you can use the Administer Unresolved Flows dialog to examine individual unresolved flows and control when (or if) they are resubmitted.

- Guaranteed event delivery features

  For JMS-enabled connectors (connectors that use JMS as their transport mechanism), the following features can be useful for guaranteed event delivery in recovery situations:

  – Container managed events

    The container managed events feature is valid for JMS-enabled connectors that use a JMS event store. It ensures that if a system crash and recovery occurs, an event that was in process between the event store and the connector framework will be received exactly once by the connector framework, and will not be delivered twice. This feature is optional and configured through connector properties; it is used only with connectors that use JMS as their transport mechanism.

  – Duplicate event elimination

    The duplicate event elimination feature, valid for JMS-enabled connectors, uses unique event identifiers in the application-specific code of the connector to ensure that events will not be delivered in duplicate to the delivery queue. This feature is optional and configured through connector properties; it is used only with connectors that use JMS as their transport mechanism

Communication transport infrastructure

The modular architecture of an InterChange Server implementation constitutes a distributed system that accommodates many different types of site configurations. InterChange Server implementation architecture enables distributed interaction over multiple machines on a network, and distributed interaction across Internet firewalls.

Distribution on a network

Interactions between distributed InterChange Server components on a network are enabled by the Common Object Request Broker Architecture (CORBA) and by messaging technologies, including native WebSphere MQ messaging and Java Messaging Service (JMS).

Different configurations are possible. CORBA might be used for request/response interactions and administrative communications between the connector agent and ICS, with native WebSphere MQ performing the delivery of events in publish-and-subscribe operations. Alternatively, Java Messaging Service might be used for all communications and interactions between a connector agent and ICS.

CORBA

The Common Object Request Broker Architecture (**CORBA**) defines a set of standards and interfaces for distributed objects on a network. The Object Request Broker (**ORB**) is a set of libraries and other components that client applications and object servers use to communicate. InterChange Server uses the IBM Java ORB product.

The ORB makes InterChange Server accessible to its clients, the connector agent, and System Manager. An InterChange Server registers with the ORB's name service, from which a client obtains the information it needs to find and start interacting with the server. The client and server perform object-to-object interactions by means of the ORB's Interface Definition Language (IDL). At a transport level, they communicate by means of the Internet Inter-ORB Protocol (IIOP). ORB-based communication is typically used for the following purposes:

- The Server Access Interface uses ORB-based communication to handle calls.
- In request/response interactions, collaborations and connectors use ORB-based communication to exchange business objects.
- A connector agent uses ORB-based communication:
  - At startup, when it interacts with InterChange Server to get its initial configuration.
  - During operation, when it receives directives from the connector controller to report its status, pause, stop, or resume.
- Optionally, ORB-based communication can also be used for event delivery in a publish-and-subscribe interaction.

In the IIOP request/response protocol, communication either succeeds or fails immediately, so both programs must be running for the components to communicate. However, for request/response interactions in an InterChange Server implementation, you can use a connector property to set a store-and-forward mode, specifying how a connector controller will respond to a collaboration's request in a situation where the connector agent is unavailable:

- If you set the property to True, the connector controller won't fail any collaboration requests even if the connector agent is unavailable. A request is

blocked until the connector agent is operational. This causes a collaboration to wait until the connector agent is operational before it completes the processing flow for the request.

- If set to False, the connector controller fails all collaboration requests if the connector agent is unavailable. This causes a collaboration to complete the processing of the request according to its business logic for processing a failed request.

Figure 16 shows the components that constitute the IBM Java ORB for use with InterChange Server. There are different components for the different languages in which InterChange Server components address their ORB drivers, but the C++ and Java components communicate with each other as one system. The IBM Transient Naming Server provides the naming service.

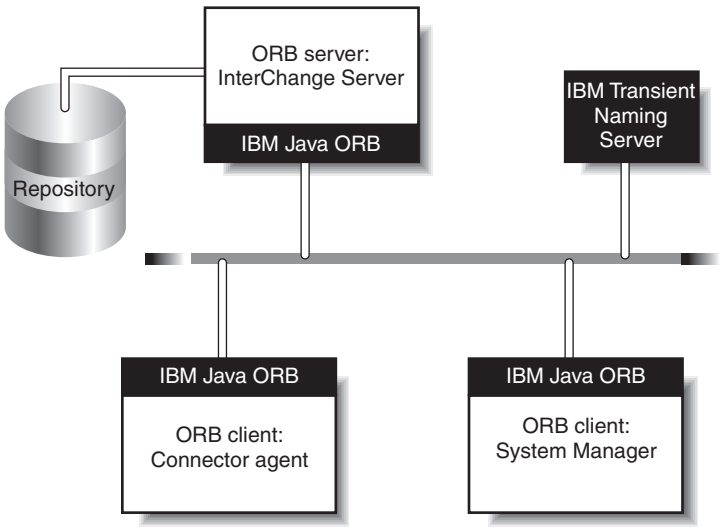

Figure 16. ORB components in the WebSphere InterChange Server system

These ORB components need not be separated. At some sites, all of them might reside on the same machine.

Messaging technologies

Messaging embodies a communication style in which programs asynchronously exchange discrete units of data (messages). Programs that use a messaging transport need not establish connections or wait for messages; each program asynchronously sends and receives messages by interacting with the messaging service. The messaging service provides guaranteed delivery, storing the message if the destination program is unavailable and retrying until it is available.

Messaging systems supported with an InterChange Server environment include both native IBM WebSphere MQ messaging and Java Messaging Service (JMS) software. A common implementation approach is to use a messaging system for the delivery of events in publish-and-subscribe interactions, and to use the ORB for request/response interactions. However, there can be circumstances where JMS is used for both types of interactions.

When JMS is the delivery transport mechanism, data persistence can be provided through the long-lived business processes feature. When this feature is used, a process initiated by a request on a collaboration can be placed in a waiting state with a timeout value, so that the process will be resumed if and when a specified

data response is received. Use of this feature requires that the feature be enabled during the creation of the collaboration template.

Distribution across the Internet

The InterChange Server system supports secure socket layer (SSL) communication for distributing connector controller/connector agent interactions across Internet firewalls.

The implementation is a hub-and-spoke relationship, in which the hub is a site that has installed a complete IBM WebSphere InterChange Server system, and the spokes are remote sites that exchange data with the hub across firewalls. A spoke site requires a remote connector agent, but does not require a complete IBM WebSphere InterChange Server system.

Two alternative configurations can be used:

- WebSphere MQ with SSL
- WebSphere MQ internet pass-thru using HTTP/HTTPS

Securing InterChange Server

The WebSphere Business Integration system is a distributed system with critical business data that needs to be secured. Unauthorized parties need to be prevented from viewing or modifying the data while it is in transit or when it is stored on disk. Securing data in any system includes three security services, authentication, message integrity, and privacy. Authentication involves verifying that someone is who they claim to be. Authentication governs access control which restricts access to parts of the system based on who the authenticated user is and what permissions the user has been granted. Message integrity ensures that the data has not been modified. Privacy ensures that only authorized users can view data. Integrity, privacy and authentication can be accomplished with encryption, while RBAC can be accomplished by setting up user IDs and passwords.

Encryption

Besides the three security services, there are three security mechanisms that are involved with securing InterChange Server. Two of the mechanisms involve encoding data through encryption. They are **symmetric encryption** and **asymmetric encryption**. Symmetric encryption involves encoding data using an encryption algorithm, or key. The same key, is used to encode the data and decode the data. In a distributed system, where one process has to encrypt the data and another process has to decrypt the data, the key needs to be securely shared or exchanged, so that unauthorized users cannot use the key to access the data. The ability to share a key can also be a challenging task in symmetric encryption.

The mechanism of asymmetric encryption is commonly used to share or exchange a key. It involves encrypting data and decrypting data using a **key pair**. One key in the key pair is a **private key** that is kept secret, and the other key in the key pair is a **public key** that is shared and distributed to others. Data encrypted with a public key can only be decrypted by the corresponding private key in the key pair. The sender encrypts the data using the public key of the receiver, so that only the appropriate receiver can decrypt the data using its own private key. Since asymmetric encryption is that it is slow, it is not used to encrypt data. Instead, it is used to asymmetrically encrypt and decrypt the secret key, which in turn is used

to symmetrically encrypt the data. This process affords the best mix of security and speed because it is the strength of the key that is more important in maintaining privacy within a system.

The private key can also be used to create a **digital signature** for the data, which can be used to verify the identity of the sender (authentication) and the integrity of the data sent (message integrity). To create a digital signature, the data is a small size fixed-length **message digest** in a process called **hashing**, which is a one-way function that creates a map or message digest. The message digest is encrypted with the sender's private key, which creates the digital signature. The digital signature is appended to the data, and sent to the receiver. The receiver decrypts the digital signature back into the message digest using the public key, which verifies the signature, since the public key can only decrypt data encrypted with the private key, which only the sender has. Then the receiver hashes the data to create a message digest, and compares it with the message digest that was sent. If both message digests are the same, the receiver can verify that the data has not been changed since it was signed by the sender.

To secure critical business data, InterChange Server includes the following security features:

- End-to-end privacy, (also labeled end-to-end security), which secures data as it flows from a source adapter process, through the InterChange Server, to the destination adapter process. End-to-end privacy uses asymmetric and dynamic security in addressing authentication, integrity and privacy.
- Role-based access control, which restricts access to parts of the system based on who the authenticated user is and what permissions the user has been granted. Role-based access control addresses access control and authentication.

End-to-end privacy

Adapter processes and InterChange Server communicate by sending messages. Messages are bidirectional, so that both InterChange Server and the adapter processes send messages and receive messages. SSL offers node-to-node, or link level security. Messages can be secured as they travel between nodes, but not when they are stored to disk at a node. The messages and data are in the clear, where they can be viewed and modified by unauthorized users. End-to-end privacy can secure the messages as they flow starting at the node of origin, through any intermediate nodes, to the destination node. Using asymmetric and dynamic security, messages are secured in transit and are secured as they are stored when they are at queue managers on disk, awaiting processing. Messages are secured with varying levels of security, which can include encryption and **end point authentication**. End point authentication occurs when the end points in the communication, such as InterChange Server or an adapter, prove they are who they claim to be by using digital signatures.

Asymmetric and dynamic security

In a distributed system, communicating parties can be both senders and receivers. When the communication needs to be secured, and the security levels are the same for incoming and outgoing messages, it is considered symmetric security. This type of security creates a tight coupling between the server and an adapter. SSL represents a form of symmetric security.

Many third party security products are based on the **Secure Sockets Layer** (SSL) security protocol. SSL provides data encryption, server authentication, message integrity, and optional client authentication for a TCP/IP connection. SSL is usually associated with tightly coupled systems that use symmetric security, so all data is

encrypted whether the data is critical or not. SSL cannot tell non-secure data from secure data. If the security levels between incoming and outgoing messages can be different, it is considered asymmetric security. For example, suppose a user needs to authenticate with a server, using a user ID and password, before receiving a large document from the server. The user ID and password need to be encrypted, so that they cannot be viewed in transit by an unauthorized user, but the document does not need to be encrypted when in transit. Using SSL, both the user ID and password and the large document are encrypted. Encrypting the large document can impact the performance of the server.

Using asymmetric security, the message flow from the user can be set to encrypt the user ID and password, but the message flow from the server to the user can be set so it is not encrypted. The user ID and password from the user are protected by encryption, while server performance is enhanced because the large document is not encrypted before it is sent to the user. Asymmetric security also enables dynamic updates to security parameters between communicating parties. Using asymmetric security, such as SSL, security parameters are established when the communication starts. Changes to security parameters are not in effect until a new communications session begins. Using asymmetric security, changes to the security parameters can be dynamic because the communicating parties have individual security parameter definitions. A change to the security parameters for one party does not interrupt the current security session and require an update to other parties. While asymmetric security can provide flexibility, its weakness is that it directly couples with end nodes leaving the loosely or indirectly coupled middle nodes vulnerable to data to manipulation by unauthorized sources.

Using end-to-end privacy, security can be set at the application layer in order to secure messages before they enter and leave an application. This type of security eliminates the danger of unauthorized sources accessing messages while residing on middle nodes. End-to-end privacy also provides a way that different message types can have different levels of security.

Levels of security

The level of security is configured for each message type, for individual adapters and InterChange Server. There are four levels of security:

None No security level is set, so messages are sent from an adapter or InterChange Server as normal. This is the default level of security.

Integrity
The sender, either an adapter or InterChange Server, adds a digital signature to the message. The receiver verifies the signature using the public key of the sender. The receiver is authenticating the sender by verifying the signature, since only the sender has access to its own private key, and could use it to generate the signature. The signature is a message digest of the data, so when the receiver creates a message digest and verifies that the message digest matches the message digest sent with the message, the message integrity is guaranteed. Integrity also implies authentication, since in the process of verifying that the data has not changed, the identity of the sender is verified.

Privacy
The sender completely encrypts the message. The receiver decrypts the message and passes it on for further processing. When an adapter or InterChange Server sends a message with Privacy configured, it generates a secret key; encrypts the message with the secret key; encrypts the secret key with the receiver's public key; includes the encrypted secret key with the message; then sends the message. The receiver decrypts the secret key,

using its own private key; decrypts the message using the decrypted secret key; and forwards the message for further processing.

The secret key is encrypted with the public key of the receiver. The encrypted secret key can be decrypted only with the private key of the receiver, so the message is secure because only the receiver has access to its own private key.

Integrity plus Privacy
The sender adds a digital signature to each message, as described in Integrity, and then completely encrypts the message, as described in Privacy.

InterChange Server and the adapters maintain their own keystores. A keystore is a password protected file used to securely store public and private keys. InterChange Server maintains a keystore that contains its own public and private key pair, and the public keys of all installed adapters. An individual adapter maintains a keystore that contains its own public and private key pair, the public key of InterChange Server and the public keys of all the processes to which it communicates.

Combinations of the levels of security and the types of messages to secure are possible, for individual adapters and InterChange Server. For example, Integrity can be configured for administrative messages from InterChange Server, while Privacy can be configured for business object messages from the SAP adapter, and no security can be set from the e-mail adapter.

Message types

There are four types of messages, coming from an adapter or InterChnage Server, that can be secured:

All messages (All)
All messages from an adapter or InterChange Server are secured.

Administrative messages (Admin)
All administrative messages from an adapter or InterChange Server are secured.

All Business Objects (BO)
All business object messages from an adapter or InterChange Server are secured.

Individual Business Object (BO specification name)
Specific business object messages from an adapter or InterChange Server are secured by specifying the name of the business object specification to be secured.

To secure messages using end-to-end privacy, an administrator needs to determine what type of messages to secure, what level of security is required, and then configure both InterChange Server and individual adapters. When end-to-end privacy is configured for InterChange Server, it only applies to messages sent from InterChange Server. When end-to-end privacy is configured for an individual adapter, it only applies to messages sent from the individual adapter. All existing events must be removed from the system before turning on end-to-end privacy. Existing events will not be processed correctly when end-to-end privacy is configured. For more information concerning configuring end-to-end privacy, see the *System Administration Guide*.

Role-based access control

To prevent unauthorized users from accessing any system, user IDs and passwords address a basic security requirement. A user ID is issued to a user, who is an individual or a process that wants to access the system. Authenticating a user does not mean that they should have access to all parts of the system. There might be critical or sensitive parts of the system that should only be accessed by select users, while other parts of the system can be accessible to all users. Access control refers to restricting access to parts of the system based on who the authenticated user is and what permissions the user has been granted. Individual users can be granted permission to access various components, but managing permissions in this way can be time consuming for administrators. Role-based access control authenticates users who want to access InterChange Server, then limits the access the user has to individual components based on what role the user has been assigned.

Roles

A role is a collection of one or more users that have some functional association. Assigning permissions based on roles reduces the administrative burden significantly. For each operation that needs to be secured, the administrator defines roles that are allowed to perform the operation. Only users that are members of the permitted roles are allowed to perform the operation. When permissions are assigned to roles, the role permissions are referred to as the security policy for the role. Users can be assigned to multiple roles.

An operation or an action is not secured if no roles are assigned to it, and every authenticated user can perform the action. Users are only allowed to perform an action that is secured if the user has been assigned to a role that has been granted access to the action.

A predefined role of administrator has been included with InterChange Server as a default role. Members of the administrator role have permission to perform all operations on the server, including creating users, creating roles and granting permissions to roles. The administrator has the ability to create additional roles that better reflect the organizational structure. For example, an organization may have several developer roles such as Finance Developers or Redevelopers to restrict access to the components developed by the different groups. Users can belong to multiple roles. Default roles cannot be deleted.

An administrator can query the system for users who are currently logged in to the system. A user can be logged in to a maximum of 20 sessions at one time. The administrator can logout the particular session of any user that is currently logged in. The administrator can also logout all the open sessions of a user, so that if a user has forgotten to logout, the administrator can logout for the user. An exception is thrown when a user tries to log in and already has 20 open sessions. An administrator can also reset a user's password.

A guest user has been predefined as a default user, and assigned a password of guest. Default users cannot be deleted. There is no limit to the number of sessions that guest can log into.

InterChange Server administrator can import or export security policies, which are the permissions assigned to components, as security.xml. Role definitions and role membership information can be exported to and imported from a file, from the System Manager and from the command line tool *repos_copy* by a user with the appropriate role. The file is named `RoleMembership.xml`. Users and passwords can be exported to and imported from a binary file, which can be imported to another

server. Exporting passwords is a security risk. The file should be exported to a directory that is secured by the operating system security and should be deleted after it is imported.

The default security policy is that only a user in the administrator role can start, stop, administer security, export config files and deploy config files in InterChange Server. The default permissions for a new component are that all actions can be performed by an authenticated user. A component is not secured by default.

Role-based access control is *not* automatically configured during InterChange Server installation. To configure role-based access control, refer to the *System Administration Guide*. Before turning on role-based access control, at least one user must be added to the administrator role. If InterChange Server uses a repository implementation, a user needs to be created to add to the administrator role. If role-based access control has been turned on and the server is started without at least one user in the administrator role, the server logs an error message and starts with role-based access control turned off. A user ID and password are not required to start the server, and all users have access to all operations. When role-based access control is turned on, a user ID and password are required to start the server and all permissions or access checks are functional.

Auditing

Auditing allows administrators to track who performed secure operations, such as logging in, logging out, and administering users, roles and permissions. Auditing data that is collected includes the date and time, user ID, operation performed and status of the operation, such as success, access denied or an exception occurred. The audit log is not written to the usual Interchangesystem.log file. It is written to a separate file in the *your_ics_dir_name/log/audit directory*, which is not secure. The audit log file contains information that should be secured by the operating system, so that unauthorized users cannot view the files. The size of the log file and the frequency with which new log files are created can be configured. The name of the file depends on how often a new log files needs to be created, after the log file exceeds the file size settings. Frequency options include daily, weekly and monthly. The default value is weekly. The format of the audit log file name is

```
Daily file: icsaudit_yyyymmmdd_nnn.log
Weekly file: icsaudit_yyyymmmwx_nnn.log
    Where x = 1,2,3.... Week numbers
Monthly file : icsaudit_yyyymmm_nnn.log
```

where yyyy is the year, mmm is the month, dd is the day, wx is the week number and nnn = 001.....999, when multiple files need to be created in a period because the log file size setting has been exceeded. The operations that can be audited include:

- Login and logout
- Server start and stop, security, administering roles and users, monitoring, view failed events, MMS - deploy, export, delete, compile, export configuration files, and deploy configuration files
- Collaboration object start, stop, pause, shutdown, execute (AccessFramework call), resolve transactional status, submit Failed events, delete Failed events, cancel LLBP flow
- Connector start, stop, pause, shutDown agent, submit failed events, delete failed events
- Map start and stop
- Relationship start and stop
- Benchmark start and stop

User Registry

The user registry is the system that stores usernames and passwords. By default, users and passwords are stored in the local WBI repository. The userRegistry configuration parameter can be used to configure InterChange Server to use a Lightweight Directory Access Protocol (LDAP) directory as a user registry. The LDAP is a protocol for accessing enterprise directory services. Enterprise directories are used to store information that can be shared across several applications. For example, user information that may be required by all enterprise applications may include username, password and email address. If this data is stored in the proprietary database of each individual application, the data is duplicated, and it is difficult to be kept synchronized. When a new employee joins the company, a user ID and password have to be created for the employee in each application. However, with a directory service, the user ID and password only need to be created once in the directory, and enterprise applications can reuse that information.

