Retek® 10 Integration Bus

Deployment Guide

Retek

The software described in this documentation is furnished under a license agreement and may be used only in accordance with the terms of the agreement.

Information in this documentation is subject to change without notice.

Retek provides product documentation in a read-only-format to ensure content integrity. Retek Customer Support cannot support documentation that has been changed without Retek authorization.

Corporate Headquarters:

Retek Inc.
Retek on the Mall
950 Nicollet Mall
Minneapolis, MN 55403

888.61.RETEK (toll free US)
+1 612 587 5000

European Headquarters:

Retek
110 Wigmore Street
London
W1U 3RW
United Kingdom

Switchboard:
+44 (0)20 7563 4600

Sales Enquiries:
+44 (0)20 7563 46 46
Fax: +44 (0)20 7563 46 10

Printed in the United States of America.

Customer Support

Customer Support hours:

8AM to 5PM Central Standard Time (GMT-6), Monday through Friday, excluding Retek company holidays (in 2002: Jan. 1, May 27, July 4, July 5, Sept. 2, Nov. 28, Nov. 29, and Dec. 25).

Customer Support emergency hours:

24 hours a day, 7 days a week.

Contact Method	Contact Information
Phone	US & Canada: 1-800-61-RETEK (1-800-617-3835) World: +1 612-587-5000
Fax	(+1) 612-587-5100
E-mail	support@retek.com
Internet	www.retek.com/support Retek's secure client Web site to update and view issues
Mail	Retek Customer Support Retek on the Mall 950 Nicollet Mall Minneapolis, MN 55403

When contacting Customer Support, please provide:

- Product version and program/module name.
- Functional and technical description of the problem (include business impact).
- Detailed step by step instructions to recreate.
- Exact error message received.
- Screen shots of each step you take.

Contents

Chapter 1 – Introduction

Welcome to the Retek 10 Integration Bus Deployment Guide. This guide seeks to aid a system designer or a project manager with issues and solutions associated with implementing the Retek 10 Integration Bus (RIB). The RIB is a set of pre-developed EAI components developed by Retek. It contains software deployed within Retek applications and incorporates the SeeBeyond e*Gate Integrator EAI system. This guide assumes a familiarity with EAI concepts and RIB terminology. Readers not familiar with these topics should read the Retek 10 Integration Bus Primer and the Retek 10 Integration Bus Technical Architecture Guide.

Hardware and software base system requirements for the RIB may vary based upon a client's specific deployment. RIB message components are developed with the SeeBeyond e*Gate Integrator platform in mind. See the SeeBeyond deployment guidelines for determining final deployment of the RIB. These guidelines are specified in the SeeBeyond Business Integration Suite Deployment Guide, available from SeeBeyond Technology Corporation. In it you will find information you can use when analyzing, planning, and managing an EAI deployment.

Chapter 2 introduces the RIB schema concept and suggests deployment options to be aware of. For Retek 10 RIB purposes, "schema" is defined as components that are required to facilitate the flow of messages. A schema may include SeeBeyond e*Gate JMS Intelligent Queue Managers, a control broker, and the RIB adapters. ("Adapter" is synonymous with an e*Gate (Multimode) e*Way). The schema discussion in this chapter is focused on the Retek 10 applications RMS, RCOM, and RDM. Alternative, suggested schema designs are also presented.

If a client is considering the integration of external applications to the RIB, Chapter 3 focuses on a five-step process to follow. The chapter continues with an expanded discussion of message specifications when integrating an external application. Retek 10 applications create messages based upon business events. Those events themselves are derived from a convergence of business process and data process. Thus, Chapter 3 focuses on the importance of aligning message 'events' among applications that will share data on the RIB. Other deployment considerations presented here include message representation, sequencing, and transformation and filtering.

System design and development considerations make up Chapter 4. Any deployment of the RIB must take into consideration, at a minimum, currently expected message traffic volume and flow, security, and scalability for future system and transaction volume growth. Incorporated into this discussion are the operating system and SeeBeyond components, collaborations, connection points, and queue locations. Chapter 4 concludes with a summary of best practices and guidelines for deployment planning.

In Chapter 5, the critical issues of system availability and failover are presented. Any deployment would include identification of potential points of failure and strategies for addressing them. Key points of identification are data center location and availability, dynamic data reroutes for hardware failover, and load balancing. Chapter 5 also concludes with a summary of best practices and guidelines for high availability planning.

Chapter 2 – The RIB schema

An e*Gate Schema contains the message flow configuration for a related set of components. This includes the definition of a message's structure and semantic content. It also describes the deployment of the components. This chapter focuses on design considerations for schemas used in deploying the Retek Integration Bus (RIB).

An e*Gate Schema may contain a variable number of EAI components. In one extreme, one schema can be defined that contains all of the EAI components an enterprise uses. In another extreme, each schema may contain only a few EAI components, such as a single publisher adapter and subscribing adapter used within a single message family.

Further complicating this issue is the fact that the e*Gate Enterprise Manager tool may export and import schema components, such as an e*Way, independently. As with other flexible systems, tradeoffs between logical cohesiveness, operational characteristics, software lifecycle concerns and other factors influence the content of a schema. The RIB 10.0 release contains a "Messaging Schema" that contains all Retek adapters for publishing, subscribing, transforming, and routing RIB messages.

The RIB messaging schema

All of the messaging components used in the RIB software are supplied as a single schema. This schema has the name RIB100 and is found in a "zip" file on the CD containing the RIB software and documentation. These components include SeeBeyond e*Gate JMS Intelligent Queue Managers, a control broker, and the RIB adapters. These components work together to provide the integration between Retek applications.