The provider is set to LDAP to use an LDAP directory for the user repository, while it is set to REPOS to use the local WBI repository as the user registry. When the WBI repository is used for storing users and passwords, it is strongly recommended that a separate database be used for this. The URL for the database that hosts the user registry can be specified. This is a hierarchical property called USER_REGISTRY in the configuration file interchangesystem.cfg. The structure is identical to the existing REPOSITORY property such that a databaseURL, username and password can be specified. The property can be edited using the server configuration tool. It is recommended that there be one user registry per enterprise, whether using LDAP or a WBI database, so that multiple instances of InterChange Server can share the same user registry without importing and exporting users and passwords. InterChange Server can use the username and password from an LDAP directory, using the InetOrgPerson schema as described in RFC 2798. The InetOrgPerson schema has Object Identifier (OID) 2.16.840.1.113730.3.2.2, which are numbers assigned by IETF that uniquely identify each schema. InterChange Server makes no assumption on the directory tree structure within the directory. Users can specify a base distinguished name (baseDN) as part of configuration for both users and role. This baseDN is used as the root for any searches or updates.

Securing components and actions

The list of components that can be secured, along with securable actions, includes:

Table 2. Securable components

Securable components	Securable Actions
InterChange Server	• Start • Shutdown • Security - administering roles and users • Monitor • View failed events • MMS - Deploy • MMS - Export • MMS - Delete • MMS - Compile • MMS - Export config files • MMS - Deploy config files

Table 2. Securable components (continued)

Securable components	Securable Actions
Collaboration objects	• Start • Stop • Pause • Shutdown • Execute (Access Framework call) • Resolve transactional status • Submit failed events • Delete failed events • Cancel LLBP flow
Connectors	• Start • Stop • Pause • Shutdown agent • Submit failed flow • Delete failed flow
Maps	• Start • Stop
Relationships	• Start • Stop
Benchmarks	• Start • Stop

Summary

This chapter introduced the major components of the IBM WebSphere InterChange Server system. The key points to remember are:

- Collaborations interact with connectors and with the Server Access Interface to create distributed business processes. Collaborations contain the business process logic; connectors bridge the IBM WebSphere InterChange Server system and the application environment to help implement the business process logic across applications. The Server Access Interface and the XML Connector let you extend the business process logic across the Internet.
- All interactions involving a collaboration take place through the exchange of business objects. When the Server Access Interface receives a call, or when a connector receives information about an application event, a business object is created and sent to the collaboration.
- A business object is generic or application-specific. Collaborations process generic business objects, and connectors process application-specific business objects.
- InterChange Server is the center of collaboration execution. It includes a repository, connector controllers, the event management service, the database connectivity service, and the transaction service.
- System Manager displays, configures, and controls the execution of the objects stored at each InterChange Server.

- Components are distributed. The collaboration/connector protocol, messaging, and the ORB provide the glue between components.
- Messages in InterChange Server can be secured using end-to-end privacy. InterChange Server can be secured using role-based access control.

The next chapter takes a closer look at collaborations.

Chapter 2. Tools for use with InterChange Server

Available with InterChange Server is the following set of tools:

WebSphere Business Integration Toolset

The WebSphere Business Integration Toolset provides administrative and development tools for use with InterChange Server. The WebSphere Business Integration toolset is only supported on Windows 2000 and Windows XP. Administrative tools include:

- System Manager
- System Monitor
- Flow Manager
- Log Viewer
- Relationship Manager

For more information about administrative tools and tasks, see the *System Administration Guide.*

Development tools include:

- Map Designer
- Relationship Manager
- Process Designer
- Business Object Designer
- Connector Configurator

These development tools are described in more detail in the *Implementation Guide for WebSphere InterChange Server*, and in the development guides for different components.

The WebSphere Business Integration Toolset tools can be accessed through a Windows shortcut by selecting **Start > Programs > WebSphere Business Integration Toolset**, then selecting **Administrative** for the administrative tools, or **Development** for the development tools.

The IBM WebSphere Business Integration Toolset provides enablement for use with IBM Tivoli License Manager (ITLM). It provides support for inventory functions. ITLM provides more control and flexibility over what is paid for software, and it enables customers to pay for software based on utilization rather than full machine capacity. ILTM measures the usage of products and reports the use to IBM. Features include:

- Collection of inventory data, and extraction of information on products used and installed at each site.
- Storage and maintenance of software procurement and contract information for products purchased by different customers or independent organizations, together with license terms and conditions.

- Comparison of procured, installed and used licenses, and reconciliation of the information derived from these three different views.
- Compilation of metering and inventory results to identify the license entitlement needed for each product and procure the required number of licenses.
- Supervision that software products are used in accordance with the agreements between customers and vendors.

System Manager

The System Manager interface is derived from the Eclipse platform—an open-source integrated development environment for the creation of tools. The Eclipse platform provides tools developers with a development kit and runtime that enables the developer to write plug-ins that allow the user to work with a particular type of resource.

Note: System Manager is installed as a plug-in that can be used in IBM-branded versions of the Eclipse platform.

You access System Manager as an Eclipse-based *perspective*. The System Manager perspective provides views and editors that help you generally with using your system and accessing the tools, as well as providing an interface for deploying the integration projects that you develop. These perspectives include support for the following tasks:

- Accessing IBM WebSphere InterChange Server development tools
- Performing configuration tasks on individual modular components used in an IBM WebSphere InterChange Server system
- Working with instances of InterChange Server
- Handling groups of integration components as user projects, and deploying components to the repository
- Benchmarking the performance characteristics of a business integration system and its components

Tasks in System Manager are accessible through a menu and corresponding toolbar, as well as through the WebSphere Business Integration System Manager view. The WebSphere Business Integration System Manager view, displayed in the left-hand panel of System Manager, shows a folder-style list of libraries of modular integration components (such as business object definitions, database connection pools, and collaboration templates and objects). You can perform development and configuration tasks by right-clicking on the icon for a component in a library.

From System Manager, you can access the GUI tools for creating collaboration templates, business object definitions, maps, and relationships. Note that some development tools can also be accessed and run independently of System Manager.

See “Development tools” on page 41 for a list of some of the available development tools.

Component configuration

You can use System Manager to manipulate the modular IBM WebSphere InterChange Server components—referred to as **integration components**—that are used to implement the exchange of business data in an IBM WebSphere InterChange Server environment. These components include collaborations,

connectors, business objects, maps, relationships, and database connection pools. From System Manager, you can perform component-related configuration tasks such as:

- Configuring collaboration objects from collaboration templates, including their port bindings and the general properties with which they run in InterChange Server.
- Accessing the tool for viewing and revising the repository definitions of standard and specific connector properties, and specifying the business objects and maps that the connectors use.

System Manager and InterChange Server modes

When you are developing integration components, you can use server mode selections in System Manager to control the manner and extent to which the component definitions and configurations you have created are loaded into the repository. Server mode selections are available for the design, testing, and production stages of development. For information about using server modes, see the *Implementation Guide for WebSphere InterChange Server*.

Development tools

You can create and modify collaborations, connectors, business objects, maps, and relationships. Development tools are used for creating and configuring the system components in a development environment, before the system is made live for production. These tools are described in more detail in the *Implementation Guide for WebSphere InterChange Server*, and in the development guides for different components. Table 3 lists and describes some of the software tools available in a development environment.

Table 3. Development Tools

Audience	Tool	Description
Collaboration developers	Process Designer	A graphical tool with which you can define collaboration templates. For more information about Process Designer and how to develop collaborations, see the *Collaboration Development Guide*.
Collaboration and map developers	Test Environment and Virtual Test Connector	Provides an environment in which you can test business integration interfaces you have developed. Provides graphical interfaces to emulate connectors, start the required components, and examine business object data. For more information about Test Environment and how to test collaborations, maps, and connectors, see the *Implementation Guide for WebSphere InterChange Server*.
Connector developers	Connector Configurator	Used for adding application-specific properties to a connector definition, for setting property values, and for configuring the connector definition with its business objects and maps. For more information about Connector Configurator and how to develop connectors, see the *Connector Development Guide for Java* or the *Connector Development Guide for C++*.
Map developers	Map Designer	A graphical tool that specifies data transformations. For more information about Map Designer and how to develop maps, see the *Map Development Guide*.
Map developers	Relationship Designer	A graphical tool that defines relationships between types of objects. These relationships are important in mapping, for example, to specify the relationship between one type of business object and another. For more information about Relationship Designer and how to develop relationships, see the *Map Development Guide*

Table 3. Development Tools (continued)

Audience	Tool	Description
Business Object Developers	Business Object Designer	A forms-based interface used for creating business object definitions both manually and from Object Discovery Agents (ODAs). For more information about Business Object Designer and how to design business object definitions, see the *Business Object Development Guide*.
Business Object Developers	Object Discovery Agent Development Kit (ODK)	An API that facilitates creation of Object Discovery Agents (ODAs). ODAs identify business object requirements specific to a data source and to generate definitions from those requirements. For more information about Business Object Designer and how to develop Object Discovery Agents (ODAs), see the *Business Object Development Guide*
All Developers	Benchmarking wizard	The benchmarking tool enables you to test various IBM WebSphere components, interfaces, and systems to measure their throughput. The benchmarking tool is accessed through System Manager. For more information about the Benchmarking wizard, see the *Benchmarking Guide*.

Administrative tools

System Manager provides access to some administrative tools, including access to System Monitor for managing the states of integration components. For more information about administrative tools and tasks, see the *System Administration Guide.*

Chapter 3. Collaborations

In an InterChange Server implementation, the term **collaborations** refers to software modules that contain code and business process logic that drives interactions between applications. A collaboration can be simple, consisting of just a few steps, or complex, involving several steps and other collaborations.

Collaborations can be distributed across any number of applications, can handle synchronous and asynchronous service calls, and can support long-lived business processes. This chapter describes collaborations, what they consist of, and how you interact with them.

This chapter is an overview of collaborations and the components you use for viewing, modifying, and creating collaborations. It contains the following sections:

For detailed instructions about using the IBM WebSphere InterChange Server collaboration tool to create collaboration templates, refer to the *Collaboration Development Guide.*

Collaboration templates and objects

When you install a collaboration, you install a **collaboration template**. A collaboration template contains all of the collaboration's execution logic, but it is not executable. To execute a collaboration, you must first create a **collaboration object** from the template. The collaboration object becomes executable after you configure it by binding it to connectors or to other collaboration objects, and by specifying other configuration properties.

Note: The toolset available with InterChange Server includes a collaboration design tool called **Process Designer**, with which you can customize and create collaboration templates.

You can use the same collaboration template to create multiple collaboration objects. For example, suppose that you are using a Contact Manager collaboration to integrate front office and back office applications. If one department of the company uses Clarify and another department uses Siebel front-office applications, and both use SAP back-office applications, you can create two collaboration objects from the ContactManager collaboration template.

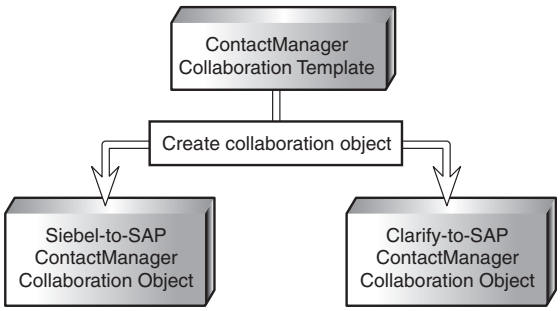

Figure 17. Creating collaboration objects from a template

You then can configure each collaboration object so that both are bound to different connectors for their source application but they are each bound to the same connector for their destination application.

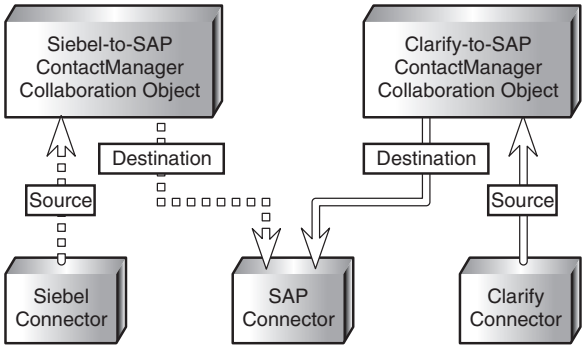

Figure 18. Configuring collaboration objects

Note: This manual uses the term "collaboration" when describing behavior, design, and features. It uses "collaboration template" and "collaboration object" only when the distinction is important, usually in reference to the configuration process or the repository.

Collaboration processing

A collaboration begins a processing flow when it is triggered to do so by the arrival of a business object. The trigger can be any of the following:

- A business object from an event published by a connector in a publish-and-subscribe interaction (described in Chapter 1)
- An access request received through the Server Access Interface (described in Chapter 1)
- A service call from a connector, described later in this chapter

Service call handling and long-lived business processes

Collaborations have the capability to interact with connectors in a manner that persists the context of a service call data flow for a specified period of time, without requiring the originating service call process to wait synchronously. If an appropriate response to the service call is received with the specified time period,

the data flow is resumed and processing continues. This feature is referred to as **long-lived business processes** for collaborations, and can be enabled during the creation of the collaboration template.

Collaborations in an InterChange Server implementation can support the following types of service calls:

- Synchronous outbound from the collaboration
- Asynchronous outbound from the collaboration
- Asynchronous inbound to the collaboration

Synchronous outbound service call

The synchronous outbound service call uses a synchronous request/response mechanism. The service call sends the request but does not complete until the response arrives and is processed.

Synchronous service calls support compensation. In addition, they support a time-out value for long-lived business processes.

Asynchronous outbound service call

An asynchronous outbound service call sends a request from the collaboration but does not expect or wait for a response before continuing its processing.

Asynchronous outbound service calls support compensation, but do not support a time-out value for long-lived business processes.

Asynchronous inbound service call

An asynchronous inbound service call waits to receive an incoming event and is used in conjunction with long-lived business processes. When an asynchronous inbound service call is created, it is given a time-out value; if the service call does not receive an incoming event before the timeout expires, an exception is raised.

Asynchronous inbound service calls are available only if the Long Lived Business Process Support option of the collaboration template has been enabled. The feature is used only for exchanges with connectors

Asynchronous inbound service calls do not support compensation.

Collaborations and concurrent processing

A collaboration begins a processing flow when it is triggered to do so by the arrival of a business object. By default, the collaboration completes the processing of one flow before it processes the flow for the next triggering business object.

However, you can configure a collaboration to process multiple concurrent event-triggered flows. If a collaboration is configured in this way, it will identify any flows that have business object dependencies and process them in order, but it will allow flows that do not have conflicts to process concurrently.

Collaboration groups

A collaboration object can be bound to another collaboration object to form a **collaboration group**. A collaboration group harnesses the power of multiple discrete collaborations to integrate related business processes.

The relationships between collaborations in a collaboration group are restricted only by the fact that one collaboration supplies business objects to the other. Collaboration groups can be connected in any number of topologies, such as chains or even webs.

Collaboration groups are also useful for providing data isolation. For details, see the *Collaboration Development Guide*.

Ports

A collaboration has a set of interfaces to the outside world called **ports**. In a collaboration template, each port is a variable that represents a business object that the collaboration object receives or produces at runtime. For example, consider a hypothetical Service Billing collaboration that is triggered by the closing of a case in a customer service management application. The collaboration retrieves the customer's contract as a business object from a customer records application, then uses the information in the contract to send an invoice business object to an accounting application.

Such a collaboration might use three ports, one for each of the connectors with which it interacts. Each port is associated with a particular type of business object, as shown in Figure 2-3:

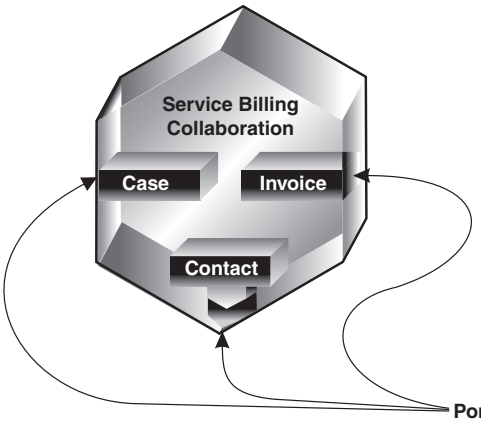

Figure 19. Ports in the Service Billing collaboration

At configuration time, an administrator creates a collaboration object that contains the template's ports. The administrator **binds** the ports for that specific collaboration object, associating each port with a connector, or with another collaboration object. For some collaborations (those whose templates accept the Retrieve verb), a port can be configured instead as *External*, so that it can receive a triggering business object from an access request—an external call on the Server Access Interface.

The ports enable communication between bound entities, so that the collaboration object can accept the business object that triggers its business processes, and then send and receive business objects as requests and responses.

Figure 20 illustrates a fictional collaboration, with its triggering port bound to accept the Case business object as a call from external processes, and its other ports bound to connectors that use Contract and Invoice business objects.

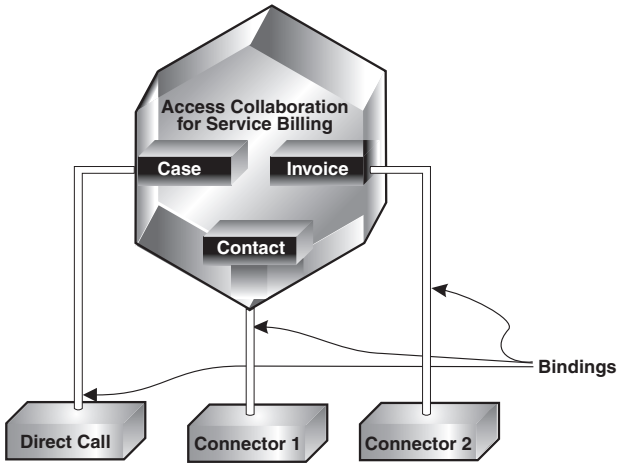

Figure 20. Ports bound to connectors

To make an analogy, a computer has various hardware interfaces. To hook up a computer with peripherals such as speakers or a monitor, you attach cables and wires whose shapes are complementary to the shapes of the computer's interfaces. The "shape" of a collaboration port is the type of business object that can pass through it. An administrator can bind a port only to a connector that supports specific types of business objects, just as you can plug a computer cable only into the interface that matches its shape.

A collaboration can be bound to another collaboration if each has a port that supports the same type of business object. Figure 21 illustrates two collaboration objects bound at ports that support the Item business object.

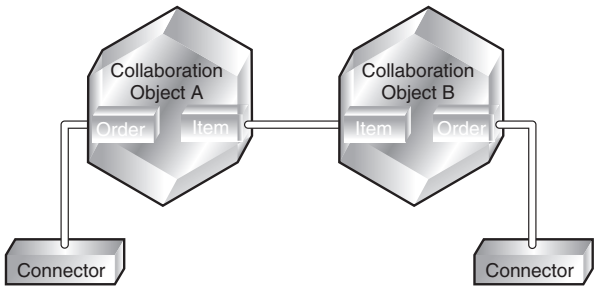

Figure 21. Binding two collaboration objects

Port names often indicate the role of the port's supported business object in the collaboration. For example, if a collaboration receives customer information and generates contract information, its ports might be called InCustomer and OutContract.

Dynamic service calls

Collaborations implement service calls with explicit binding, which means that the collaboration service call needs be configured to connect to a known destination, such as a connector or another collaboration, while the collaboration is being developed from a collaboration template. The explicit binding is static, which means that if the port assignments are changed, the collaboration objects need to be redeployed into InterChange Server before they can be used. Collaborations also cannot be deployed until all ports are bound.

InterChange Server includes an API which enables dynamic service calls, so that a collaboration can make a service call to a destination that has not been pre-defined and explicitly bound during collaboration development. The dynamic service call API captures all the required details about the consumer so that a collaboration can interact with many connectors, and the number and type of connectors can change dynamically. When creating a collaboration to receive a dynamic service call, an inbound port should be bound as an external port during port binding in the System Manager tool, so that the collaboration can receive the triggering business object through the external port. The API can pass the essential information about the client, such as the client's name and type. The API can also use the source adapter information to determine where to send the service request. The dynamic service call API is synchronous.

The dynamic service call API enables dynamic service binding without the need for any tools. The API and the associated set of parameters can be used within collaboration templates with minimal changes or influences to existing collaborations. The API itself does not handle exceptions or define failure behavior, so these issues need to be addressed in the collaboration template. The API also does not provide quality of service, such as transaction support compensation. See the *Collaboration Development Guide* for API information.

The dynamic service call has limited or no support for the following:

- Dynamic service calls from one collaboration to another do not form a collaboration group, since there is no explicit port binding between the collaborations.
- Dynamic service calls do not support compensation, so there are no rollback semantics for dynamic service calls.
- An LLBP collaboration can only make a dynamic service call to a connector, and it must use JMS as the transport type. An LLBP collaboration cannot receive a dynamic service call, because an LLBP collaboration cannot bind to an external port.

Scenarios

Inside a collaboration, one or more **scenarios** contain the "code" that implements a business process. Each scenario specifies what happens in response to the arrival of particular types of business objects that represent particular types of events. Scenarios contain all of the processing work of a collaboration.

The relationship between collaborations and scenarios is similar to the relationship between traditional programs and routines. Routines enable a programmer to break up the logic of a program in any number of ways. For example, a program can contain one large routine that handles various input arguments or several routines, each of which handles a different input argument.

Similarly, a collaboration developer can use one scenario or multiple scenarios to perform the work of a collaboration. A collaboration that synchronizes employees, for example, could be designed in either of the following ways:

- One scenario handles Employee.Create, Employee.Delete, and Employee.Update event notifications.
- One scenario handles Employee.Create, one handles Employee.Delete, and one handles Employee.Update.

Figure 22 illustrates these two approaches.

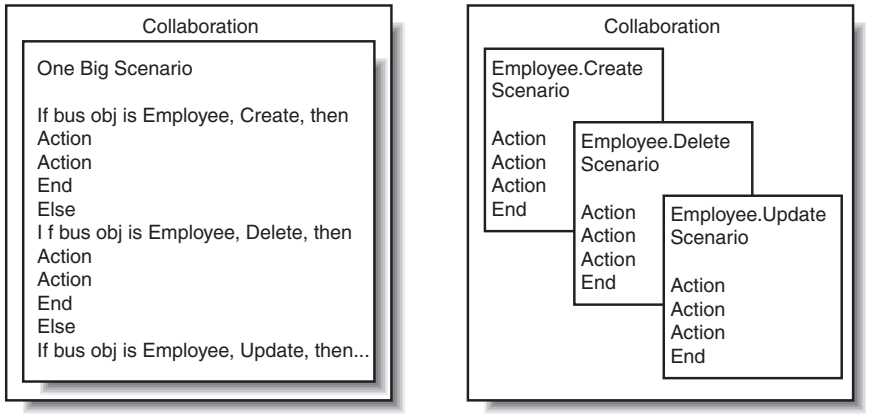

Figure 22. Scenarios in Collaborations

Although scenarios execute independently of each other, you do not independently configure and manage each. You configure the collaboration, and all of the collaboration's scenarios inherit those settings.

Business process logic

Scenarios contain the collaboration's business process logic. A highly simplified example, shown in Figure 23, illustrates the logic in a fictional scenario. The collaboration replicates changes to the employee information in a source application by moving them to a destination application. This scenario specifically handles the Create verb.

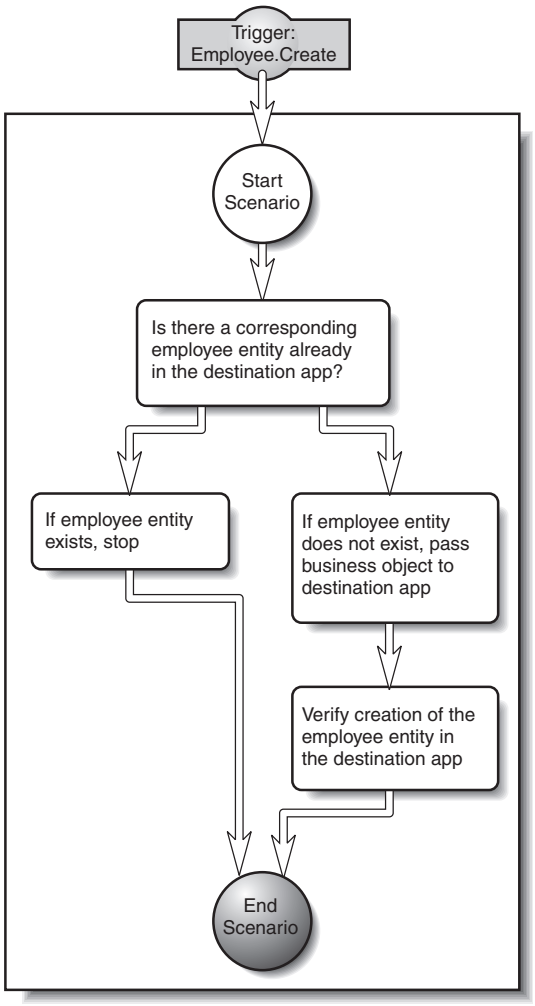

Figure 23. High-level view of a scenario

In Figure 23, the scenario's trigger is the Employee.Create event notification business object. When the Employee.Create business object arrives, the scenario starts. First, the scenario checks to see whether the employee information already exists in the destination application.

If the employee information already exists in the destination application, the scenario stops. If the employee information does not yet exist, the scenario passes the Employee.Create request business object to the connector for the destination application. A verification step ensures that the operation is successful.

The business process logic in an actual scenario typically consists of many more steps, each of which implements an action by the collaboration. These are some examples of the types of actions that a collaboration might contain:

- Get the type, attribute values, or verb of a received business object
- Create a new business object, either without values or by cloning an existing business object
- Compare an attribute value with a constant or with another attribute value
- Compare two business objects to see whether they are equal
- Send a business object to a connector or another collaboration to request an operation; process the result of the operation

- Get the value of one of the collaboration's configurable properties
- Log informational, warning, or error messages

Interactions with connectors and applications

Through its ports, a collaboration interacts with connectors, with the Access Interface, and with other collaborations. The collaboration receives its trigger, sends requests, and gets responses through these ports. These are translated to and from application operations.

Using the scenario shown in Figure 23 as a simple example, Figure 24 shows how the collaboration drives operations throughout the in a publish-and-subscribe interaction. Figure 24 shows only a single execution path, the path that is taken if the destination application does not yet contain information about the employee. The shaded arrow paths in Figure 24 show the translations that Connector B makes as it transfers collaboration requests into requests to the application API and transfers application responses into its own responses to the collaboration.

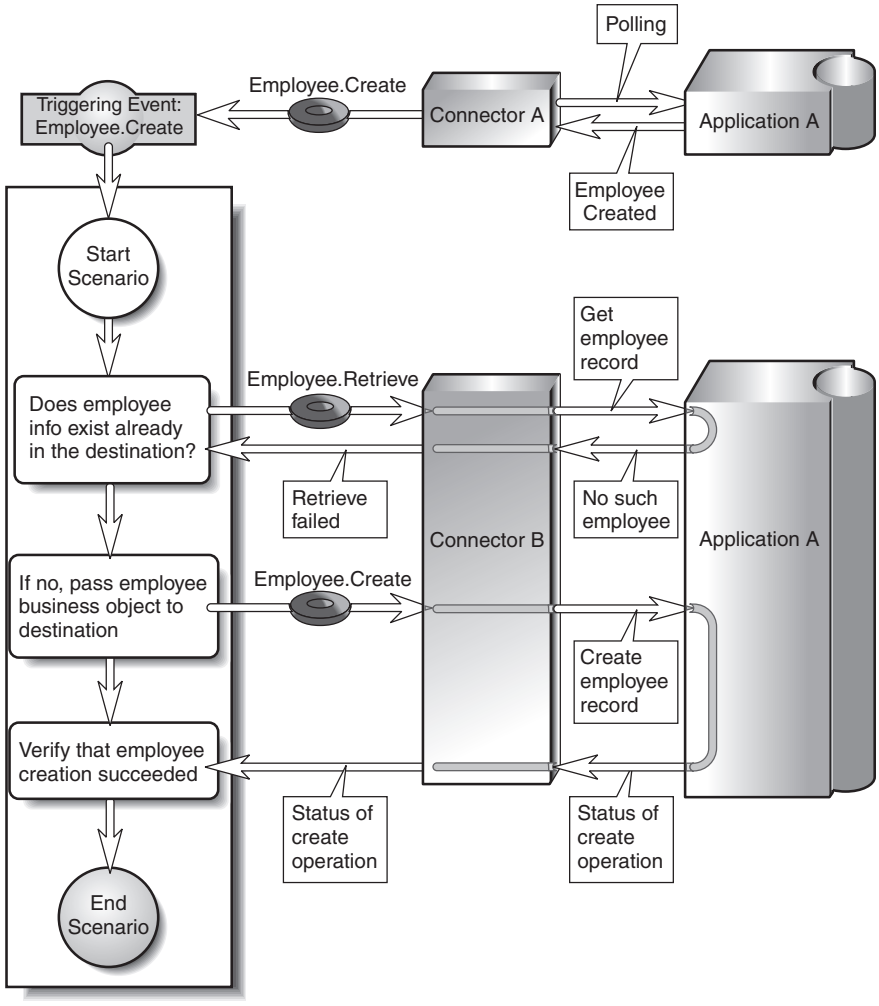

Figure 24. Processing at runtime

In this figure, the representation of a connector includes both the connector agent and connector controller functions.

Figure 24 illustrates the following process:

1. When Application A creates a new employee record, Connector A retrieves it.
2. Connector A creates an Employee.Create business object and sends this event notification to the collaboration, where its arrival triggers the start of this scenario.
3. The scenario checks to see whether the employee information is really new by trying to retrieve information on the same employee from the destination application. To do so, the scenario sends a Employee.Retrieve request to Connector B.
4. Connector B tries to retrieve the employee information, but learns that the employee does not exist in Application B. The connector returns a failure status to the collaboration.
5. Now the scenario proceeds to send the Employee.Create request. When Connector B receives the Employee.Create request, it uses Application B's API to create the new employee.
6. Connector B returns a success status to indicate that the operation succeeded.
7. The scenario receives the success status and ends.

Collaboration startup

At design time, the collaboration developer specifies the business objects whose receipt by the collaboration triggers the execution of each scenario. When you configure and enable a collaboration, the collaboration starts up. The collaboration subscribes to all of the business objects that trigger its scenarios, and then it waits.

When a connector agent sends its controller one of the subscribed-to business objects:

1. The connector controller passes the business object to the collaboration.
2. The collaboration invokes the appropriate scenario, which begins executing.

Once you have configured and enabled a collaboration, you do not need to enable it again unless you explicitly disable it. If InterChange Server shuts down and restarts, it also restarts all previously active collaborations.

Summary

This chapter described some of the features of collaborations. The key points to remember are:

- You create collaboration objects from collaboration templates and bind them to connectors and to other collaboration objects to make them executable.
- Collaborations represent business objects indirectly, through ports. You configure a collaboration object by binding each of its ports to a business object definition and to either a connector, or another collaboration object, or to external calls.
- Scenarios implement the business process logic of collaborations. Scenarios are event handlers that execute upon the arrival of business objects. As scenarios execute, they both send business objects to connectors and receive business objects from connectors.
- To enable a collaboration to execute, you configure and start it. The collaboration subscribes to the types of business objects that can trigger its scenarios. When one of them arrives, the collaboration executes the appropriate scenario.

The next chapter takes a closer look at business objects.

Chapter 4. Business objects

Business objects carry the meaning of business processes from one application to another, traversing collaborations and connectors with both data and action requests. This chapter looks at the structure and components of business objects. It contains the following sections:

Business object definitions and business objects

Chapter 1, "Overview of IBM WebSphere InterChange Server," on page 1, introduced business objects without distinguishing between business object definitions and instances of the business objects themselves. Let's look at that distinction now:

- A **business object definition** specifies the types and order of information in each entity that IBM WebSphere InterChange Server handles, and the verbs that it supports. The InterChange Server repository stores business object definitions.
- A **business object** is an instance of the definition, containing actual data. Business objects are created at runtime and not stored in the repository.

Figure 25 illustrates the relationship between business object definition and business object.

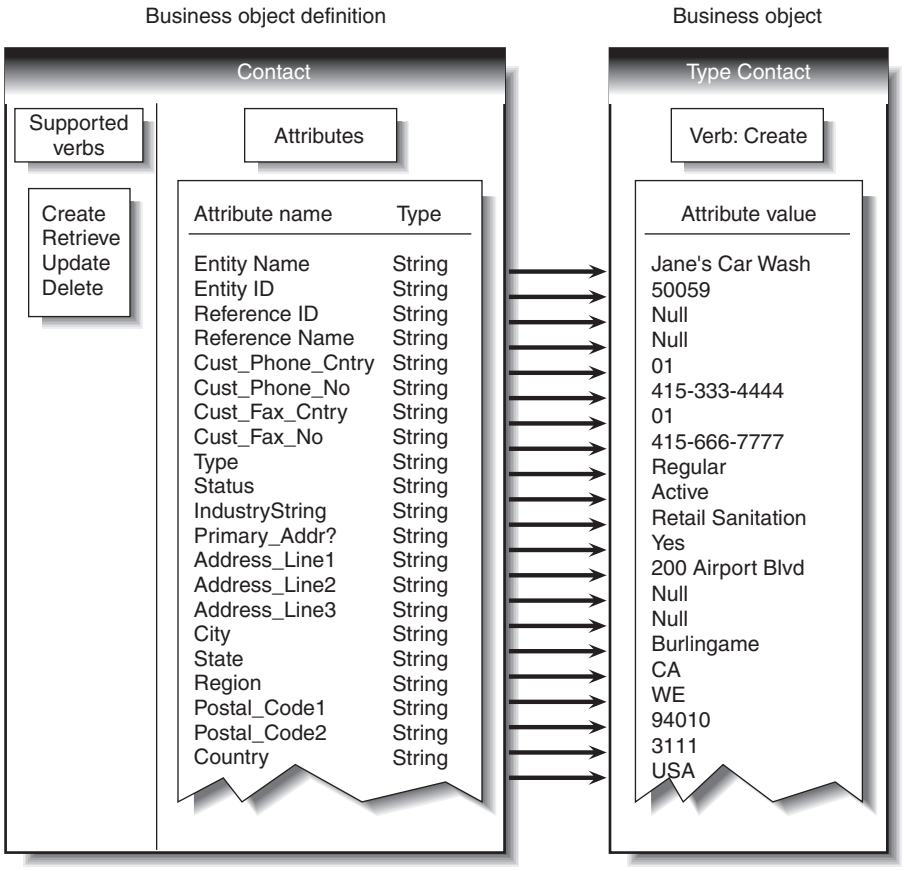

Figure 25. Business object definition and business object

Both connector agents and collaboration scenarios use business object definitions to create business objects.

Components of a business object definition

Chapter 1, "Overview of IBM WebSphere InterChange Server," on page 1, described the components of a business object in somewhat simplified terms as a type, attribute values, and verbs. This section goes into more detail about the major components of a business object definition.

Overall, a business object definition is identified by its name. The name indicates the business object definition type, such as Customer, VantiveCase, or Invoice. An application-specific business object can also have application-specific information that helps the connector agent process it. All business objects also contain attributes and verbs, as the next sections describe.

Attributes

Attributes in a business object definition describe the values connected with the entity, such as Last Name, Employee ID, Case Number, Amount, or Date Initiated. At runtime, attributes are filled in with actual data.

For example, an Employee business object definition might contain attributes for the employee's name, address, employee ID, and other relevant information. The attributes of a business object are analogous to the fields of a form or columns in a database table.

An attribute can also refer to a child business object or to an array of child business objects, such as an array of line items in a contract or part references in an invoice.

ObjectEventId attribute

The ObjectEventId attribute is a required attribute and is the last attribute in every business object. When a connector publishes an event, it uses the ObjectEventId attribute of the business object definition to store a unique value that identifies the specific business object instance that is being created.

The value of the ObjectEventId attribute is generated and handled by the system, which uses it to identify and track the flow of the specific event through the system. The developer should not map the ObjectEventId attribute or populate it through the use of a connector agent or data handler.

Simple and compound attribute types

If an attribute's type is a basic data type, such as String, Boolean, Double, Float, or Integer, the attribute value is a discrete piece of data, such as the value of a field in a database. Such an attribute is often called a **simple attribute**. Examples include LastName, CustomerID, PartNumber, AssignedTo, and Price.

If an attribute's type is the name of another business object definition (a compound type), the attribute value is a child business object or an array of child business objects. Such an attribute is called a **compound attribute**. Examples include Customer, Contract, and Oracle_Contact.

Attribute properties

A number of **properties** define the value that the attribute represents. Without showing all possible properties, Figure 26 illustrates the place of attribute properties in a business object definition.

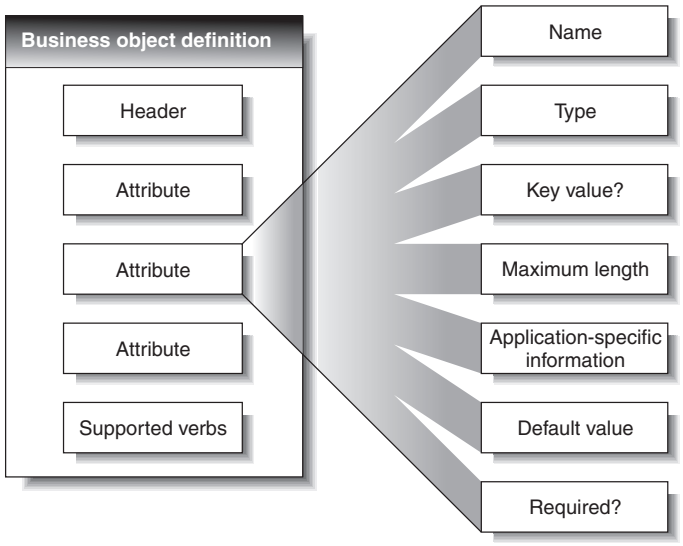

Figure 26. Attribute properties

The set of properties for a particular attribute depends on whether the attribute type is basic or compound; that is, an attribute's properties differ depending on whether the attribute refers to a single unit of data or to a child business object.

Verbs

Verbs indicate actions on the data in the business object. A business object definition contains a list of verbs; a business object contains only one verb. The most common verbs associated with business object definitions are Create, Retrieve, Update, and Delete. The meaning of a verb differs according to the role of the business object. The verb can describe an application event, make a call, make a request, or identify the result of a previous request.

Note: Some applications do not support requests for hard deletes. For such applications, the IBM WebSphere InterChange Server system performs the equivalent logical deletion, which is usually an update to inactive status. Furthermore, even if an application supports hard deletes, you can configure the IBM WebSphere InterChange Server system so that it converts Delete verbs to Update verbs when sending requests to that application.

A closer look at application-specific business objects

An application-specific business object contains the data that a connector agent moves into and out of a particular application. Therefore, each application-specific business object definition reflects the application's data model and the connector agent's access method.

Even when two application-specific business objects refer to similar application entities, differences appear in the way that attributes are organized and in the application-specific information for them.

Attribute organization

Applications often organize the same information in different ways. For example, Application A stores a telephone number and fax number for a contact in four fields, but Application B stores the same numbers in two fields.