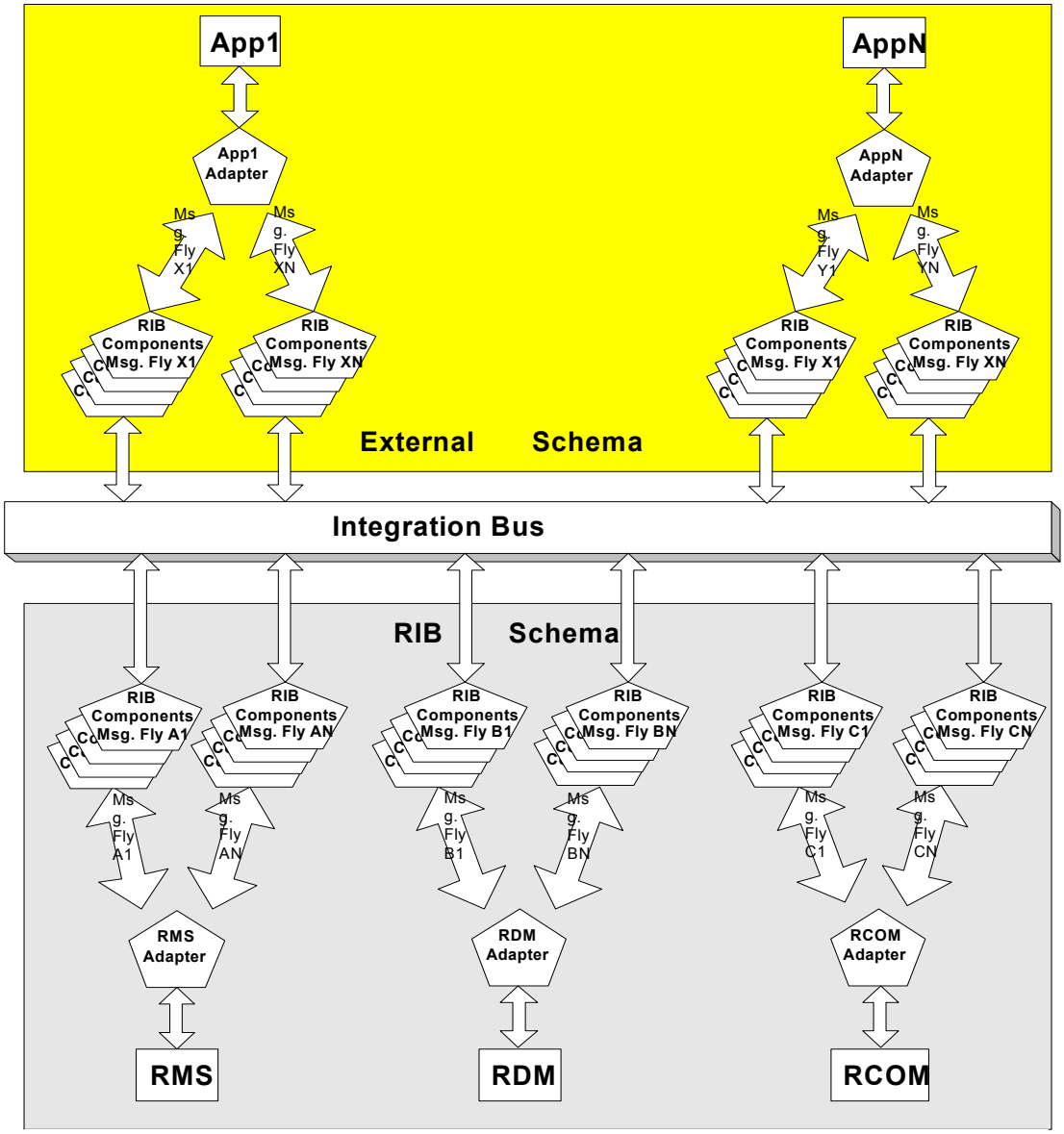
Note: In this document, the term "Adapter" is synonymous with an e*Gate (Multimode) e*Way.

All components within the RIB100 schema are initially configured for deployment to a single host. However, this deployment situation may not be appropriate for some enterprises. The reasons for this may vary and include considerations for a business's organizational structure, the internal network topology, entry points into and out of the EAI system, performance, and availability.

Retek suggests that an enterprise deploy all RIB components within a set of self-contained schemas. When interfacing with applications external to those created by Retek, Retek suggests that these components are placed into one or more separate schemas. Examples of "external" applications include Oracle Financials, SAP, PeopleSoft, or any legacy system. This design allows for an easier installation of RIB updates, highlights the integration points between Retek and non-Retek applications, and allows different availability and performance strategies for each. Limiting one schema to the Retek applications may also allow operations to quickly discern the source of a problem.

Other schema designs may be used when deploying the RIB. The main disadvantages for these lies in the risk that an enterprise's schema during development or testing could affect a Retek component. For example, a configuration parameter "tweaked" for a specific test. This risk is alleviated by having the Retek components in a different schema, since the unit of code migration is usually based on a complete schema.

The suggested RIB deployment framework can be viewed as two (or more) distinct subsystems. Each subsystem is contained within a single schema. An example is shown below.



Msg. Fly A1 = Message Family A1 Msg. Fly AN = Message Family An
Msg. Fly B1 = Message Family B1 …..

Figure 2-2: RIB Deployment Framework

Note: The diagram above assumes the reader is familiar with the term "Message Family". A Message Family is a set of related message formats that are published by a single application. Multiple instances of an application may all publish the same message families, but different applications must publish separate message families. A single application may publish multiple message families.

Any external schema is the responsibility of the enterprise deploying the RIB for its EAI.

Once we logically segregate the RIB and External schemas, we can use one of several schema-bridging techniques described in the *Schema Bridging* section to bridge these the two schemas. An external schema can contain e*Ways that subscribe to some or all of the RIB message families. External schemas can also publish messages to the RIB via schema bridges.