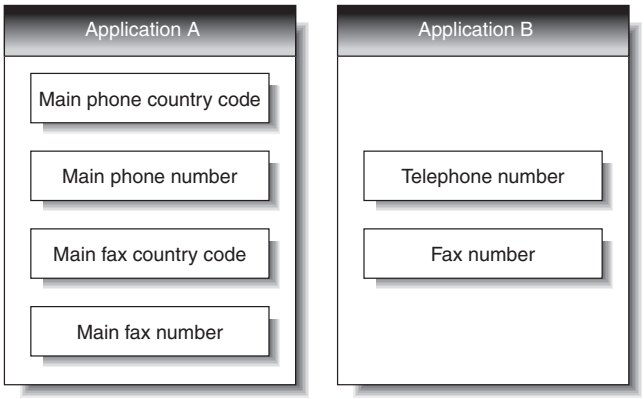

Figure 27. Telephone data in two applications

The business object definitions for the Application A business object and the Application B business object have different attributes to reflect this difference.

Application-specific information

Application-specific business objects also differ because each can optionally contain built-in processing instructions for its connector agent, called application-specific information. Application-specific information can consist of any information that the connector agent needs to process the business object.

A business object definition can have application-specific information that applies to the entire business object, to each attribute, and to each verb. At each place where application-specific information appears in a business object definition, it provides information that the connector uses in its interactions with the application.

Application-specific information for a business object

Application-specific information for the business object provides information that the connector agent uses when processing the business object as a whole.

Application-specific information for an attribute

Often, application-specific information that applies to an attribute identifies the attribute value's location in the application. The connector agent uses this identifier when building API calls to the application to retrieve or enter the attribute value.

Application-specific information takes different forms for different applications. Sometimes the connector agent can reference the attribute location by means of the application's form and field names; other times the reference is more complex.

Table 4 illustrates two examples of application-specific information for a Customer business object.

Table 4. Application-specific information for sample attributes

Attribute	Application-specific information for sample Application 1	Application-specific information for sample Application 2
Customer ID	CUST:CID	cust:site_id::3::
Customer name	CUST:CNAM	cust:name::3::
Status	CUST:CSTAT	cust:status::1:0:Status
Industry	CUST:CIND	cust:industry_type::3::Industry

The values have the following meanings:

- The Sample Application 1 value specifies a field in the CUST table.
- The Sample Application 2 value specifies a table, field, and relation in the application database, using the following format:

  table:*field*:*relation*:*type*:*default-value*

  where *type* is 1 (int), 2 (float), or 3 (string)

In exceptional cases, application-specific information for attributes is unnecessary. For example, some applications provide very direct and easy to use designations for units of data. Imagine that an application identifies sample fields as Table 5 illustrates.

Table 5. Sample application identifiers

Attribute	Application's identifier for the field containing the value
Customer ID	XCustomerID
Customer name	XCustomerName
Status	XStatus
Industry	XIndustry

In the example that Table 5 illustrates, it is easy for the connector agent to associate an attribute with its identifier in the application because the rules for conversion

are so regular: add the X or subtract the X. Therefore, the attributes in business objects for this application may not need application-specific information.

Application-specific information for verbs

A business object definition can include application-specific information for each verb that it supports. The application-specific information instructs the connector agent how to process the business object when that verb is active.

Modification options

You can modify either generic or application-specific business object definitions, and might need to modify both. For example, to change a collaboration so that it processes some additional application data, you must modify both types of business object definitions, adding attributes so that each business object carries that data. If you modify a business object definition, you must also modify mapping components.

Summary

This chapter described the definition and use of business objects in the IBM WebSphere InterChange Server system. The key points to remember are:

- Business object definitions are specifications that define the types and order of application data and the set of verbs that can trigger operations on the data. A business object is an instance of a definition, containing actual data and a verb.
- Application-specific business objects for the same types of entities differ because application models differ. The differences are apparent mostly in the order and number of attributes and the application specific information.
- Business object definitions contain verbs and attributes. Attributes with basic data types refer to data; attributes with compound data types refer to child business objects.
- Inside a business object definition, application-specific information contains relevant application data. A connector agent uses application-specific information to drive its interactions with the application.
- You can modify both application-specific and generic business objects if you need to modify the data flow at a site.

Chapter 5. Connectors

A **connector** mediates between an application (or other programmatic entity, such as a Web server) and one or more collaborations. A connector can be specific to an application—such as SAP R/3, version 4—or to a technology, such as a data format or protocol (XML or EDI). A connector's communications with collaborations can take two forms:

- Application **event notifications**, which connectors for specific applications pass to collaborations, and
- **Request processing**, which connectors perform on behalf of collaborations.

Connectors can be configured for applications that reside on the local network, and for remote applications that reside across an Internet firewall.

Most connectors share certain common behaviors, differing only in the manner in which they interact with applications and with application-specific business objects. This chapter is an introduction to both the common behavior of connectors and to the areas in which they differ. It contains the following sections:

Connectors for many applications and technologies are available as part of the WebSphere Business Integration Adapters product. A WebSphere Business Integration Adapter includes both a connector and its message files and configuration tool, and it may also include mechanisms for creating and handling business objects specific to that connector. If you need to create a custom connector or to modify an existing connector, however, you will need detailed knowledge of connector behavior. For more information on how to create a custom connector, see the *Connector Development Guide for Java* or the *Connector Development Guide for C++*.

Connector startup

In an InterChange Server system, a connector has the following components, each of which must be started for the connector to run:

- A **connector controller** starts up with the InterChange Server; if the connector has been configured, the InterChange Server automatically creates a controller for it.
- A **connector agent** must be explicitly started, using a command line or shortcut. However, you can optionally configure a connector agent to restart automatically after an abnormal shutdown or failure.

A connector agent communicates with a particular InterChange Server at startup and throughout the time that the connector agent executes. When you start a connector agent, you supply the name of an InterChange Server. The first thing

that the connector agent does is contact InterChange Server, to requests its own configuration settings and the business object definitions that it is configured to support.

An InterChange Server need not be running when you start a connector agent. However, the connector agent cannot perform any useful work until it obtains the information it needs to initialize. If you start a connector agent when its InterChange Server is not running, the connector agent repeatedly attempts to contact the server.

Before a connector agent can connect to an application, it must receive its configuration and download its business object definitions from the InterChange Server repository. It also obtains a list of collaboration event subscriptions that it must support.

The following figure illustrates the connector agent's three-step startup.

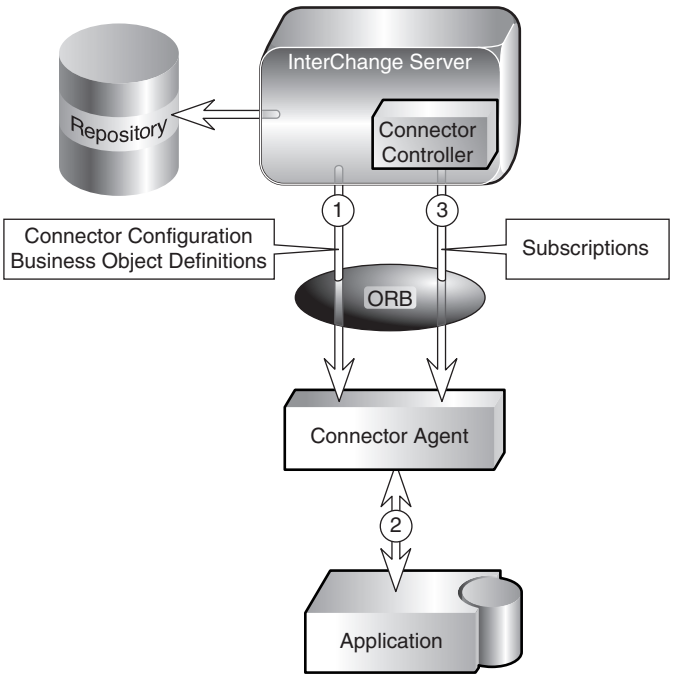

Figure 28. Connector startup

The connector agent performs its startup tasks in this sequence:

1. Contacts the InterChange Server for configuration information and business object definitions.
2. Connects to the application.
3. Contacts the connector controller for a list of subscriptions.

A connector downloads its business object definitions and certain of its configuration properties, such as its application user name and password, only at startup time. The values of some other properties, such as the property that controls connector tracing, can be changed dynamically.

Event notification

Many (but not all) connectors engage in a publish-and-subscribe interaction with collaborations and use event-notification to trigger events in those collaborations. A connector whose application provides triggering events for InterChange Server collaborations must learn about those events and send the associated data to the subscribing collaboration. The following figure illustrates a connector's interactions with respect to event notification.

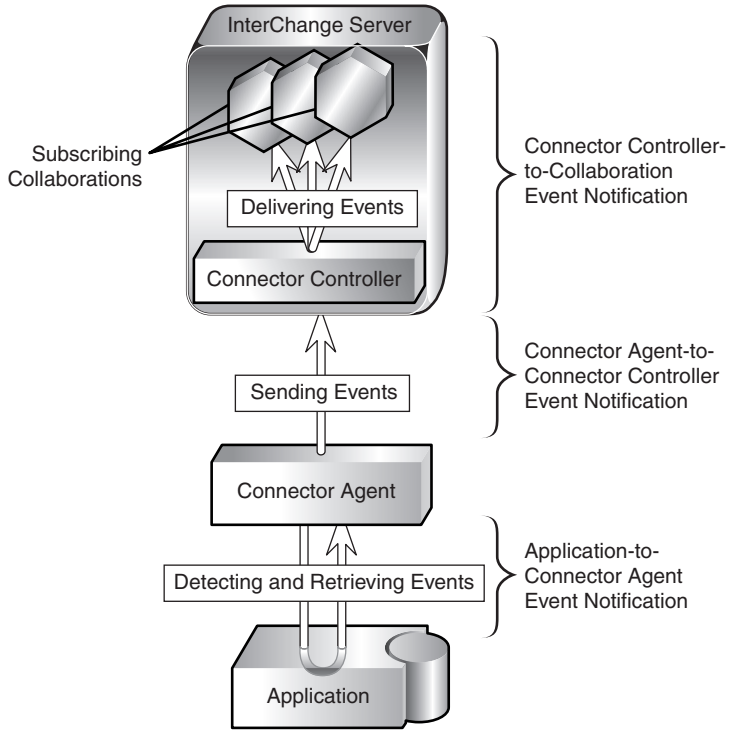

Figure 29. Event notification in a connector

The ways in which connector agents detect and retrieve events differ from one connector to another. However, the way in which connector agents send events to connector controllers, and the way in which connector controllers deliver those events to collaborations, is standard across all connectors.

The following subsections describe general concepts regarding the operation of most connectors, including:

- How connectors use application event notification mechanisms, and
- How connectors detect and process events.

This discussion is not intended to describe the specific implementation of any particular connector.

Setting up the application's event-notification mechanism

To a connector, an **application event** is any operation that affects the data of an application entity that is associated with a business object definition. There are other types of events in applications; for example, a mouse click is an event to an application's window system or forms interface. The connector, however, is

interested only in data-level events that create, update, delete, or otherwise affect the content of the application's data store.

Some applications explicitly trap and report events, providing user-friendly event management and configurable event text. Other applications, without a concept of discrete, reportable events, might silently update their databases when something happens.

For most connectors, an implementor needs to perform some application configuration to set up an **event notification mechanism** for the connector's use. An event notification mechanism is an ordered list of operations that take place in the application. It might have the physical form of an application event queue, an email inbox, or a database table.

What types of event notification mechanisms do connectors use? The next sections illustrate some general approaches.

When applications have event support

If an application is event-based, it probably has an event notification interface for use by client applications such as connectors. The application might also permit implementors to configure the text of the event report. For such applications, setting up the connector's event notification mechanism is a normal application setup task.

For example, imagine that an application lets you install a script that executes when a particular type of event occurs and that the script can place a notification in an event inbox. To install the connector for that application, you create a user account for the connector, write or obtain scripts for handling the events you want to track, install the scripts, specify the type of event that triggers each script, and create the inbox. When you are done, the connector agent periodically retrieves the inbox contents to check for new events.

The following figure illustrates an application configuration that includes an event inbox.

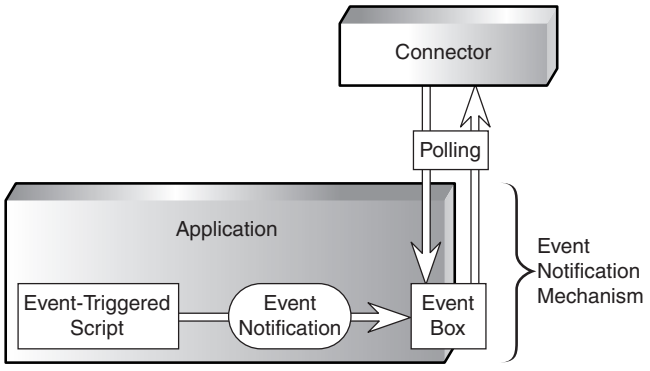

Figure 30. Example Using an event inbox for event notification

Another application might have an internal workflow system that can generate email messages or write to an event queue when a particular operation occurs. Figure 31 illustrates an application that has a business object repository where

business objects and events are defined. In the figure, Customer is an application business object and Create, Delete, and Update are the types of events associated with it.

When a business object event such as Customer.Update takes place, the event is sent to the workflow system, which places an entry in an event table in the application database.

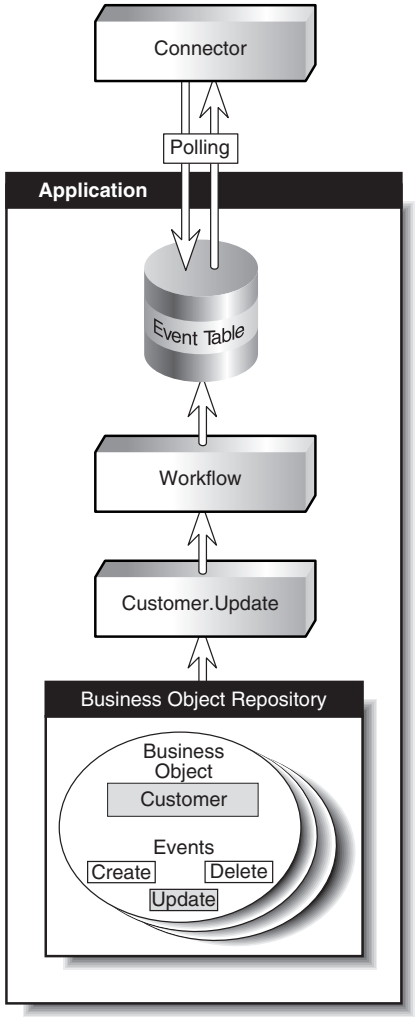

Figure 31. Example: Using application workflow for event notification

When applications lack event support

The preferred method for a connector to interact with application events is through the application's API, which provides a framework that enforces the application's data model and logic. However, some application APIs do not provide native support for event notification.

One way that a connector can receive event notifications from such an application is to interact with the application database. For example, an implementor can set up a trigger on an Employee table that detects updates to the rows. When an update occurs, the trigger inserts information about the update into a special table. Each new row that appears in this table of events represents an event notification. The connector can use SQL queries to retrieve new events from the table.

Figure 32 illustrates this approach.

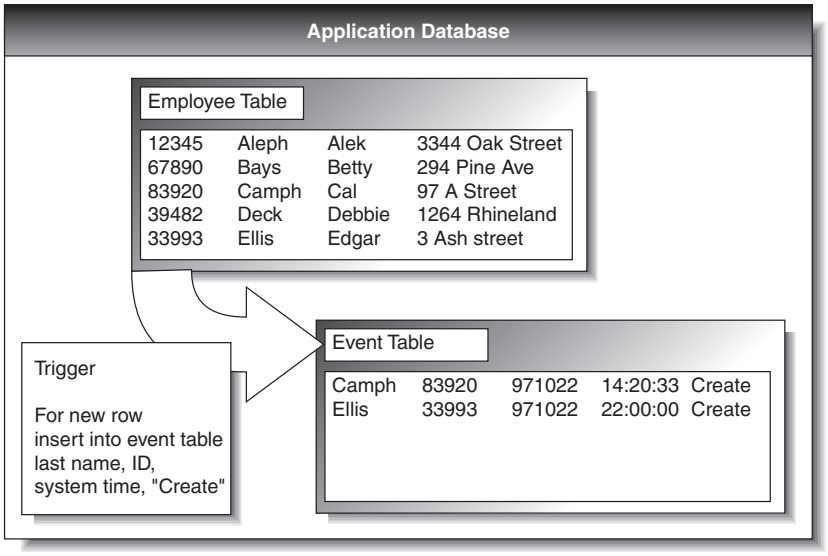

Figure 32. Example: Using the database for event notification

In Figure 32, the application database has a trigger on the creation of records in the Employee Table. Each time the application inserts a new record, the trigger creates a row in the event table. The row contains the key values of the new employee record (last name and employee ID), the system time, and the event type, Create.

Detecting an event

A connector agent learns about application events by polling for new events in the event notification mechanism. The polling method is completely application-specific, based on the event notification mechanism that the connector uses.

Polling is configurable. When you use System Manager to configure a connector, you can:

- Adjust the frequency with which the connector agent polls
- Specify the hours during which the connector agent polls

A connector agent need not poll an application if collaborations are not interested in the application's events. If a particular application participates in collaborations but is not the source of events, you can stop the connector agent from polling by setting its polling frequency to "no."

Processing an event

After detecting an event, the connector agent:

- Associates the application event with a business object definition and verb that represent the event
- Checks for subscriptions to the event
- Retrieves application data
- Sends a business object to the connector controller, if necessary
- Archives the event (optional)

Associating an application event with a business object definition

When a connector agent retrieves an event, it must determine whether any collaboration has subscribed to the event. However, because collaborations subscribe to events in the form *business-object-type.verb*, the connector must first determine which business object definition and verb represent the event.

The connector agent uses the event text to associate the event with a business object definition and verb, as Table 6 shows.

Table 6. Event Text and Business Object Formation

Type of data in the application event	Examples	Use
Application entity type	Customer, Part, Item	Determining the associated business object definition
Operation that occurred	Create, Update, Delete	Determining the active verb of the business object

For example, a connector can associate the following event text with an Employee.Create business object:

```
1997.10.19.12:50.22 employee created lname="como" id="101961"
```

The event text contains:

- A time stamp that helps to uniquely identify the event
- The application entity "employee"
- The operation "created"
- The employee's last name and ID, key values with which the connector agent can retrieve the rest of the employee information

Although this example is simple, other types of event text might require more processing by the connector agent.

Checking for subscriptions

After the connector agent associates an event with a business object definition and verb, it checks its internal list of subscriptions to find out whether any collaboration is interested in the event. The list of subscriptions is always current because the connector controller updates the connector agent whenever subscriptions change.

If the event does not match any current subscription, the connector agent logs a warning message and discards the event.

Building a business object

If there is a subscription for an event, the connector agent builds a business object, uses the key values to retrieve application data, and fills in the business object. "Business object construction and deconstruction" on page 69 describes and illustrates the process of building a business object.

Sending the business object to the connector controller

A connector agent sends a business object to the connector controller by means of the transport in use (messaging or CORBA). The connector agent knows only that there is a subscription for the business object but does not know which collaboration or collaborations are the subscribers; the connector controller takes care of delivery to the collaborations.

Archiving events

Application event archives are useful for troubleshooting and record keeping. An event archive contains status information about each event, such as:

- Successfully sent to the InterChange Server
- No subscription for the event
- Processing failed

If an application provides an event archiving feature, the connector generally uses it. A connector for an application that does not support event archiving might have its own event archive. For example, if a connector's event notification mechanism is like the database mechanism illustrated in Figure 32, a database trigger could copy deleted events to an archive table.

Request processing

Connectors can receive requests from collaborations and make requests to the application on their behalf. For example, a collaboration might send a connector a request to delete a contract, update a part, or create a customer, in the form of a Contract.Delete, Part.Update, or Customer.Create business object.

When a connector controller receives a collaboration's request, it forwards the request to the connector agent. The connector agent translates the business object into an application request—typically a set of calls to the API—and then returns the results.

Figure 33 illustrates a connector's interactions with respect to handling collaboration requests.

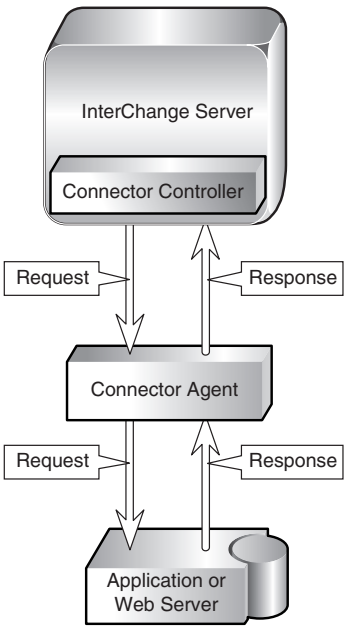

Figure 33. Connector interactions for request handling

Note: A collaboration sends a business object to a specific connector in a request/response type of interaction, unlike the publish/subscribe interactions that characterize event notification.

When a connector agent receives a request, it determines how to process the request based on three types of information:

- The verb of the business object
- Application-specific information for the verb
- Metadata that is contained in the business object definition itself and used in the construction and deconstruction of the business object

These factors are described in the topics that follow.

Verb-based processing

A connector agent reacts to the Create, Retrieve, Update, or Delete verb in a request according to the logic and API of its application. Two connector agents might handle the same type of request differently, although the result is logically the same.

For some connectors, only one method is required for performing operations on a business object, regardless of what verb the request contains. But for many connectors, each verb requires a different method.

When a connector agent receives a request, it invokes the method in the application that matches the business object's active verb. For example, when a connector agent receives an AppAEmployee.Update business object, it invokes the Update method on the AppAEmployee object. The Update method interacts with the application in order to perform the update.

Figure 34 illustrates some verb handling methods.

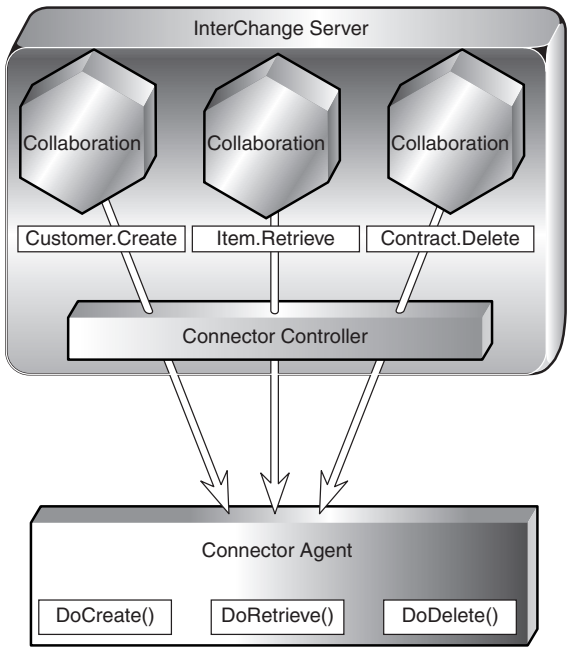

Figure 34. Processing requests

When the connector in Figure 34 receives a Customer.Create, Item.Retrieve, or Contract.Delete request, it invokes its DoCreate, DoRetrieve, or DoDelete method, respectively.

Verb-based application-specific information

In a business object definition, application-specific information provides additional input to a connector agent that is processing an instance of the business object. Each verb can have application-specific information. The format and content of the application-specific information itself is completely connector-specific.

For example, application-specific information for the Retrieve verb in a business object definition might supply special input arguments to the Retrieve method in that connector agent.

As an example, suppose that the MyApp application has three forms in which information about InventoryItem appears:

- InventoryItem-New
- InventoryItem-Change
- InventoryItem-Remove

When the MyApp connector agent performs an operation on an inventory item, it must reference the correct form for that operation. The application-specific information field associated with each verb in the InventoryItem business object definition can store the form name.

The combination of verb-specific methods and application-specific input to those methods gives a connector agent unique instructions for processing.

Concurrent processing capabilities

The multi-threaded nature of the Java-based InterChange Server makes concurrent processing possible for events that connector controllers deliver to collaborations.

Depending upon the applications and connectors being implemented, concurrent processing can also be used for requests received by connector agents. This is done either through multiple threads or through the use of parallel processes. This capability can be used for requests originating from either collaborations or access clients.

The manner in which concurrent processing can be implemented depends upon the type and design of the connector:

- If a connector agent was written in C++, it is inherently single-threaded. However, C++ connector agents can be made to run concurrent processes through the use of Connector Agent Parallelism, a feature that runs processes concurrently by instantiating multiple slave processes out of a single-threaded master process in the connector agent. This feature can be activated or deactivated through the use of configurable settings in the System Manager.
- If a connector agent was written in Java, it is inherently capable of concurrent processing through multi-threading, without use of the Connector Agent Parallelism feature.
- Some connector agents written in Java might have been designed to enforce single-threaded processing, even though the Java Connector Development Kit (JCDK) itself is multi-threaded. Typically a Java connector agent might be designed in this way if the API libraries of the application impose restrictions that require that the connector agent run only a single thread. For such connectors, the Connector Agent Parallelism feature can be used to instantiate multiple slave processes out of the single-threaded master process of the connector agent.

Note that the use of Connector Agent Parallelism requires that the application itself be able to support concurrent processing. Some applications may have architectures or processing requirements that make concurrent processing impractical.

Business object construction and deconstruction

A connector agent accomplishes its event notification and request-handling tasks by constructing and deconstructing business objects:

- When a connector agent retrieves an event that it must send to the InterChange Server, it constructs a business object that represents the event.
- When a connector agent receives a business object that represents a request from a collaboration, it deconstructs the business object to create an application request.

Business object metadata and connector actions

A connector's transformation of an application event to a business object and from a business object to an application request is driven by data definitions (**metadata**) that are defined when a business object is designed.

Connector agents and business object metadata are designed to work together. The design of a connector agent and its business objects is analogous to the design of a computer device in which certain functionality can be implemented by either the software or hardware. The developer considers performance, extensibility, and other issues to decide where to implement key features.

Similarly, the division of work between the connector agent and its business objects is a result of the connector developer's design decisions. IBM WebSphere InterChange Server design principles encourage using the business object metadata to drive connector logic, rather than hard-coding logic in connector agents.

In addition to properties that specify the types, sizes, and default values for attributes, business object definitions use application-specific fields to pass specific instructions to the connector agent on how to process the business object.

For example, recall that Chapter 4, "Business objects," on page 53, presents some examples of application-specific information for the attributes of a business object that represents a customer. Table 7 shows some of those examples.

Table 7. Sample application-specific information

Attribute	Application-specific information
Customer ID	CUST1:CID
Customer name	CUST1:CNAM
Status	CUST1:CSTAT

When processing a business object, the connector agent reads the definition and uses the application-specific information to build an application request:

- To obtain the customer ID, the connector agent obtains the value of Form CUST1, Field CID.
- To obtain the customer name, the connector agent obtains the value of Form CUST1, Field CNAM.
- To obtain the customer status, the connector agent obtains the value of Form CUST1, Field CSTAT.

Because application-specific information and other metadata in the business object definition guides the actions of a connector agent, a connector agent's behavior can be described as **metadata-driven**.

Benefits of metadata-driven connector agents

A metadata-driven connector agent is flexible, because it does not have hard-coded instructions on how to handle each type of business object that it supports. Without recoding or recompiling, the connector agent automatically supports new business object definitions, as long as they match the connector's specifications.

A metadata-driven connector agent also supports new or changed attributes within a business object definition. The connector agent processes the attributes automatically as it loops through the attributes of the business object definition.

An example of business object construction

The following process describes how a connector agent creates a business object from its definition:

1. The connector agent loops through the business object definition attribute by attribute, using application-specific information to prepare an API call or build a query to obtain the application entity.
2. The connector agent sends the request to the application and retrieves the results.
3. The connector agent loops through the results, using the value of AppSpecificInfo to determine which retrieved value represents each business object attribute.

Figure 35 is an example of a connector agent that is building a business object from the definition. The connector agent has retrieved an application event involving an item whose key value, the item number, is 123. The connector agent must build an Item business object from the business object definition, which contains four attributes: Group, Description, Price, and ItemNum.

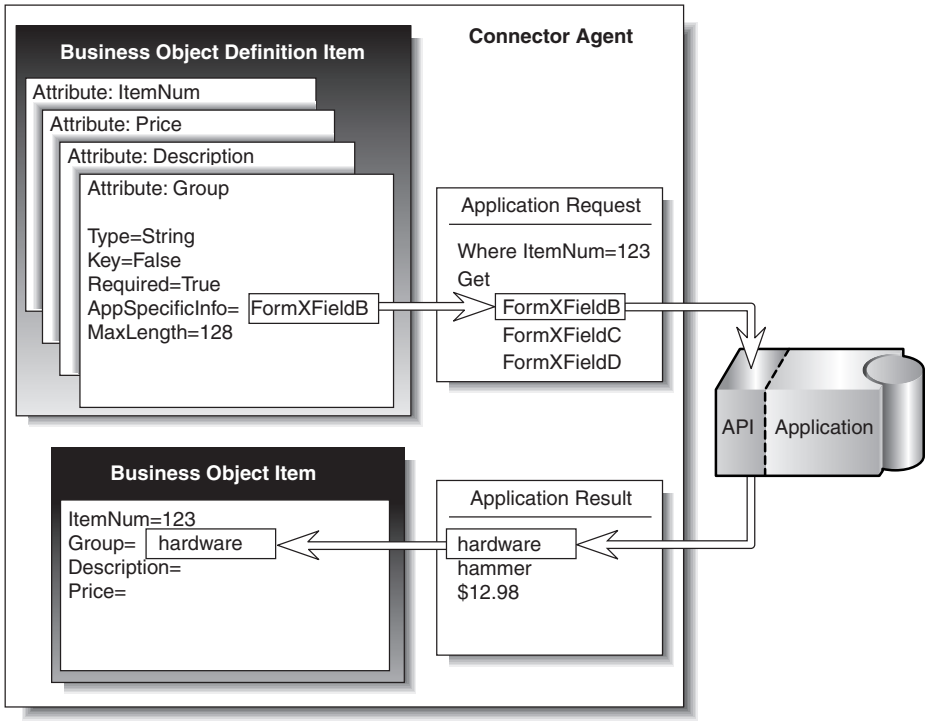

Figure 35. Building a business object in a connector

Using the item number, 123, to identify the item, the connector agent retrieves the values of the remaining attributes. Application-specific information provides the form and field identifier for the required data.

For example, FormXFieldB identifies Group data. The connector agent requests the value of Field B in Form X for item ID 123. The connector agent then uses the returned value, "hardware," to fill in the value of the business object's Group attribute.

The process of deconstruction works in the opposite way. The connector agent uses the business object definition to determine how to make an application request from the data contained in the business object that it received.

Connector configuration

Before a connector can be used, you must provide it with a connector definition, which contains the following information:

- Values of connector configuration properties for use by the connector agent and connector controller
- The business objects that the connector will support
- In some cases, specific maps for use between specified business objects

The Connector Configurator tool provides tabs to allow you to set this information.

Connector properties

There are two types of connector properties:

- **Standard properties**

Standard properties are available for *all* connectors. Standard properties include the following:

- An indication of whether the connector agent logs messages locally or sends them to the InterChange Server for inclusion in the general log file
- Location of the log file for connector agents that log locally
- Application login and password

- **Connector-specific properties**

  Connector-specific properties are typically values that a specific connector agent needs in order to establish a session with the application. These are examples of connector-specific properties for various connectors:

  - Name or IP address of the machine running the application
  - Identification of application gateway systems
  - Name of the application database
  - Name of the event inbox

Connector properties can be set using the Connector Configurator tool. With certain restrictions, connector properties can also be set in local configuration files that reside on the machine where the connector agent is installed. Some connector configuration properties can also be set at the command line.

Associated maps

The connector controller uses map references when it receives a business object that requires mapping. If data transformations are required between a particular generic and application-specific business object, you must specify the map that performs this transformation. You specify the associated maps in Connector Configurator.

Connector development

When you develop or modify a connector, you create the connector agent itself and the business object definitions it will use, and then you create a connector repository definition. Note that you do not need to create or modify a connector controller; the connector controller component is internal to InterChange Server and is instantiated by InterChange Server for each connector that you define in the repository.

Connector development involves creating the relationship between the connector and a particular application. The actual coding of a connector is usually a fairly straightforward process. The most challenging tasks are:

- Designing the application's event-notification method
- Defining application-specific business object definitions and mapping them to generic business object definitions
- Defining the relationship between the application-specific business objects and the connector

For detailed information on connector architecture, modifications, and development, refer to the *Connector Development Guide for Java* or the *Connector Development Guide for C++*.

Summary

This chapter was an overview of connectors and how they work. The key points to remember are:

- Connectors have two main roles. These are notifying collaborations about application events and carrying out application requests on behalf of collaborations.
- In its event notification role, a connector interacts with the application to detect changes in the application and process the data associated with those changes.
- In its role as the implementor of collaboration requests, a connector uses unique functions that implement each business object verb that the connector supports.
- When a connector agent constructs a business object from an application event or deconstructs a business object to create an application request, the connector agent is driven by the application-specific information and other metadata in the business object definition.

The next chapter goes into more detail about how mapping works.

Chapter 6. Data mapping

Data mapping is the process by which the IBM WebSphere InterChange Server system passes data from an application-specific business object to a generic business object or from a generic object to an application-specific business object. Mapping allows the IBM WebSphere InterChange Server system to handle differences in the data modeling of different applications.

Note: When a collaboration transfers information across two or more installations of the *same* application, mapping is *not* required unless different customizations have been done at each installation.

Data mapping occurs in the IBM WebSphere InterChange Server system at runtime, using maps that you create prior to runtime.

This chapter is an overview of the mapping process and the components you use for viewing, modifying, and creating maps and relationships. It contains the following sections:

For detailed instructions about using the IBM WebSphere InterChange Server mapping tool to create maps, refer to the *Map Development Guide.*

How the InterChange Server system uses mapping

Mapping make it possible for collaborations to take the data from a business object of one application and transform it to generate a business object for a disparate application. In the mapping process, collaborations interact with connectors, with the Access Interface, or with both.

The IBM WebSphere InterChange Server system provides comprehensive support for data mapping between business objects, including the following capabilities:

- Transforming data values from one or more attributes in a source business object to one or more attributes in a destination business object
- Establishing and maintaining relationships between data entities that are equivalent but are represented differently and cannot be directly transformed
- Enabling access to external mapping resources, such as databases for performing queries and third-party mapping products

In an IBM WebSphere InterChange Server environment, mapping typically takes place between application-specific business objects and generic business objects. The IBM WebSphere InterChange Server system does not map application-specific business objects directly to other application-specific business objects. Instead, the generic object acts as an intermediary between two application data models, carrying the mapped information from one data model to the collaboration (either through a connector or through the Server Access Interface), then carrying mapped information from the collaboration to a connector for another data model.

Figure 5-1 illustrates the way that data mapping occurs at runtime, using a fictionalized Employee Management collaboration as an example: The Employee Management collaboration receives an employee business object from the source connector, then sends an employee business object to the destination connector. (In this example, a collaboration receives a business object from a connector; a similar mapping process takes place when a collaboration receives a business object from the Access Interface.)

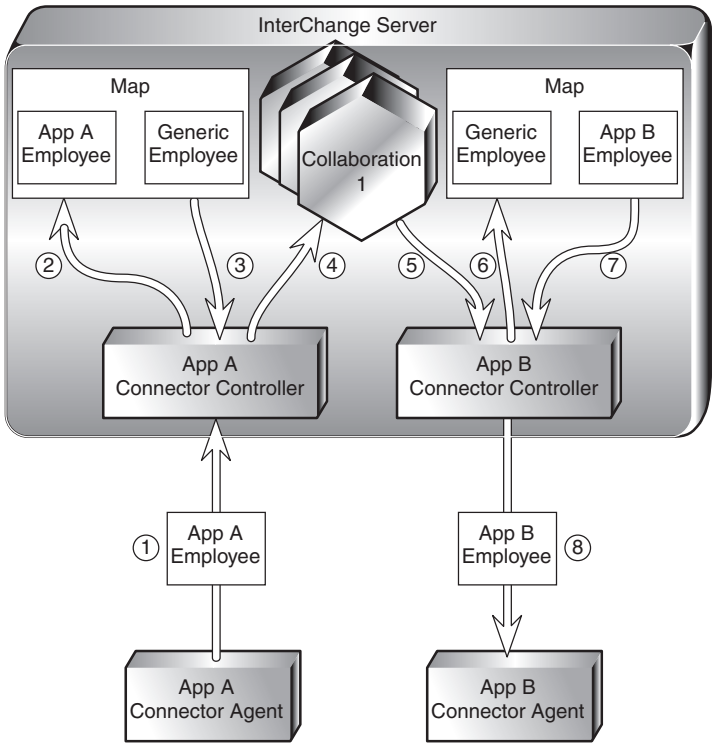

Figure 36. Data mapping at runtime

Figure 5-1 illustrates the following sequence:

1. The App A connector agent produces an App A Employee business object and sends it to the connector controller.
2. The connector controller passes the App A Employee business object to the InterChange Server for mapping. The request includes the name of the data map that the Server must use, based on the map name specified in the connector configuration.
3. The map returns the generic Employee business object to the connector controller.
4. The connector controller checks the collaborations that have subscriptions to the generic Employee business object. In this case, Collaboration1 has a subscription, so the connector controller hands the business object to Collaboration1.
5. The collaboration performs some processing and then produces another generic Employee business object as output, which it sends to the connector controller.
6. The connector controller passes the generic business object to the InterChange Server, requesting mapping to the App B Employee business object.
7. The map returns the application-specific business object to the connector controller.

8. The connector controller passes the App B business object to the App B connector agent, which can then pass the data in the business object into Application B.

The example above used two maps—one from the App A Employee business object to the generic Employee business object used by the collaboration, and one from the generic Employee business object to the App B Employee business object. The Employee data moved in only one direction—from App A toward App B.

If you wanted to exchange the Employee data in *both* directions between the two different applications, four maps would be required:

- A map from the application-specific business object of Application A to the generic business object used by the collaboration.
- A map from the generic business object to the application-specific business object of Application B.
- A map from the application-specific business object of Application B to the generic business object.
- A map from the generic business object to the application-specific business object of Application A.

Map components and tools

The IBM WebSphere InterChange Server system contains a Java mapping API, the Mapping API, that includes methods for handling common data transformation situations. Graphical design tools are provided for creating the two principal components of IBM WebSphere InterChange Server mapping—maps and relationship definitions:

- A **map** contains the Java code that specifies how to transform attributes from one or more source business objects to one or more destination business objects. You need a map for every business object that you intend to transfer between different applications. When you modify business objects, you might also need to modify the associated maps. You typically create one map for each source business object you want to transform.

  The **Map Designer** tool is used for creating and compiling maps.
- A **relationship definition** establishes an association between two or more data entities in the IBM WebSphere InterChange Server system. Relationship definitions within maps are most often used for transforming business object attributes in which the data has a similar purpose but is represented differently in each application. Most maps use one, or a few, relationship definitions.

  The **Relationship Designer** tool is used for creating relationship definitions and the table schemas that are used to store the runtime relationship instance data.

Both map and relationship definitions reside in the InterChange Server's repository. Like business object definitions, relationship definitions function as specifications or templates for the instances that are created. Unlike instances of business objects, relationship instances persist, and are stored in special tables for each relationship.

Each time the system receives a request to transform a given business object, it executes the associated map, and, depending upon the purpose of the transformation, creates one or more instances of its associated relationship definitions. Relationship instances created during map execution contain the runtime data from the attributes they associate, and this data is stored in the relationship tables.