RIB messaging schema deployment

Although the RIB comes with a well-defined schema, there is still configuration and development activities that must take place before the components can be deployed to a production environment. Most of these activities involve component configuration. Under some circumstances, the RIB Messaging Schema may be dispersed among multiple installations of the RIB. The following bullet points highlight the activities needed for correctly installing the RIB for integration among installed Retek Applications:

- **Installed Retek Applications:** The RIB schema is to be designed to ensure proper integration all Retek applications. In the Retek 10.0 release, the RIB is capable of interfacing to the RMS, RDM, and RCOM applications. The specific application needs of each application and the time / dependencies of each should be reviewed in the context of their business processes and the needs for external integration.
- **Message Family Flows:** Retek applications publish and subscribe to "message families". These message families contain operations on a related set of business entities. For example, one message family is specific to Purchase Orders. These families flow between the applications and these message flows must be known for performance and availability considerations, since they affect the deployment of specific RIB components. Message Families are detailed in "*The Retek 10.0 Integration Guide"*.
- **RIB Adapter Configuration Requirements:** Retek applications determine the number and type of application adapters required. All appropriate adapters need to be deployed and configured. Some adapters found within the supplied RIB schema require duplication, configuration, or simple logic changes due to the specific deployment application configuration. For example, messages flowing to and from RDM warehouse systems need to be created for each RDM instance. This customization typically involves configuration changes or simple, well-defined additions to the processing logic. RIB component configuration changes are documented in the *Retek Integration Bus Operations Guide.*

- **Message Flow Customizations:** Due to the variances in the number and deployment of application adapters, processing surrounding the message flow may need to be modified. An installation with applications co-located in multiple warehouses or stores need to insure that each receives messages from the appropriate message families. Such additional applications require modifications to RIB TAFR adapter components and / or the creation of additional bridges and queues.
- **RIB Connection Points:** All RIB publishers, subscribers, and TAFRs require one or more SeeBeyond e*Gate Connection Points. A Connection Point is used to specify a database session or JMS Server session. Messages can be published to a connection point or subscribed from a connection point. A set of connection points (including JMS queues and Oracle database connections) is already contained in the RIB Messaging schema. At the very least, these connection points need configuration changes in order for the system to become operational.
- **Component Placement:** Once the components have been determined, the location where they will execute needs to be determined. If all components are to run within a single computer, then only the participating host's definition needs to be changed. However, if some components are to be placed onto another computer, then additional participating hosts are required. The creation of additional hosts is detailed in the *SeeBeyond e*Gate Integrator User's Guide.*

Message flow customization

A message may need transformation, routing or other manipulation after it has been published by an application and before it arrives at a subscribing adapter. A category of adapters, known as TAFRs (Transformation, Address Filtering/Routing), is available to perform many of these functions. However, additional configuration or logic enhancement may be needed in order for a TAFR to perform the desired operations.

Routing: A TAFR performing a routing operation takes a single message as input and then publishes zero or more messages as output. The number and types of the messages are dependent on the data contained in the input message. For example, a TAFR that routes stock allocation messages from RMS may route the message as a "stock order" to a specific RDM instance.

The mechanism used for TAFR routing involves creating an "Event Type" specific to the stock order message and the RDM instance to receive the message. Because Retek cannot a priori know all possible instances of RDM among all of its customers, this work is delegated to the deployment phase of the RIB.

Additional work may also be needed for routing, since once a "routed" message has been published, it resides on a queue. This queue may be specific to the destination and as such, it must be created as part of the RIB deployment activities.

Remote Data Centers: some messages may traverse from a local system to a remote system. The remote system may have all of its RIB components in a separate schema. If so, a schema bridge is needed to send messages from the local RIB schema to the remote schema. Schema bridges are not part of the RIB Messaging schema and must be added later. Additional queues may also be needed to store incoming messages from remote schemas.

RIB component placement

A simple topology for deploying the RIB Schema is to place all components on a single host. However, this may not be the most efficient and may produce performance bottlenecks.

Placing all components on a single host may be appropriate when:

- **All** database connections used by RIB components are accessible on the same host or via the Local Area Network connected to the hosting computer.
- **All** messages within the RIB have the same availability requirements and these requirements are satisfied by the disk subsystem attached to the computer and fail over mechanisms available.
- **Message analysis** indicates that the host computer can easily handle peak message volume.
- **All application interfaces** can be administered within a single administration domain. Organizational, political and security issues do not conflict with having a single set of roles, users, or privileges under a central administrative authority.

However:

If a RIB component interfaces with a database that is not on the same LAN, then the RIB component should run on a computer located near that database.

If messages on the RIB have different availability requirements and those messages requiring higher availability *cannot* be failed over in adherence to these requirements, then the queues storing this data and the adapters processing these messages should be moved to a host that satisfies the availability requirements.

If message analysis reveals that the host computer cannot handle expected peak load, then it may be appropriate to move some components to another computer.

If organizational, political, or security issues conflict with having a centralized administrative authority, then multiple schemas or even multiple SeeBeyond registries (also referred to as multiple SeeBeyond e*Gate installations or multiple SeeBeyond e*Gate instances) should be used and the appropriate RIB components placed into them. This will also require the creation of schema bridges between the sites.

Schema bridging

An EAI implementation may contain multiple schemas across an enterprise. The reason for this may derive from a desire for a remote site to have complete control over the administration of its systems, to separate logically distinct message families, to provide a high degree of availability, or to for ease of updates.

When a RIB installation contains multiple schemas for processing the same message, the message will need to traverse a bridge between the schemas. This section lists various SeeBeyond e*Gate components and techniques for this to occur.