For more information about how the mapping tools work, start with Chapter 1 of the *Map Development Guide*.

Mapping transformations

A map associates a source business object with a destination business object, and contains a series of transformation steps—one for each attribute that is being transformed. Each transformation step contains Java code that calculates the value of the attribute.

Within a data map, the conversion of source to destination attributes can be simple, or it can require establishing and maintaining relationships between data entities that are equivalent but are represented differently and cannot be directly transformed.

Simple transformations

In a simple case of data transformation, the values of the source and destination attributes have a clear correspondence and similar meanings, although the attributes might be structured differently.

Simple mapping includes actions such as these:

- Copy a source attribute value to one or more destination attribute values.
- Split a source attribute value into multiple destination attributes values.
- Join multiple source attribute values into one destination attribute value.
- Ignore a source attribute value for which the destination business object has no equivalent attribute.

Figure 5-2 is an example of simple mapping for a few of these operations:

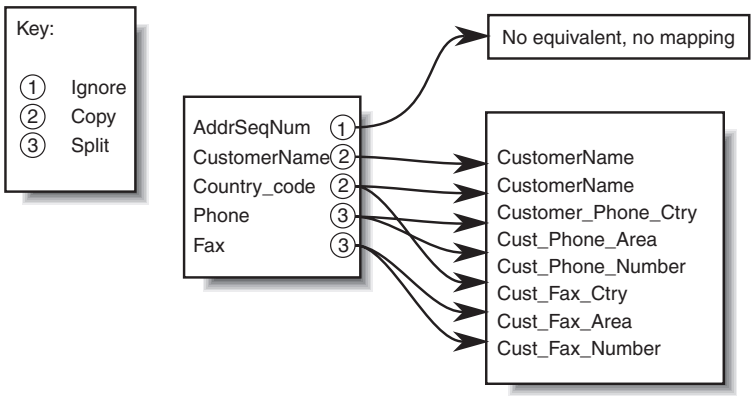

Figure 37. Simple mapping

Relationship transformations

Some attributes do not lend themselves to simple transformations. Different applications may have attributes that are equivalent, containing information for a similar purpose, but which have incompatible formats or values. For example, for a Country attribute, one application might use a two-letter code (such as US, FR, or EG) while another application uses a numeral (such as 1, 2, or 3).

To associate such attributes between different applications, you create **relationship definitions**, associating the data of the source and destination attributes.

Kinds of relationships

When you create a relationship definition, you list the participants (business objects or other data entities) that are involved in the relationship, and you specify a type (either the name of a business object definition or the word Data) for each participant. Based on the types of the participants and the number of instances of each participant that can be related in that definition, relationships are classified into the following categories:

- An **identity relationship** establishes an association between business objects or other data on a one-to-one basis. For each relationship instance there can be only one instance of each participant. Identity relationships typically transform the key attributes of business objects, such as ID numbers and product codes.
- A **non-identity relationship** establishes an association between business objects or other data on a one-to-many or many-to-many basis. For each relationship instance there can be one or more instances of each participant. An example of a non-identity relationship is an RMA-to-Order transformation, in which a single RMA (Return Materials Authorization) business object can yield one or more Order business objects.
- A **lookup relationship** establishes an association between data, such as attributes in business objects. The data can be related on a one-to-one, one-to-many, or many-to-many basis. Participants in lookup relationships are of type `Data`. Lookup relationships typically transform non-key attributes whose values are represented with codes, such as marital status or currency code.

For more information about relationship definitions, start with Chapter 1 of the *Map Development Guide*.

Configuring connectors with maps

When you use System Manager to install and configure a connector, you specify the name of an IBM WebSphere InterChange Server map, stored in the InterChange Server's repository, for transforming each business object that the connector supports.

When a connector controller sends a business object for mapping, it executes the map, sending the business object and the mapping information as input.

Summary

This chapter introduced the terminology and concepts associated with business object mapping. The key points to remember are:

- Data mapping transforms business objects. As business objects travel from source applications to collaborations, mapping transforms application-specific business objects to generic business objects. As business objects travel from collaborations to destination applications, mapping transforms generic business objects to application-specific business objects.
- Maps can be created using the Map Designer and Relationship Designer tools.
- Simple mapping can be done in the Map Designer. For mapping that involves attributes that are equivalent but incompatible in format, use Relationship Designer.

All of the previous chapters described basic features of the IBM WebSphere InterChange Server system. The next chapter describes an advanced feature relevant to some users.

Chapter 7. Transactional collaborations

Transactional collaborations provide data consistency assurances for business processes. This chapter introduces the features of transactional collaborations. It contains the following sections:

The transaction model

This section briefly reviews the concepts from which the IBM WebSphere InterChange Server transactional model are derived. If you are familiar with databases and transactional processing, skip to the next section.

A transaction is a set of related and interdependent steps that form a logical unit of work. Grouping the steps into a transaction ensures that the set of steps are treated as an atomic unit: either the entire set of steps succeeds, or the entire set of steps fails.

Figure 38 illustrates a database transaction that contains two updates, update 1 and update 2.

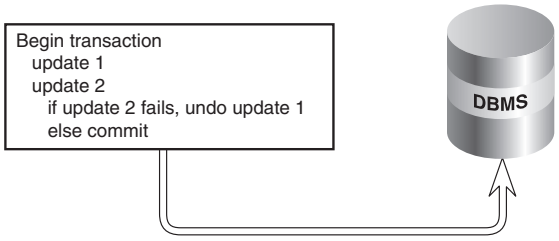

Figure 38. A database transaction

As a transaction is executing, it locks the part of the database that it is modifying, making the data unavailable for modifications by other transactions. Isolation of data from outside interference is one of the defining qualities of transactions.

If the transaction's operations succeed, the transaction completes by writing changes to disk in a **commit** operation. A commit operation releases the transaction's locks, making the updated data available to other transactions.

If an error occurs partway through execution, making it impossible for the entire transaction to succeed, the entire transaction fails. Rather than leave partial results in the database, the transaction backs out the changes that it has already made, leaving the database with the values that it had before the transaction started. The back-out process is called **rollback**.

Another type of database transaction, two-phase commit, is adapted for distributed use across multiple databases. Two-phase commit uses a transaction monitor to coordinate concurrent updates. The transaction monitor first checks that all databases can make the desired change. Even if all can make the change, they must wait for the transaction monitor's signal before doing so. If all cannot make the change, none do.

Every transaction must maintain data consistency in its database, but the two-phase commit protocol extends the scope of this requirement beyond individual databases. A common example is a funds transfer, in which the transaction monitor ensures that the funds debited from one account are credited in the other account or that neither account is modified.

What is a transactional collaboration?

A transactional collaboration is a collaboration whose data modifications can be rolled back. Like database transactions, transactional collaborations are all-or-nothing operations: either the entire collaboration succeeds or the entire collaboration fails. In addition, a transactional collaboration can detect and react to data isolation violations that might compromise the logic of its data operations.

Transactional collaborations are based on the principles established in database transactions and two-phase commit protocols. However, transactional collaborations differ from database transactions because of the unique nature of collaborations: distributed across any number of applications, asynchronous, long lived, and positioned outside of applications rather than inside them.

A transactional collaboration and a nontransactional collaboration execute differently. A transactional collaboration executes under the control of the InterChange Server's transaction service, which controls execution, rollback, and isolation checking.

This section describes the components of transactional collaborations.

Transactional scenarios

A collaboration template's Minimum Transaction Level property determines whether the collaboration is transactional, and applies to all of its scenarios. All collaboration objects created from a template inherit its minimum transaction level.

Actual transactional logic applies at the scenario level. In a transactional collaboration, each scenario is a **transactional scenario**, whose start implicitly begins a transaction and whose successful completion implicitly commits the transaction.

Each scenario runs in a single execution context. If a collaboration is part of a collaboration group, a scenario might call another collaboration to do some work and then return. The scenario that is called executes within the caller's execution context and is part of the same transaction.

Subtransactions

Each time a step in a scenario causes an application to modify data, that step initiates a transaction in the application itself. A step that causes an application transaction is a **subtransaction step**: a transaction within a transaction, where the larger transaction is the scenario itself.

To illustrate transactional concepts, this chapter uses the scenario shown in Figure 39. The scenario handles updates as part of a pricing management collaboration. The collaboration ensures that when an ERP application changes the price of a product, the change is matched in the customer service application and product configuration application. The illustration omits all processing details that are irrelevant to transactional semantics.

The scenario has four action steps, Action 1 (A1) through Action 4 (A4), and it interacts with three business objects, A, B, and C. Business object A represents a particular product in the ERP system, B represents the same product in the customer service application, and C represents the product in the product configuration application.

Note: The format *BusinessObject.Attribute* represents a particular attribute in a business object. For example, A.Price represents the Price attribute of business object A.

The scenario appears on the left. On the right, comments show what happens when a product manager logs into the ERP system and changes the price of the product to $700.

Note: For simplicity, the figures in this chapter do not show connectors.

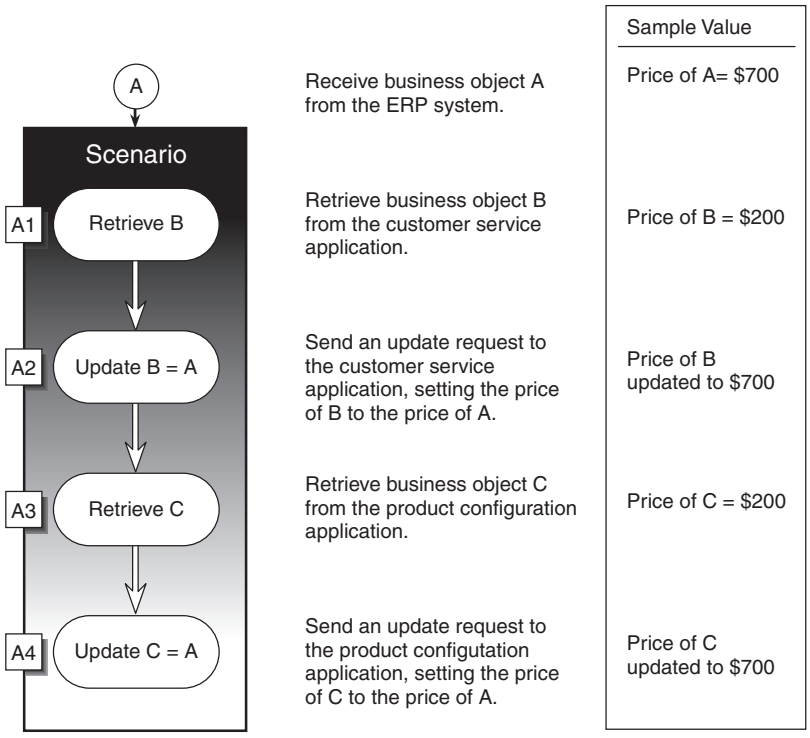

Figure 39. Sample scenario

To make the scenario transactional, the first step is to identify the subtransaction steps. Figure 40 shows that the subtransactional steps in the scenario are A2 and A4, because they cause transactions at the customer service and product configuration applications, respectively.

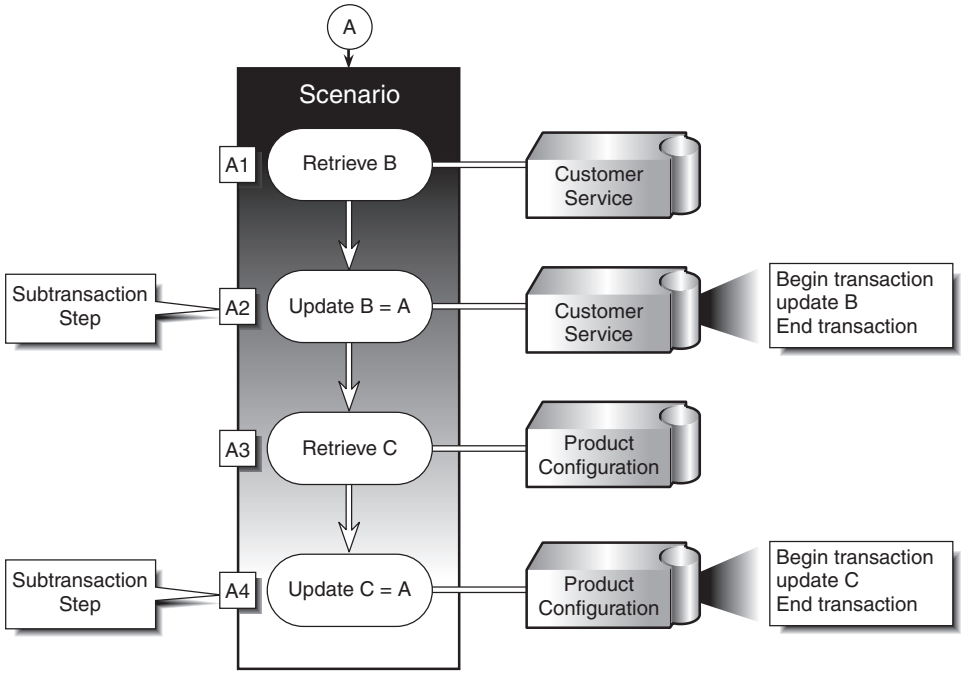

Figure 40. Subtransaction steps in a transactional scenario

A step that performs a retrieve request, such as A1 or A3, is not a subtransactional step because it does not cause the modification of data.

Compensation and rollback

The way that a transactional scenario responds to failure differs from the way that nontransactional scenarios respond to failure. When a nontransactional scenario fails, the collaboration simply logs an error and terminates. When a transactional scenario fails, the scenario rolls back to leave data in a consistent state across all of the involved databases.

Use of compensation in rollback

When an error occurs, subtransactions might have already caused applications to commit work. Therefore, rollback is done through the use of **compensation steps**, actions that counteract the effects of other actions. A compensation step executes only during rollback.

During rollback, the InterChange Server steps backward through the execution path. For each subtransaction step that completed, the server executes the associated compensation. Rollback is complete when all executed transactional steps have been compensated. If an error occurs during rollback, the InterChange Server simply logs the error.

A compensation step can consist of any action that a collaboration developer wants to use to counteract the original action. Table 8 lists some common ways in which a collaboration developer might choose to compensate for specific actions.

Table 8. Compensation Examples

Action	Compensation
Create business object	Delete business object
Delete business object	Create business object

Table 8. Compensation Examples (continued)

Action	Compensation
Update business object	Update business object, restoring the former values

Although compensation typically consists of reversing the original action's data modifications, it need not do so. For example, compensation for a create request could be another create request, this time causing a record to be written to an audit log. Compensation is therefore a **logical undo** operation, and not necessarily an actual undo operation.

Designing for rollback

A collaboration designer must design the scenario to enable rollback to occur. Figure 41 shows how the example scenario could be modified to permit rollback.

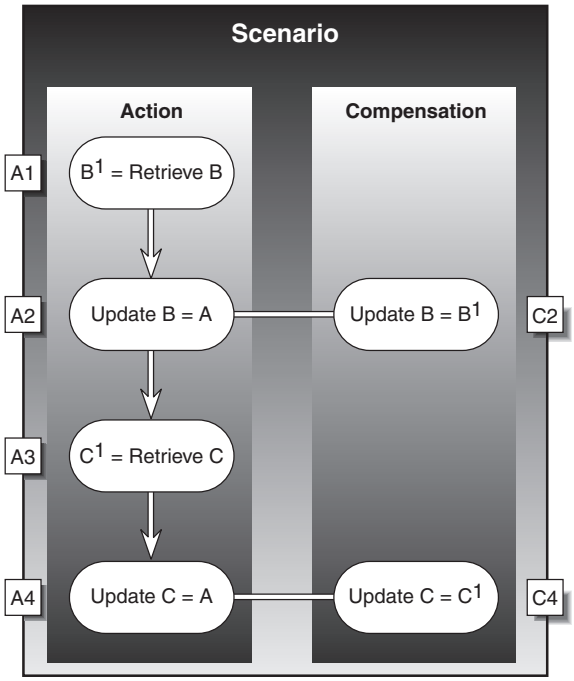

Figure 41. Actions and compensations

The example has changed as follows:

- Action steps A1 and A3 now use B^1 and C^1 to store the original values of B and C.
- Compensation steps C2 and C4 are new compensation steps that restore the original values of B and C

Imagine that the scenario encounters an error after executing A2. During rollback, compensation step C2 executes, using the value stored in B^1 to return B to its original value.

Runtime illustration

The following figure illustrates execution of the sample scenario. In the illustrated runtime sequence, action steps A1, A2, and A3 execute successfully. During the execution of A4, however, an error occurs at the application. When the scenario receives an error instead of a successful status for A4, it fails, and rollback begins.

Stepping back through the steps, the transaction service proceeds as follows:

- A4 never completed modifying data, so it does not need compensation.
- A3 was a retrieve operation, so it does not need compensation.
- A2 completed a data modification and has compensation defined, so its compensation, C2, is executed.

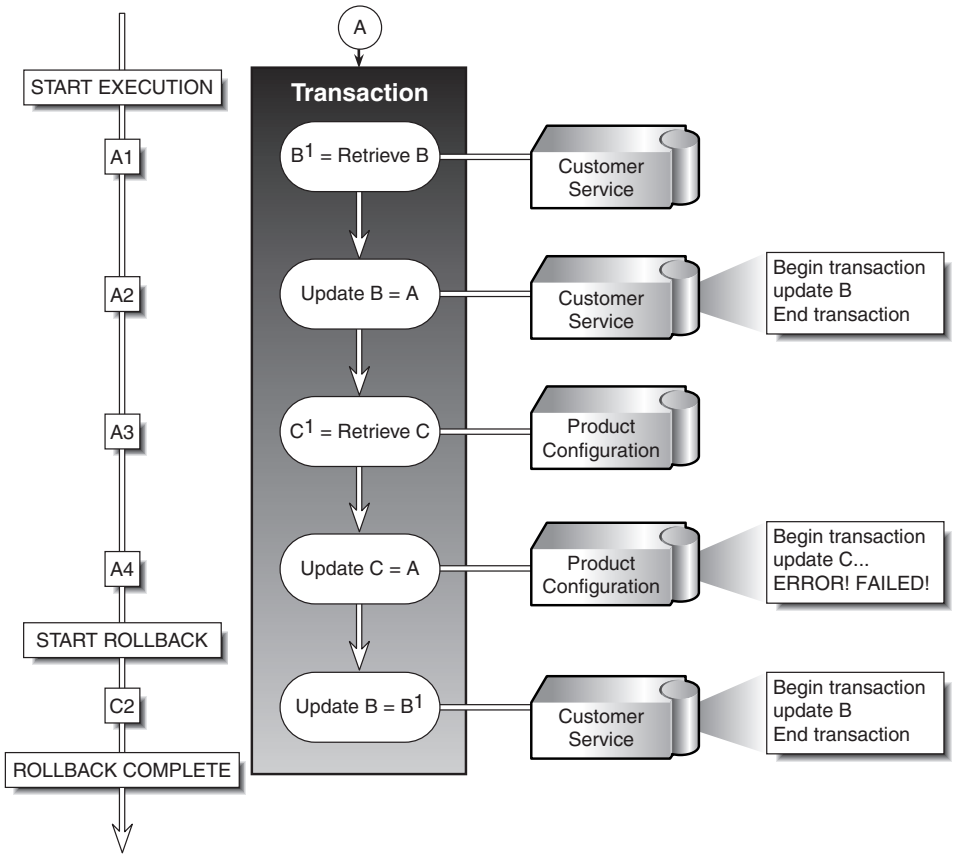

Figure 42. Rollback of a transactional scenario

Data isolation

The IBM WebSphere InterChange Server system provides several levels of transactional execution for collaborations. At the higher levels, the InterChange Server detects isolation violations. Isolation is the assurance that, while a transaction is executing, it has exclusive access to its data. Because the transaction's data does not change between operations inside the transaction, the effects of the transaction's logic are predictable.