The RIB Messaging Schema as supplied by Retek does not contain these bridge components. The deployment process used must determine how many schemas are to be used and what types of bridges are to be employed between them.

- **Java Message Service:** The SeeBeyond. e*Gate platform provides a mechanism to use a JMS service provider to bridge between systems. In the RIB 10.0 release, the JMS IQ manager is used, along with JMS connection points for all RIB queues. The motivation behind this in the RIB is the guarantee of "exactly once" message delivery. However, JMS service providers are implemented against a standard that has no connections with the SeeBeyond e*Gate platform. Any application can interface with the service provider as long as it follows the protocol. Hence, any JMS Connection Point in any schema can publish or subscribe to any of the messages published by a RIB adapter.
- **Schema Bridge e*Way:** e*Gate provides a Schema Bridge e*Way that allows a component in a master schema to send messages to components in its slave schema.
- **FTP:** A file based integration may be used where files are FTPd between hosts. Batch e*Ways within each schema act as message publishers and subscribers for sending the data to/from each system. This may be used for systems that have infrequent opportunities for communication.
- **MQ Series:** MQ Series is a *de facto* industry standard for message-oriented middleware. e*Gate has JMS protocol based interfaces to MQ Series. In this interface, a schema component can publish messages to a JMS MQ Series connection point, the same way that it can publish to a standard JMS queue. Please refer to MQ Series e*Way and connection point documentation of SeeBeyond for more details.
- **HTTP/XML:** Another bridging option is the use of HTTP protocols using an XML formatted document interchange. An application server is used as an intermediary, along with a SeeBeyond HTTP e*Way. This could be used for communicating between systems protected by a firewall.
- **SOAP:** SOAP refers to Simple Object Access Protocol. SOAP is increasingly becoming popular, especially in Windows 2000 based e*Gate servers as a means for bridging two schemas across a firewall. Please refer to the *SOAP e*Way Intelligent Adapters User's Guide* for more details on installing and configuring a SOAP bridge.

Chapter 3 – External application integration

This chapter focuses on non-RIB application integration.

Suggested process

This section suggests a process to follow when attempting to integrate an external application to the RIB.

Step 1: RIB message identification and selection

The first step focuses on selecting those RIB messages of business interest for external (non-Retek) applications. The complete list of RIB messages is contained in the *Retek 10.0 Integration Guide.*

Step 2: Analyze pub/sub models

The RIB publishes several message families to convey events in specific business processes. Typically, each business process will have a publication model that describes all the messages generated by different business activities in it. All such messages are often grouped into a message family. A key property of a message family is that its messages have a natural order dictated by its business process.

The following figure illustrates a publication model.

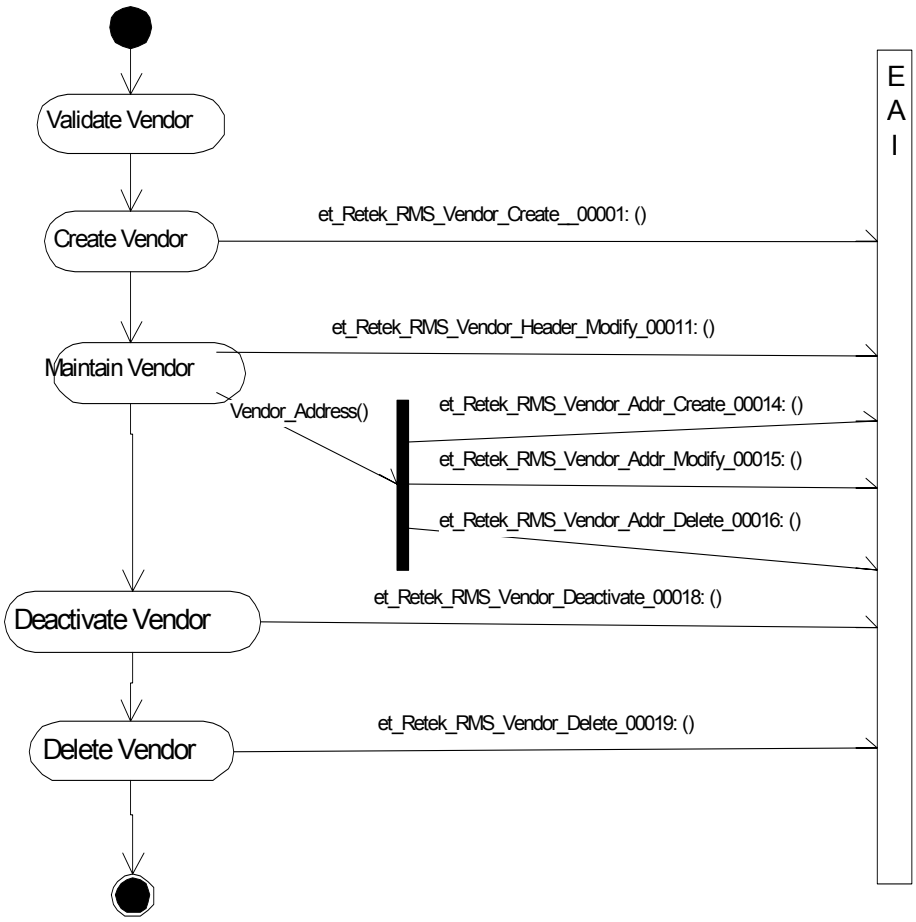


Figure 3-1: Message Publication Model

The message publication model shows not only the events that may be published, but also precursor events. In the diagram above, it is seen that no messages are produced when a vendor is validated, and possibly 4 different types of messages can be published to maintain a vendor. It also implies that a vendor must be created before maintained.

A message publication model is useful in communicating to all subscribers the context of business events (messages) submitted to the RIB. A similar analysis is required to analyze how a RIB message will fit into the business process handled by an external application.

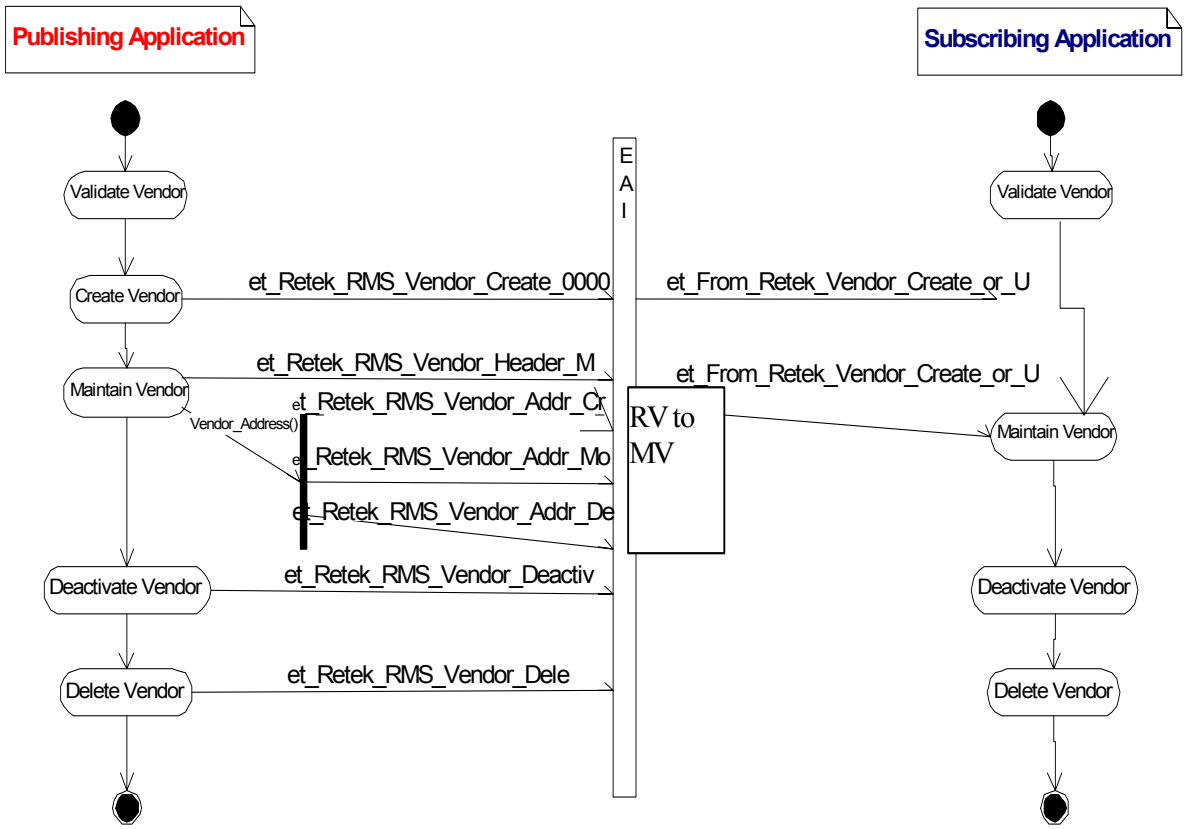


Figure 3-2: Message Subscription Model

In the figure above, it is that the subscribing application accepts only a "Create or Update Vendor message". As such, those messages published as part of the "Maintain Vendor" business process will need to be linked and/or transformed into this format. The diagram has specified that a component named "RV to MV" (Retek Vendor to My Vendor) for this purpose.

Step 3: Application message specification

Application messages need to be specified in light of an application's needs. The specification must include the following:

- The contents of the message – required and option data fields and field types. This is very important since field type incompatibilities can cause serious consequences.
- The structure of the message – whether it is XML, comma delimited, or some other format
- The delivery interface – whether a subscribing application can take messages directly from the EAI bus or if it needs to have a file-based interface.

This specification drives out the ETD, collaboration, e*Way and other integration bus components that the external application interface will need.

Step 4: Message transformations

Unless an external application can understand, process, and create RIB messages directly, some message transformations are needed. The transformation of each message published and subscribed to across the EAI system between the RIB and external application must be specified, analyzed and developed. Although the RIB contains logic of this sort in its set of TAFR adapters, it is expected that transformations between the RIB components and external components will need to be developed from scratch.

One very important input to this process is the message model analysis performed earlier. The RIB uses a very specific message model. This model may not be compatible on a message to message basis with another application. For example, an application may assume that all changes to a purchase order can be published in a single message, while the RIB breaks out changes for a PO header in a message that is different from line item changes. Another possible scenario is that a subscribing application may assume that all PO information is contained in all PO related messages – that is, that each message regarding a PO is a snapshot of the PO. In both cases, once these incompatibilities have been identified, the project plan can be updated with the appropriate task to resolve the issue.

Step 5: Component specification

Once all of the messages and message transformations have been specified, the next step is to determine the components to publish, subscribe, and transform these messages. The considerations for this include:

- Logical cohesiveness of the messages: Messages should be produced and consumed together only if there is some cohesiveness between them.
- Physical deployment concerns: The more pieces involved in a system, the greater the chance that something may break. Also, some messages may have special or non-standard resource requirements that force their deployment into stand-alone adapters.
- Availability and performance: Availability and performance can be maintained in a number of ways. However, these may impact the message paradigm or require additional categories of adapters. These issues are discussed further later in this manual.

At the end of this analysis, a good idea of all of the adapters and queues needed to interface with the RIB should be at hand.