In a database transaction or two-phase commit, database locks ensure isolation during execution of the transaction. In the IBM WebSphere InterChange Server environment, a transactional scenario cannot lock data between steps; each step causes an application transaction that, upon completion, releases its lock. Other programs can then view and modify the updated data.

A scenario might need to use the same piece of data more than once. For example, multiple action steps might update the same data. Furthermore, if a scenario fails and rolls back, a compensation step might revisit data that was set by previous actions, restoring it to the original value.

For data that a scenario uses more than once, the time period between the completion of an application transaction caused by one step and the start of an application transaction caused by another step is a **window of vulnerability**.

Figure 43 illustrates the window of vulnerability between two transactional steps that update the same business object.

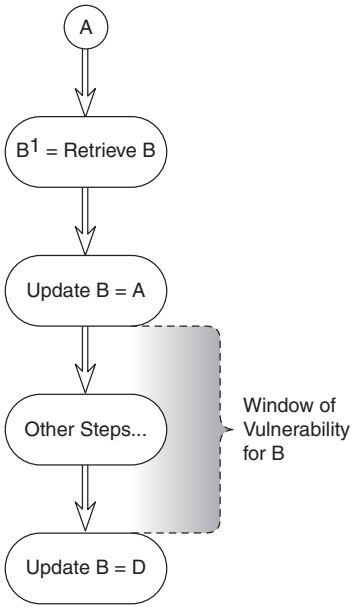

Figure 43. Window of vulnerability

During the window of vulnerability, data isolation cannot be guaranteed. Other transactions might step in to change the state of the data between the scenario's visits, possibly causing errors in the scenario's results.

For some collaborations, isolation might not be relevant; for example, a collaboration might be the only program that modifies the data. However, if a scenario modifies data that other programs might also modify, isolation violations can be problematic.

The IBM WebSphere InterChange Server system provides **isolation checking** for collaborations that need to protect against isolation violations. If a collaboration runs at Best Effort or Stringent transaction level, the transaction service and connectors work together to determine whether data has been compromised between repeat visits. The next section describes the transaction levels and explains how isolation checking occurs at each level.

Transaction levels

A collaboration's transaction level determines the mechanisms by which the system executes the scenarios in the collaboration.

A collaboration becomes transactional at the development phase, when its developer determines whether the collaboration requires transactional execution. If it does, the developer adds compensation steps to the scenarios in the template and specifies the collaboration's transaction level. Every collaboration has one of the following transaction levels:

- None

- Minimal Effort
- Best Effort (note that this level is not supported for long-lived collaborations)
- Stringent

During configuration, System Manager displays the minimum transaction level of a collaboration template, which a collaboration object made from that template inherits.

Other factors affect the transaction level at which a collaboration object actually executes, however. At configuration time, an administrator binds the collaboration object to connectors or collaboration objects, or to both. Each connector or collaboration object has a particular transaction level, and the collaboration must run at a level that all bindings support. For example, a connector's transaction level represents the level of support that it can provide, based on the capabilities of its application. The collaboration cannot run at a level that one of its connectors does not support.

None

A collaboration whose transaction level is None is not transactional. If an error occurs during the execution of the collaboration, an error message is logged.

Even if a collaboration template contains compensation steps, a collaboration object created from that template could execute at transaction level None. This could happen, for example, if the connector object's bindings do not support a higher transaction level. If that happens, the compensations are ignored.

Minimal Effort

A collaboration with Minimal Effort transaction level has compensations defined for its scenario's transactional steps. If an error occurs during execution of a scenario, the InterChange Server rolls back the scenario, executing the compensation for each transactional step that executed.

If the scenario called a different collaboration object to do a piece of work—that is, if the collaboration is part of a collaboration group—the scenario that executed in that collaboration also rolls back.

The Minimal Effort transaction level might be appropriate for a collaboration under the following conditions:

- It is important to roll back on failure to undo committed work.
- The collaboration's scenarios modify data that no other program or collaboration modifies, so there is no need to use isolation checking.

Best Effort

A collaboration that executes at the Best Effort transaction level uses isolation checking. The Best Effort transaction level is appropriate for collaborations that must guard against any data inconsistency.

When a collaboration executes with isolation checking:

- The InterChange Server's transaction service works with connectors to check data that a scenario revisits during the course of its execution.
- The system checks the current state of the data against its last known state before applying a subsequent operation.

- If the data is the same since the previous check, the data has been virtually isolated, and the operation proceeds.
- If the data is not the same since the previous check, the operation results in an error.

Isolation checking provides good transactional semantics, but requires additional database access at the InterChange Server and more communication between connectors and the InterChange Server. This additional system work has an effect on performance. For this reason, you should weigh performance concerns against data consistency issues when choosing a transaction level.

However, even at Best Effort transaction level, a small window of vulnerability exists. Between the start of the isolation check and the time that the collaboration continues its operation, another transaction could modify the data. Nonetheless, Best Effort is the highest transaction level available with many combinations of applications, because few application APIs support the Stringent level.

Note: The Best Effort transaction level is not supported for long-lived collaborations.

Stringent

Stringent transaction level provides the highest degree of isolation assurance, but it is available only with applications whose APIs allow client applications to do atomic test and set operations. The connector working with such an application locks data during the isolation check, thereby removing the possibility that another program could modify the scenario's data. Few applications currently provide this ability.

Recovery

Any software program runs the risk of interruption from a hardware or software event that unexpectedly stops execution. The InterChange Server has a robust mechanism for recovering transactions that are in progress when an unexpected exit occurs.

When the InterChange Server comes back up after an unexpected exit, it checks for collaborations that were in an active transaction state at the time of exit. A two-phased recovery then begins:

1. The InterChange Server reactivates each interrupted transactional collaboration and rolls it back (unless the collaboration was using the long-lived business process feature). During this period, the server does not deliver new events to the collaboration.
2. The InterChange Server retrieves the original triggering event for each interrupted collaboration from the event management service and redelivers it. The collaboration runs, reprocessing the event.

When recovery is complete, the InterChange Server allows the collaboration to process new events.

Transactional collaborations and long-lived business processes

Transactional collaborations can make use of the long-lived business process feature, which is enabled as an option when you create the collaboration template. For a collaboration object created from a template that has the feature enabled, if the InterChange Server crashes while the flow is waiting for a service call

response, the flow will be able to continue from its saved context during recovery. But if the feature is not enabled, an InterChange Server crash triggers compensational rollbacks

Summary

This chapter was an overview of transactional collaborations. The key points to remember are:

- The IBM WebSphere InterChange Server transactional model is derived from the use of transactions in DBMS environments. However, the IBM WebSphere InterChange Server model is adapted to suit the unique nature of collaborations.
- A transactional collaboration is one in which the scenarios are equipped to handle rollback, and that executes under the control of the InterChange Server's transaction services.
- When a scenario in a transactional collaboration fails, the InterChange Server rolls the scenario back, executing the compensation for each completed action.
- With higher transaction levels, the system can also provide isolation checking.
- Recovery mechanisms for transactional collaborations ensure that the InterChange Server does not leave collaborations in an unknown transactional state, even after a system failure or other unexpected event.

Chapter 8. Language-specific behavior support

WebSphere InterChange Server and WebSphere Business Integration Adapters (WebSphere Business Integration products) support bidirectional languages such as Hebrew and Arabic through *bidirectional enablement*. Bidirectional enablement is a mechanism for accurately displaying and processing of bidirectional script data inside components (connectors, Adapter Framework, collaborations and maps) bundled with WebSphere Business Integration products. While WebSphere Business Integration products can process data in any bidirectional format, the data in WebSphere Business Integration products domain is set to one uniform bidirectional format, which is the Windows standard bidirectional format (logical left to right).

This chapter provides an overview of bidirectional support in WebSphere Business Integration products and the requirements and processes used in enabling bidirectional support. It contains the following sections:

Locale support in the WebSphere Business Integration products

The *locale* is the part of a user's environment that brings together information about how to handle data specific to the particular country, language, or territory. The locale is typically installed as part of the operating system.

The locale provides the following information for the user environment:

- Cultural conventions according to the language and country (or territory):
  - Data Formats:
    - Dates: define full and abbreviated names for weekdays and months, as well as the structure of the date (including date separator)
    - Numbers: define symbols for the thousands separator and decimal point, as well as where these symbols are placed within the number
    - Times: define indicators for 12-hour time (such as AM and PM indicators) as well as the structure of the time
    - Monetary values: define numeric and currency symbols, as well as where these symbols are placed within the monetary value
  - Collation order: how to sort data for the particular character code set and language
  - String handling includes tasks such as letter case (upper case and lower case) comparison, substrings, and concatenation
- Character encoding: the mapping from a character (a letter of the alphabet) to a numeric value in a character code set. For example, the ASCII character code set encodes the letter A as 65, while the EBCIDIC character set encodes this letter as 43. The *character code set* contains encoding for all characters in one or more language alphabets.

The locale name has the following format:

```
ll_TT.codeset
```

where ll is a two-character language code (usually in lower case), TT is a two-letter country and territory code (usually in upper case), and codeset is the name of the associated character code set. The codeset portion of the name is optional. The locale is typically installed as part of the installation of the operating system.

Establishing a locale

Setting default user locale to a bidirectional locale, (Arabic or Hebrew), is a prerequisite for enabling correct processing of locale-dependent data. If the user default locale is not set to a bidirectional locale, then the bidirectional characters are incorrectly displayed (see *System Installation Guide for Windows* for more information). The Windows locale setting for Hebrew is usually: iw_IL, and for Arabic it is usually: ar_AE. The ASCII code set for Arabic is 1256 and for Hebrew is 1255 (see *System Installation Guide for Windows* for details on how to set bidirectional locales).

Processing locale-dependent data

The Java runtime environment within the Java Virtual Machine (JVM) represents data in the Unicode character code set. Unicode contains encoding for characters in most known character code sets (both single-byte and multibyte). Most components in the WebSphere Business Integration products are written in Java. Therefore, when data is transferred between most WebSphere Business Integration product components, there is no need for character conversion.

If data is transferred between an external application and the WebSphere Business Integration environment, it can be in one-byte character code rather than Unicode which is two-byte, therefore steps need to be taken to make the correct character conversion from one format to another. To address this event, connectors have been internationalized so that they can support both double-byte and single-byte character sets to deliver message text in the specified language. When a connector transfers data from a location that uses one character code set to a broker, it transforms the data from a single-byte code set to a double-byte code set (Unicode) because the integration broker uses Unicode.

Because log and informational messages include data in one-byte code set, the appropriate locale setting is needed for accurate transformation of bidirectional text between Unicode and one-byte code sets. Therefore, to log error and informational messages in the appropriate language and for the appropriate country or territory, configure the locale standard configuration property for your environment to the specific bidirectional language setting. For more information on configuration properties, see Establishing a locale in the *System Installation Guide for Windows*.

Note: You must properly set the default code page in the DOS prompt. If the code page is not properly set, the data flashed out might not be readable by standard Windows editors or viewers (see Changing your DOS prompt code page for ASCII settings for Hebrew and Arabic in the *System Installation Guide for Windows*).

Design considerations

Encoding is processed within the WebSphere Business Integration environment at three levels:

- Content data
- Meta and configuration data
- Log and trace data

The content data level processes the business object runtime instance data. The meta and configuration level processes the data used by connectors to establish and maintain communication between external applications and the WebSphere Business Integration environment. The log and trace data level processes informational messages logged into various logs and trace files. Some adapters are more flexible than others with encoding support. For example, an XML connector has the ability to process the encoding specification in the XML file headers that provides full encoding support. A connector for SAP, on the other hand, has limited encoding support provided by the SAP client API. For content level as well as the meta and configuration data level, if the connector does provide a configuration option, then the locale setting is overwritten by the encoding specification option. For the log and trace data level, the default DOS prompt code set is used for encoding flashed data.

Content data encoding

There is no uniform approach for content data encoding support among connectors. There are three basic approaches that can be used to deal with the variety of ways connectors handle content data encoding support. These approaches are:

- Automatically handling encoding by middleware or third party APIs used for serialization and de-serialization of business object data (for example, SAP/PS client API and JDBC driver for RDBMS)
- Using the underlying technology that has the appropriate configuration for the connector or data handler (for example, using the charset parameter in the XML header for XML DH or MIME charset specification per attachment in the Email connector)
- Enabling WebSphere Business Integration products to handle encoding internally and specifying via a provided mechanism such as the DataEncoding attribute of `MO_JTextConnector_Default` meta-business object used in JText connector

These three approaches provide a means of handling content data by the adapters (see your adapter guide to find out what approach is applicable for your specific adapter).

Meta-configuration data encoding

The encoding support for meta-configuration data often depends on the capabilities of the middleware adapter code used for setting up a connection to an external application.

The meta and configuration data encoded in ASCII in WebSphere Tools (as part of the adapter template or supported business object templates) is stored in the repository as Unicode. At runtime, this data is used as an argument for either native Java APIs or third party APIs to establish a communication link with an external source. The middleware used for establishing a communication link in the adapter code can range from native Java APIs or packages, (for example, Email or WebSphere MQ connectors), to third party APIs (for example, PeopleSoft). In

addition, there can be several layers of middleware separating the adapter Java code from the external source as in the case of JText and the File System folder located on a non-Windows system. If the File System folder is accessed from the Windows process, the call issued from the Java API might be indirectly serviced by the software allowing the mounting of non-Windows partitions instead of the target Windows File System. Because neither native Java APIs, third party API, nor middleware is guaranteed to support explicit encoding specification, WebSphere Business Integration products only support Unicode encoding for meta and configuration bidirectional data.

Log and trace data encoding

By default, the encoding of the bidirectional data flashed out into the log and trace files is identical to the code page of the DOS prompt that invokes the process. Unfortunately, the default DOS prompt code page associated with the bidirectional locales for Arabic and Hebrew is not the same as the standard code page for those languages on the Windows operating system (1256 and 1255). Consequently, if the DOS prompt code page for the WebSphere Business Integration Java processes is not changed, the bidirectional data flashed out by those processes into the log trace files is not correctly displayed in most Windows viewers using standard code pages for bidirectional languages data display (see Changing your DOS prompt code page in the *System Installation Guide for Windows* for more information).

Bidirectional script support

WebSphere Business Integration products provide bidirectional support for languages with a bidirectional script. A bidirectional script contains both text that is written from right to left and embedded numbers or segments of text in western scripts, (for example Latin-based scripts such as English, French, Cyrillic-based, or Greek) that are written from left to right.

Arabic and Hebrew are the two major language groups that use bidirectional scripts. The Arabic script group includes Arabic, Farsi (Persian), Urdu, as well as other languages. The Hebrew script group includes not only Hebrew, but also Yiddish and Ladino. Because both language groups have alphabets (only 27 characters), they can accommodate a single-byte encoding scheme.

Bidirectional language characteristics

There are two main characteristics that distinguish bidirectional script from a Western language script (such as English, French, German, Greek and others). These two characteristics are *bidirectionality* and *shaping*.

Bidirectionality

Bidirectionality consists of seven key concepts:

- Segmentation
- Nesting
- Global orientation
- Logical vs. physical order of bidirectional text
- Text-types and associated re-ordering methods
- Symmetrical swapping
- Widget mirroring of translated GUIs

Note: In all examples shown below, capitalized letters such as, DCBA are used to represent Arabic or Hebrew letters.

Segmentation

Segmentation is defined as a string that has one portion of text within a string that has a distinct directionality. Therefore, a script can have the main portion consisting in a right-to-left orientation while another portion(s) has a left-to-right orientation. An example of bidirectional segmentation is the street address, Entrance B 25 Maple Street. this address when written in Hebrew is: B ECNARTNE 25 TEERTS ELPAM. In this example, the major parts of the string text, B ECNARTNE and TEERTS ELPAM, have a right-to-left orientation but the number 25 has a left-to-right orientation.

Nesting

Nesting is defined as a text segment that has one directionality while also having within that segment an additional segment with an opposite directionality. Again, using the street address, B ECNARTNE 25 TEERTS ELPAM. This address has one level of nesting. The street name, TEERTS ELPAM, is written with a right-to-left orientation, but the flow is then reversed to allow the correct entry of the street number 25that has a left-to-right orientation. After the street number, the flow orientation is reversed again to right to left for B ECNARTNE.

Global orientation

Global orientation, which is also referred to as Writing Order, Reading Order, or Paragraph Direction designates the side of the screen, window, or page where the writing of the text has started. In addition, global orientation is *context dependent*. This means that text meaning is dependent on its context. An example, of context dependency is demonstrated using a sentence written as a bidirectional script. The sentence reads:

```
FRED DOES NOT BELIEVE taht yas syawla i.
```

This sentence has one meaning when it is read from left to right (Fred does not believe I always say that), and another meaning when read from right to left, (I always say that Fred does not believe). Because the global orientation is not always obvious from the context, the application developer must be aware of how the text will be read (left to right or right to left).

Physical and logical text ordering

Bidirectional text can be stored in either logical or physical order. In workstation environments, the preferred means for entering and processing bidirectional text is logical order because text is processed similarly to Latin text. When using logical order in storage, you must provide the means to reverse segments whose direction is opposite to the global orientation. For example, if global orientation is English (left to right), then segments in Arabic and Hebrew need special processing to appear in their native right-to-left direction. Conversely, the preferred way to store bidirectional text in mainframes for entering and processing bidirectional text is in physical order. Therefore, when integrating bidirectional text from mainframe and workstation environments you must transform the text to a layout where all the text has the same text order.

When using text order in bidirectional scripts, physical and logical order are also important. Physical order refers is how the text segment is physically presented and logical order refers to how text script segments are typed (or pronounced if read aloud). Depending on the situation, some segments may need to be re-ordered in either logical or physical order. For example, the statement:

```
my wife's name is ILIN
```

Overall, this statement has a left-to-right orientation. The reader reads the text with the first letter being m, followed by, y, and so forth. In the physical order, the letters i and s are followed by the letter I of the segment containing ILIN, but in Hebrew, ILIN is pronounced NILI, therefore, in the logical order, the first letter of the name segment, is N not I.

Text-type

Text-type is defined as the most appropriate approach for recording a specific text. This means that different text-types require different recording techniques. There are three text-types. used for recording: visual, implicit (logical), and explicit.

Visual text-type is the oldest form of recording, and is a simple copy of the entire screen. This form of text-type is dependent on the programmer knowing where the embedded segments are located and processing them accordingly. Many legacy applications and their files use this type of text.

Implicit text-type recording assumes that the letters of the Latin alphabet have inherent left-to-right directionality, and that Arabic, Persian, Urdu, and Hebrew alphabets have inherent right-to-left directionality. To accommodate bidirectionality, an algorithm of implicit text processing is used to recognize segments based on their inherent directional characteristics, allowing segment inversion to be performed automatically. The main limitation of implicit text-typing is the inability to handle strings where numbers and letters (in both left-to-right and right-to-left) are mixed, such as in the case of part numbers.

Explicit text-type recording assumes that there are additional control characters embedded in a text string that directs the explicit algorithm to perform segment inversions, shaping or numeral selections, and other transformations. The limitation of explicit text-typing is needed for automatic processes to handle embedded controls. There is a specific technique that bridges implicit and explicit text-typing. This technique is the *basic display algorithm* that is defined in the Unicode Standard Bidi algorithm.

Symmetrical swapping

Symmetrical swapping is the ability to handle characters, such as: `<`, `(`,`[`,`{`, that have a complementary symmetric character with an opposite directional meaning: `>`,`)`,`]`,`}`. These characters are problematic because global swapping would, for example, change `A > B` to `B > A`. Symmetrical swapping enables character conversion of the symbol to `B < A`.

Widget mirroring

Widget mirroring of a translated GUI mirrors the GUI to match the directionality of the language. For example, Widget mirroring can move the menu buttons and navigation tree to the right instead of the left. Otherwise the frames and windows are not mirrored. Figure 44 on page 97

shows widget mirroring of a drop-down menu.