Message specification for external schemas

Message specification is an art as well as a science. The goal is to determine context, scope, structure, content, relationships, and the representation format of each message. The following message aspects familiarize you with some considerations so that you can specify messages properly for an external application. It is worth noting here that a comprehensive and careful analysis has already been conducted in specifying all of the RIB messages

Message semantics & statefulness

A primary difference between application integration using data synchronization techniques (such as Extraction, Transformation, & Loading tools or batch integration) and message-oriented integration of EAI is that the latter is based on aligning business processes. Message subscribers should understand both the contents of a message and the business context that generated the message. In this way, the subscriber can implement the appropriate business logic on the messages.

Message content mapping strategies

This section details strategies defining message contents.

Data Model Mapped Messages: Traditionally, middleware message formats tend to mimic the hierarchy of elements in the data model of publishing applications. However, as any experienced data modeler attests to, data models are not often ideal candidates for message format. A primary reason for this concern is that data models of entities often go through ‘concept creep.’ Over time, the model includes new entity attributes and business constraints that can violate prudent normalization standards or are inconsistent with other business entities.

Middleware message specification is an opportunity to fix any inconsistencies or anomalies in data models of underlying business entities. Message specification provides a higher level of abstraction where the business entities can be rationalized and abstracted away from clumsy cumbersome models.

Canonical Messages: A problem modeling messages from a database model is that the same business entity can have different representations in different applications. For example, an invoice payment relationship might be one-to-one between invoice and payment entities in one application and in another application have a one-to-many relationship. A third application might allow a single payment for multiple invoices. In such a situation, how can we determine a message format that minimizes translation complexity and hides the differences in data models?

A common resolution to this is to use the message format of the application that is designated as the System of Records (SOR) for that business entity. For example, one can generate a message format A′ for a business entity based on the data model for that entity in application A, which is the system of records. From now on, A′ can act as a *de facto* canonical message format. Here, once the translation from message format A (data model based) to A′ is made, that format is broadcast in the EAI message. Each subscribing application, in turn, needs to transform A′ into their native message format. This approach often works when the underlying data models of the entity are commensurate. In other words, the hierarchical message structures in all applications need to be very similar.

A canonical message is an abstract representation of a business event. The format must be compatible with all application specific message formats that are interested in that business event. If the canonical form retains all the hierarchical structures of these application specific formats, then translation efforts as messages traverse back and forth from different applications are reduced. Some XML schema formats discussed below are capable of such generic message representations.

However, if the canonical form contains incompatibilities, then translation overhead can be high. One example of such an incompatibility is the flattening of a hierarchy: that is, one application says that x must contain y, while another says that x and y must exist, but are independent. Translating messages in one direction must create a hierarchy, while translating in the other destroys a hierarchy.

Message representation options

Once the content has been defined, the representation of a message should be determined.

Delimited or Fixed Formats: A simplest message format is a fixed format or delimited text message. An important consideration here is the stability of underlying business entities that generate the message. It is typical for legacy systems or batch oriented systems to use such text based message formats. It is feasible to define a message at any granularity level. For example, we can treat an entire file of transactions or individual transaction as a single message.

XML Messages using DTDs: The RIB uses an XML DTD (document type definition) to specify the format of its messages. This message format allows one to define a document as a hierarchy of elements and their attributes. It is possible to define name spaces and data types. With these building blocks, complex message types can be specified. A limitation of DTD format is that all data types eventually have to be represented as character data. Hence, the publishing application has to ensure that data type constraints and business constraints are adhered before publishing the message onto EAI platform.

XML Messages using XML Schemas: XML Schema formats were specified more recently by W3C consortium. XML Schemas improve DTD format with more comprehensive data type verification and business rules enforcement capabilities. XML Schema format is more ideally suited for developing canonical message forms. It allows for complex functionalities such as <xs:choice> that allows for isomorphic representation of message hierarchies. That is, it is possible for a branch of a message to take one out of a specified list of branch structures.

Additional message format patterns

Within a specific representation, one may use a variety of patterns for the contents of a message:

- **Message Body Only:** A simple design pattern is to convert the message body into an event type definition (ETD) format, which is natively used by SeeBeyond for its internal message representation. Each ETD is specific to a single business event. The ETD fully specifies all fields within the message.
- **Envelope & Payload:** This design pattern hides the complexity of a message into a single element called as payload and wraps it up with an envelope message. The envelope contains several attributes related to message addressing, filtering, and routing. This pattern is useful when the focus is on message sequence or temporal order, rather than on the message content itself. The RIB uses this design pattern. One driver for this pattern is that general-purpose components can be developed that operation only on the envelope.
- **Envelope & Body Fully specified:** A less frequently used design pattern is to fully specify all envelope attributes discussed above along with the hierarchy of elements in the message body. This pattern is best used when the message parsing and processing has a facility similar to the "inheritance" concept used in Object Oriented programming languages. nterest.

External application message sequencing considerations

Within the RIB, the publishing application preserves the business event sequence within the message publication sequence. In other words, messages associated with business events are published in the same order. However, it is possible for messages to arrive out of sequence at the subscribing application if the messages are handled by parallel processing paths. For RIB messages using the Retek 10.0 architecture, this sequence is only guaranteed when a single thread of message processing is used. That is, there is only one path for a message to take from the publisher to any given subscriber.

For non-RIB applications, another solution is to deposit all messages into a staging area for each subscribing application. If messages arrive out of sequence, the API of subscribing application will delay consuming the message until all its predecessors arrive. However, this adds complexity and possible performance delays on the subscriber.

Message transformation considerations

Message transformation can be a resource intensive effort and should be minimized. Some important considerations for message transformation include location, complexity, and frequency of message transformation.

All transformations execute within a SeeBeyond e*Gate collaboration. The following considerations should be examined when determining transformation approach for messages in an external schema.

- **Complexity of Message Transformation:** Transformation complexity stems from dissimilarities of input and output message formats. Canonical formats may be themselves complex. However, canonical message formats decrease transformation complexity in much the same way as an integration bus decreases integration complexity.
- **Frequency of Message Transformation:** To maximize performance, minimize the number of times a message needs to be transformed. One aid in this is to base the message format on the data model in the application designated as the “system of record”. One advantage of this approach is it promotes a common view of the business entity in question.
- **Location of Message Transformation:** A message can be transformed anytime between publication and final the subscribing system. The location of the transformation can be based on the relative number of publishers and subscribers and the specific message pathways. When there are more subscribers, it makes sense to transform a message into common format near publication. When there are more publishers, it makes sense to transform a message near the subscription. If messages are multiplexed through multiple pathways, a transformation adapter may make sense located midway in the processing stream.

Message filtering considerations

Sometimes a message or its attributes might contain a well-specified range of values. When such message is published frequently, its subscribers are forced to process the message quite often. However, in some situations, the subscribing application might be interested in a business event only if a specific field falls within in a narrow range of values. For example, a purchase order application can publish PO status whenever any PO changes its state. If the publication of the other messages is quite high, it may be fruitful for load balancing or performance reasons to separate the filtering logic to a different adapter. The filtering component on RIB can prevent the deliver of PO status messages to that subscribing application. In one sense, filtering is a special case of routing.

The location of filtering should be based on considerations such as network bandwidth, application complexity, the message publication and subscription models, and future integration plans. Does it make sense to insert filter logic in an adapter that will later be stripped out? One may gain more flexibility by developing a separate (TAFR) adapter for filtering. This also has the advantage of allowing greater flexibility if these messages need to be re-filtered: it could be performed by a simple configuration change of the TAFR adapter, as opposed to a logic change within the subscriber.

Event linking & sequencing

Sometimes an application can publish messages at level of granularity that is not acceptable for consumption by one or more subscribing applications. For example, a purchase order (PO) application can publish separate messages for PO header and line items. However, a subscribing application might require a full PO with line item details. e*Gate's Event Linking and Sequencing (ELS) component enables linking and sequencing of events. Considerations for this process include:

- **Part-Whole Relationship of Messages:** As the PO example above illustrates, when message granularities are different, ELS provides an elegant mechanism for holding messages until some conditions are satisfied for composing one or more higher-level messages from these lower-level messages.
- **Message Synchronicity Considerations:** Even when message granularity is same for publishing and subscribing applications, ELS can be used to sort messages and deliver them in proper chronological order to subscribing applications. This is effective if all messages are guaranteed to arrive at the subscriber. Messages that are filtered or routed to different destinations may cause problems when ELS is used in this fashion.
- **Batching Messages:** In some instances, an application can collect all messages published from an application, and then coalesce these messages periodically. For example, a medical billing application can publish claim messages as they occur. An ELS enabled adapter can hold all such messages for a predetermined period (say an hour) and aggregate the messages into a batch file, which then can be transmitted to a subscribing application.

Currently, the RIB does not use any of SeeBeyond's Event Linking and Sequencing facilities.

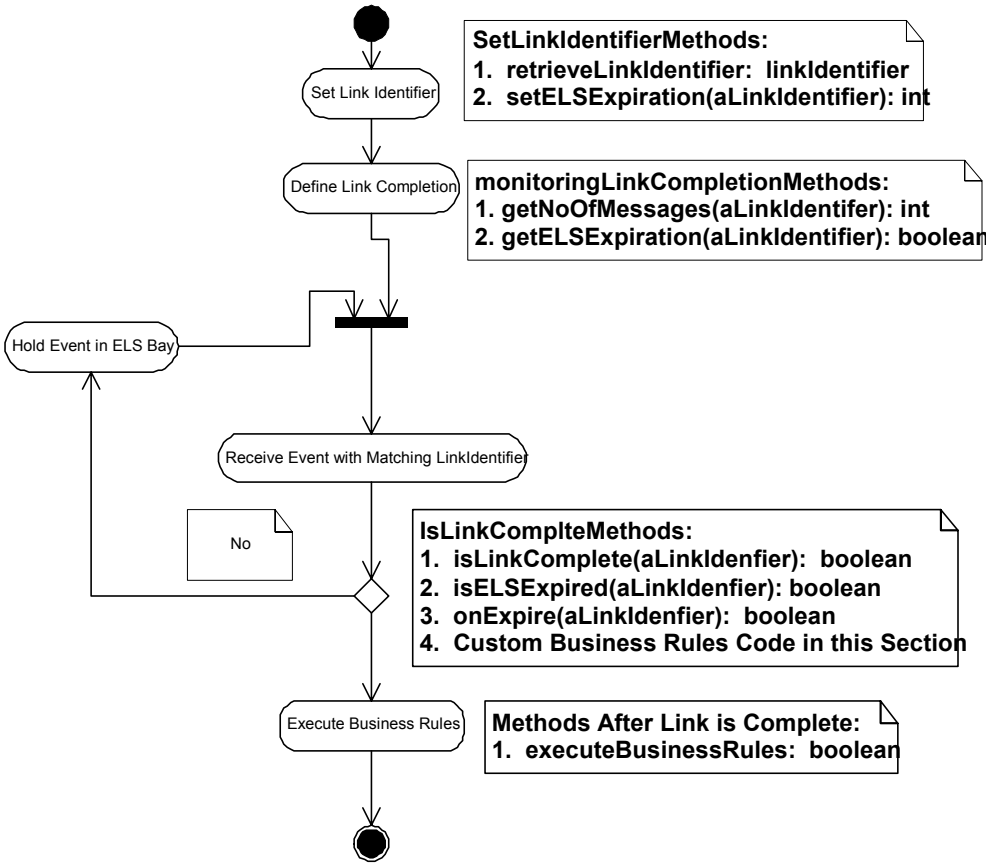


Figure 3-3: Activity Chart of an Event Linking & Sequencing Class

Overview of the ELS

The ELS component has to set a LinkIdentifier and define criteria for ELS completion. When any message with a matching ID to the LinkIdentifier arrives, it holds that message in a linked list that is also saved to stable storage (disk). All such messages are held until a message arrives that indicates the completion of ELS criteria. After that, the executeBusinessRules function is executed and tasks such as message sequencing, aggregation, and batching can be completed.