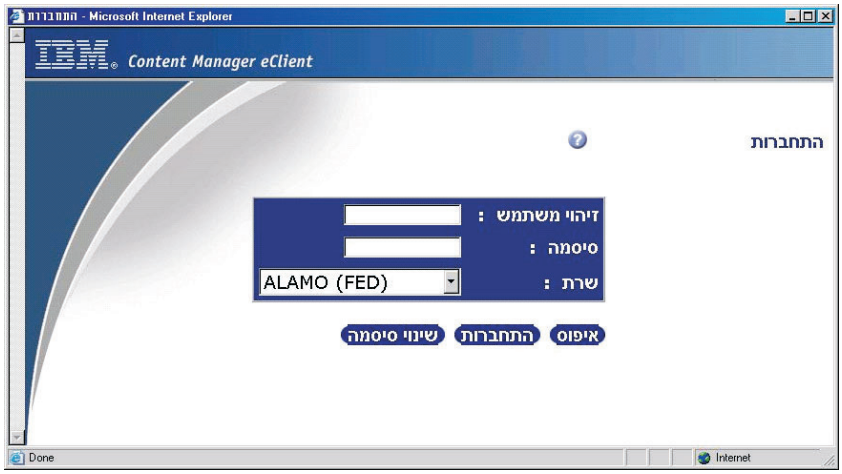

Figure 44. Widget-mirrored window showing bidirectional labels

Shaping

Shaping is a characteristic of many complex languages, particularly the **cursive** languages Arabic and Hebrew. A writing system is cursive if it has adjacent characters in a word connected to each other, and is more suited to handwriting than to printing. In Arabic, for example, some letters can only connect to the letter on their right. In addition, letters can assume different shapes according to their position in the word and the connective properties of adjacent letters. These points make shaping important in rendering bidirectional text intelligible. The shaping process renders characters to their appropriate presentation forms by replacing an abstract representation of a character with the proper shape. This is accomplished by using the base form of a character to allow the selection of a particular cursive character without specifying its shape.

The proper shape of a character is then selected by a shape determination routine that allows for automatic (algorithmic) selection of the appropriate shape according to the context directed by either the software or the user. In most cases, the basic shapes of a cursive language text are stored. There are two other characteristics that make up shaping. They are character composition and national numbers.

Character composition is defined as the correspondence between the number of text characters stored with the number of text characters presented. To maintain correspondence, devices such as **ligatures** and **diacritics** are used. Ligatures are used when two or more characters can be represented by a single character that occupies one presentation cell. Diacritics in bidirectional scripts are marks located in a certain orientation to a consonant, (either above, within, below or near it), to represent vowels. When these marks are stored, they occupy physical positions, but if they are used for representation, they can occupy the same cell as the associated consonants. In Arabic, spacing diacritics are currently implemented as separate characters that are rendered following the character to which the diacritics belong.

National numbers also need special treatment because they are used differently in different languages. For example, in Hebrew, numbers are represented using Arabic digits (1,2,3...0). However, cursive languages such as Arabic, Farsi and Urdu have their own national glyphs to represent digits. The label for digits used in

cursive languages is either Hindi or Arabic-Indic digits. Whether in Arabic, Hindi or Arabic-Indic, all simple numbers are presented left to right, but mathematical formulas can differ from language to language. For example, in Arabic, mathematical formulas are written left to right, but in Persian, they are written right to left. Therefore, while numbers are usually encoded as Arabic digits, they can be presented as either nation-specific glyphs or Arabic digits according to the intent of the user or developer.

Layout transformations and attributes

Enabling bidirectional scripts calls for special attention to:

- "Text layout"
- "Layout attributes"
- "Layout transformations" on page 99

Text layout

Bidirectional text can have different layouts. A layout differs according to the segments used in the text, and also in the case of Arabic script, it also differs in the shaping of characters and the numerical shapes used. Transformations between the different layouts require transformation functions, also called layout functions or layout services functions.

Layout attributes

To define the characteristics of a bidirectional text layout, a set of attributes is needed. Bidirectional attributes are needed to ascertain the actual layout of text is and how it should be transformed. These attributes are usually external to the text and are stored in an external resource file. The five bidirectional attributes are:

- Orientation
- Text-type
- Text shaping
- Symmetric swapping
- Numerical shapes

Orientation

Orientation designates the boundary of the presentation area, (window, frame, page), where the writing of the bidirectional text starts. This designation is based on the directionality of the first character in the text. The directionality can be right to left, left to right, or contextual.)

Text type

Text type designates the kind of algorithm used when transforming the text layout. There are three types of algorithms that can be used: visual, implicit, or explicit. A visual algorithm copies entire lines of text as they appear regardless of existing embedded directional segments. An implicit algorithm recognizes directional segments based on the natural directionality of the characters, (for example, right to left for Arabic and left to right for English), and preforms segment inversions accordingly. An explicit algorithm recognizes directional segments and performs inversions based on visual, explicit, and directional controls embedded in the text.

Text shaping

Text shaping is important for Arabic scripts where characters assume different shapes according to their position in a word as well as the connectivity traits of the surrounding characters.

Symmetric shaping
Symmetric shaping designates when specific characters such as (, >, [, { need to be interchanged with), <,], } in order to preserve the logical meaning of the presented text.

Numeral shapes
Numeral shapes designates whether the numbers embedded in an Arabic script have to be presented using the European digit shapes or Arabic-Indic digit shapes.

No single combination of possible values of bidirectional layout attributes is predominant. Existing applications process data with different possible combinations of these values. Therefore, when a bidirectional data stream is passed to an application, it is important that the application can recognize the associated text attributes.

Layout transformations

Bidirectional text is stored and processed in different environments (platforms) and in different layouts. In order to create a transformation from one layout to another, layout transformation functions are required. WebSphere Business Integration products use layout transformation functions that are based on the Unicode BiDi Algorithm. This algorithm can be found at:

`http://www.unicode.org/reports/tr9/)`

and are implemented in IBM Java SDK 1.4.1:

`http://www-106.ibm.com/developmentworks/java/jdk/bidirectional/JAVABIDI.htm`

Enabling bidirectional scripts in WebSphere Business Integration products

Bidirectional script enablement in WebSphere Business Integration products occurs at different levels and through different component configurations. Bidirectional enablement is actualized on three different levels.

- Through displaying, typing, storing, and retrieving bidirectional script characters created using WebSphere Toolset (for example, Connector Configurator, BO Designer, System Manager, etc.)
- Using the code page translation for converting bidirectional characters from one format to another (between Unicode code set and single-byte code set)
- Using bidirectional text transformations to translate between Windows bidirectional format (format used in WebSphere Business Integration environment) and different bidirectional formats used in external applications.

The following WebSphere Business Integration products are enabled for processing bidirectional scripts data: Toolset, Adapter Framework, ICS broker, as well as nine adapters (JText, JDBC, Email, XML, WebSphere MQ, SAP, PeopleSoft, Web services, and Lotus Domino) and appropriate ODAs (JDBC, XML, WSDL).

Out of those components mentioned above, those belonging to the first-level are the WebSphere Business Integration Toolset that are Java-based tools such as System manager, C++-based tools such as Business Object Designer and Connector Configurator, and Web-based tools such as Dashboard. Those components belonging to the second-level components are the bidirectional enabled adapters. Those components belonging to the third-level are the bidirectional enabled adapters and Adapter Framework.

Connectors

Connectors mediate between an application and one or more collaborations. They process content bidirectional data from an application and use meta and configuration data defined by WebSphere Tools, like Connector Configurator, to establish communication between an application that uses the native bidirectional format to one or more collaborations.

Adapter Framework

The Adapter Framework provides a structure that links an application to one or more collaborations and handles the transfer of the content data between an application and a collaboration. The Adapter Framework is enabled for bidirectional languages and enforces consistency of content data bidirectional format.

Toolset

The WebSphere Business Integration Toolset provides administrative and development tools for use with InterChange Server. The tools in the Toolset are implicitly enabled for processing bidirectional script data and do not require any additional configuration aside from the bidirectional locale (see *System Installation Guide for Windows* for more information).

The next section provides a brief overview for enabling each of the following components:

- "Enabling connectors for bidirectional scripts"
- "Enabling Adapter Framework for bidirectional scripts" on page 101
- "Enabling collaborations for bidirectional scripts" on page 101
- "Enabling maps for bidirectional scripts" on page 102

Enabling connectors for bidirectional scripts

A **connector** mediates between an application (or other programmatic entity, such as a Web server) and one or more collaborations. A connector can be specific to an application—such as SAP R/3, version 4—or to a technology, such as a data format or protocol (XML or EDI). Connector communications with collaborations can take two forms:

- Application **event notifications** that are passed to connectors from for specific applications
- **Request processing**, that connectors perform on behalf of collaborations.

(See Chapter 5, "Connectors," on page 59 for more information.)

Connectors process content data from a data source (application or programming entity) using meta and configuration data defined by a WebSphere Tool such as Connector Configurator. In the WebSphere product environment the meta and configuration data is represented and stored in standard Windows bidirectional format. Because an external application can hold the same meta and configuration data in a bidirectional format different from the Windows bidirectional format used in the WebSphere Business Integration products, a transformation is required for correct communication between the WebSphere Business Integration environment and the external application. Connectors that are bidirectional language enabled can be configured by the connector BiDi.Metadata standard property to enforce the bidirectional format of the meta and configuration data specific to that external application.

Enabling Adapter Framework for bidirectional scripts

The Adapter Framework links a connector to InterChange Server and WebSphere Business Integration Adapters allowing content to flow between an application or programming entity and one or more collaborations. Whereas connectors process the meta and configuration type of data, the Adapter Framework processes the actual data content such as the business object attribute values. This is because the Adapter Runtime component is common to all connectors.

The role of the Adapter Framework becomes one of preserving data consistency between the WebSphere Business Integration environment presentation, in which data is represented in Windows bidirectional format, and external application presentations, where might use different bidirectional formats. Therefore, if data is flowing from the WebSphere Business Integration environment to an external application, the Adapter Framework makes the necessary transformations, if needed, from Windows bidirectional format to an external bidirectional format. Conversely, if the external bidirectional format is different from the Windows bidirectional format used in WebSphere Business Integration environment, the Adapter Framework performs the necessary transformations in to the Windows bidirectional format.

The Adapter Framework can also keep data consistent when a broker other than the WebSphere Business Integration broker is used. If the broker uses a bidirectional format that is different from that of Windows, there is an option to change the default bidirectional format into a format used by that broker.

Enabling collaborations for bidirectional scripts

In a WebSphere product implementation, the term **collaborations** refers to software modules that contain code and business process logic that drive interactions between applications. A collaboration can be simple, consisting of just a few steps, or complex, involving several steps and other collaborations.

Collaborations can be distributed across any number of applications, can handle synchronous and asynchronous service calls, and can support long-lived business processes.

WebSphere product support for bidirectional scripts includes collaborations. Collaborations receive data coming from either the WebSphere environment (connectors, access interface, or other collaborations) or from external sources such as Web Service. The bidirectional data format is enforced either implicitly, by one of the enabled connectors, or explicitly, by bidirectional support or API. If data coming from a component that does not enforce bidirectional support, such as Web Service or a connector that is non-bidirectional language enabled, then format inconsistencies can occur and cause the business logic in the collaborations to fail or to produce incorrect results. You can avoid these types of errors by:

- Accepting input from sources that are enabled for bidirectional languages or in the same bidirectional format as WebSphere Business Integration products.
- Invoking bidirectional support in connectors that are enabled for bidirectional languages that enforce bidirectional format of data passed into the collaborations.
- Using the BiDi APIs in the CwBidiEngine class to enforce bidirectional format of data used or introduced into WebSphere product domain from an external data source (refer to the CwBidiEngine chapter in *Collaboration Development Guide*).

Enabling maps for bidirectional scripts

WebSphere Business Integration products support maps with bidirectional scripts. Maps receive data either from connectors or from external sources. Therefore, data passing into the WebSphere Business Integration product environment that comes from a bidirectional language enabled connector is guaranteed to be in a uniform bidirectional language format (standard Windows bidirectional format). However, data can be introduced into a map from unknown external sources, for instance, data that is exported through a Web service. In the event that a Web service operates with bidirectional data that is not in Windows bidirectional format, two results are possible. The first result is that the connection to such a service might fail. The second result is that the bidirectional data is in a format different from Windows bidirectional format that gives unpredictable results in data processing because the data is being compared against data in Windows bidirectional format (see, Using bidirectional functionality in Activity Editor in the *Map Development Guide*) These errors can be avoided using the same steps discussed in "Enabling collaborations for bidirectional scripts" on page 101.

Handling bidirectional text

There are certain situations where special attention needs to be taken when handling bidirectional text. One situation that needs special attention is migrating repository data from previous versions of WebSphere Business Integration that are not enabled for bidirectional languages (see "Migrating data" for more information). If special actions are not taken, then data with two different bidirectional formats can reside in the same repository that can interfere with proper processing and functioning of business logic. Another situation involves transforming bidirectional text with exceptional patterns such as an FTP URL, and email addresses (see "BiDi APIs" on page 103 for more information).

Migrating data

Some precautions should be used when migrating data from previous versions of WebSphere Business Integration products.

During the migration process the bidirectional data stored from an earlier version can persist and be used along with new bidirectional data introduced via the connectors that are enabled for bidirectional languages. If this situation occurs, the Window format for the bidirectional data being manipulated on the server level is not guaranteed. Consequently, the processing of bidirectional data in, for example, collaborations or maps, might be irreversibly corrupted.

It is suggested that before migrating to the current WebSphere product version, you convert all bidirectional data in the repository into the Windows bidirectional format by using the BiDi APIs provided in the current version. (See the CxBidiEngine chapter in *Map Development Guide* for more information.)

Special bidirectional strings

FTP URL, and email addresses are cases where explicit application of bidirectional transformation can result in the data being inaccurately interpreted. To ensure accurate interpretation, such strings are analyzed before transformation is begun and problematic subcomponents within the string values are identified. In cases where problematic subcomponents are identified, the string is split and bidirectional transformation is applied on each of the subcomponents. After the transformation process has completed, the subcomponents are reassembled into

one single string that represents the accurate transformed value. This value is then stored for later use. This process is used for the treatment of Meta Business Objects.

BiDi APIs

WebSphere Business Integration products come bundled with BiDi API class to enforce bidirectional format in different components. The BiDi API class functions can be used in the Adapter Framework to enforce the bidirectional format of the content data, in connectors to enforce the bidirectional format of meta and configuration data, and in collaborations and maps to enforce the bidirectional format of data imported from an external source.

BiDi methods

Within the BiDi API class are three methods:

- BiDiBusObjTransformation
- BiDiBOTransformation
- BiDiStringTransformation

BiDiBusObjTransformation function

The BiDiBusObjTransformation method transforms BusinessObject type business objects from one bidirectional format to the other. This method is useful for collaborations. (See CxBiDiEngine chapter in either the *Collaboration Development Guide* for more information.)

BiDiBOTransformation function

The BiDiBOTransformation function is applied to BusinessObject instances. This function is useful for the Adapter Framework. (See CxBiDiEngine chapter in either the *Collaboration Development Guide* or *Map Development Guide* for more information.)

BiDiStringTransformation function

The BiDiStringTransformation function is applied to String objects. This function can be used on objects both internal and external to WebSphere product environment. (See CxBiDiEngine chapter in either the *Collaboration Development Guide* or *Map Development Guide* for more information.)

Design limitations

The current design provides a limited solution for bidirectional support. Listed here are some of the limitations:

- Support for bidirectional metadata is provided by only one parameter applicable for all metadata bidirectional properties. This means that when an attribute is based on one bidirectional parameter, either all metadata attributes or no metadata attributes are transformed. Therefore, you cannot set different bidirectional formats for different metadata bidirectional properties or specify what meta-data parameters you want to leave unaffected by this bidirectional property.
- Because not all components in WebSphere Business Integration products are enabled for bidirectional languages for example, Server Access Interface and various connectors, in case such components are used alongside ones that are enabled for bidirectional languages, no uniform bidirectional format of data residing on either InterChange Server or WebSphere Business Integration Adapters can be guaranteed. Consequently, this situation might result in inconsistent representation of bidirectional data on either InterChange Server or

WebSphere Business Integration Adapters and incorrect processing based on it. However no mechanism, for enforcing uniform bidirectional format of data in the either InterChange Server or WebSphere Business Integration Adapters is provided. The responsibility for enforcing this format is the responsibility of the user. For enforcing bidirectional format on data received from components that are not enabled for bidirectional languages the user is provided with BiDi APIs.

- Support for bidirectional content is provided at the connector level. Consequently, the user is restricted to use only one bidirectional format specification per connector. In other words, all business objects supported on the particular connector are supposed to hold bidirectional data in only one bidirectional format.
- WebSphere Business Integration products are enabled for bidirectional languages when the proper manual installation configuration is performed (see *System Installation Guide for Windows* for more information).
- WebSphere Business Integration products have only nine adapters that are enabled for bidirectional languages: JText, JDBC, Email, XML, WebSphere MQ, SAP, PeopleSoft, Web services, and Lotus Domino.
- Selection of the WebSphere Business Integration products internal bidirectional format to be identical to Windows bidirectional format is closely related to the fact that WebSphere product Tools, such as Connector Configurator, Map Designer, BO Designer, etc., are supported only on the Windows platform. In the event that tools support is expanded to include different platforms, the current design will not be sufficient and will need to be modified.
- Selection of Windows bidirectional format as the internal format for InterChange Server and WebSphere Business Integration Adapters presents limits for communication with applications using different bidirectional formats. For example, two applications using a visual bidirectional format that communicates via WebSphere Business Integration products that have, by design, an internal logical bidirectional format. This is due to limitation of bidirectional transformation that is not transitive.
- WebSphere does not support bidirectional characters as part of business object and business object attribute names. Consequently, after the generation of a business object template based on XML/XSD files or database tables with attribute, element, or column names having bidirectional characters, you are required to rename all business object and business object attribute names including non-Latin characters, so that they only include Latin characters.
- Currently, only XML and delimited data handlers are supported. Request-Response data handler is supported only if it uses either XML and delimited data handlers or uses XML and delimited data handlers separately for Request-Response handling. EDI, Fixed Width, and Name-Value data handlers are not supported.
- WebSphere does not support non-Latin characters, including bidirectional characters, as part of its basic entity names, for example Connector, Collaboration, Map, Business Object, Business Object attribute, etc.

Summary

This chapter introduced bidirectional text and how it is enabled in WebSphere Business Integration system. The key points to remember are:

- Bidirectional scripts have text that runs left to right and have segments embedded that run right to left.

- Bidirectional text has two major characteristics, bidirectionality and shaping. Each characteristic must be addressed when developing bidirectional applications.
- A locale must be set up to utilize bidirectional text scripts.
- InterChange Server is enabled for bidirectional languages if appropriate system configuration is performed (see *System Installation Guide for Windows* for more information). WebSphere Business Integration Adapters include nine adapters that are enabled for bidirectional languages.
- Internal bidirectional format in WebSphere Business Integration products is Windows bidirectional format (logical, left to right).

Notices

IBM may not offer the products, services, or features discussed in this document in all countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Burlingame Laboratory Director
IBM Burlingame Laboratory
577 Airport Blvd., Suite 800

Burlingame, CA 94010
U.S.A

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurement may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not necessarily tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information may contain examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples may include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

Programming interface information

Programming interface information, if provided, is intended to help you create application software using this program.

General-use programming interfaces allow you to write application software that obtain the services of this program's tools.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

Warning: Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

Trademarks and service marks

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States or other countries, or both:

IBM
the IBM logo
AIX
CrossWorlds
DB2
DB2 Universal Database
Domino
Lotus
Lotus Notes
MQIntegrator
MQSeries
Tivoli
WebSphere

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MMX, Pentium, and ProShare are trademarks or registered trademarks of Intel Corporation in the United States, other countries, or both.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

System Manager and Adapter Monitor include software developed by the Eclipse Project (http://www.eclipse.org/).

Index

E

F

G

H

I

J

L

M

N

O

P

R

S

T

V

W

Printed in USA