Typically, ELS is meant for linking events over finite duration such as a few days. It is memory intensive, depending on the number of events and messages linked in a production system.

Chapter 4 – Systems design and development

The purpose of this chapter is to describe the considerations and design patterns involved in developing systems design and development processes for successfully deploying the Retek Integration Bus and the underlying SeeBeyond e*Gate platform. In general, an EAI platform has to be scalable and flexible to meet the increasing application interface needs of a firm. This chapter describes a process for systematically developing architecture for the physical design of the EAI infrastructure.

The design effort starts with business and technical requirements. Among these are the applications to be integrated, acceptable latencies for message traffic, and security concerns. Furthermore, these requirements should include future transaction growth estimates arising from additional participating applications and additional integration points within the deployed applications. All of this information then establishes broad parameters for systems design.

For example, an expectation of substantial growth in the transaction volumes handled by a participating application requires careful review of the proximity of application servers to EAI servers or the network bandwidth between those systems. Similarly, when new types of applications are added to EAI platform, they can put substantial performance demands on EAI platform: Web based applications typically require near real-time response rates. The ability to scale up components rapidly to maintain acceptable response times becomes important.

Systems design process overview

The first step is to determine all components involved in the EAI system based on the messages needed to support the business processes. The next task is to design the proper topology of e*Gate components to meet the performance needs of the EAI system. The issues presented in this chapter are discussed in a “top-down” fashion, beginning with domain considerations and ending with log file location considerations.

One begins with a map of all administrative domains within the enterprise in which the integration components can potentially be installed. An important consideration is the geographical scope of corporate network – whether the platform will be located entirely within a data center or dispersed over several data centers.

Typically, the location of associated application servers determine the data centers involved. Identify the network domains in which application servers reside and the domains in which the integration bus platform resides. Then determine the network bandwidth among the domains involved in the integration. An additional consideration is the identification of security requirements and firewalls, which can restrict the available architectural options. This analysis is also important for disaster recovery planning needed to resurrect operations in a new data center location after a catastrophic environment failure.

A popular design pattern for server domains is to centralize them into a single domain. Many times the EAI development team is a centralized resource. EAI teams are responsible for meeting Service Level Agreements related to an EAI platform with all application or functional teams. Furthermore, an EAI team can be a central repository of knowledge. Their EAI expertise is critical for data center network administrators, operating system administrators, and SeeBeyond administrators for properly configuring, scaling up, and troubleshooting the EAI platform. Hence, physical proximity of data center team and the EAI team is beneficial for building an EAI infrastructure rapidly, especially in the early years of EAI adoption by an enterprise. Even if physical proximity is not feasible, close working relationship (such as job rotation or joint design teams) should be fostered between data center team and EAI team to achieve proper performance of EAI platform.

Operating system selection

The selection of an operating system platform for the EAI infrastructure is important for performance and scalability. The SeeBeyond e*Gate platform is supported in several platforms such as Solaris 2.8, Aix 4.3, HP/UX 11, Linux 6.2, Compaq True64, and Windows 2000. The only requirement for e*Gate connectivity is TCP/IP compatibility. Hence, it is possible to implement e*Gate in Windows 2000 even when applications are on a Unix platform, or vice versa.

Some factors that affect the choice of the OS for e*Gate platform are the availability of High Availability cluster software, scalability of servers within a cluster, limitations on processes/threads, and limitations on memory scalability. See the *OS Considerations* section for more information.

The e*Gate client and its Graphic User Interfaces (e*Gate Enterprise Manager, e*Gate Monitor, e*Gate IQ Viewer, and e*Gate Alert Agent) run exclusively on Windows 2000 operating system. Operations personnel who are monitoring the running system use these tools.

The development server operating system platform can be completely different from that of production environment. For example, the development environment can be in Windows 2000 environment and the production environment in AIX. The task of migrating schema code from one operating system to another is a routine administrative task that can be accomplished in hours.

See the *SeeBeyond e*Gate Deployment Guide* for minimum hardware specifications for e*Gate server on each platform and e*Gate client on Windows 2000.

OS considerations

A key to designing e*Gate systems is maximizing the performance of the operating system hosting an e*Gate server. Many times performance gains are achieved simply by increasing memory and CPU processing speed. However, this is only appropriate if the system is fully utilized. Two important operating system resources are the number of processes and threads. A system that is not supporting an adequate amount of processes or threads may not be fully utilized.

Process considerations

A number of e*Gate components such as e*Ways, connection points, and IQ managers execute within a single Unix process or Windows executable. For complex schemas, these components can number in the hundreds. Computing the number of processes needed by the system may be found by counting the number of components (including the "infrastructure" components such as control brokers or the registry) that are deployed on the system by all schemas in the installation.

Thread considerations

Some e*Gate components, such as IQ Managers, spawn hundreds of threads for their proper functioning. As the number of components, especially IQ managers, increase in a schema, the number of total threads used by e*Gate increases as well.

Some operating systems have a limit on the number of threads that can be spawned per process. If the number of threads per process is beyond the permissible OS limit on threads per process, we need to redesign the components to reduce this.

Note: RIB schema components usually do not need this reduction because most of the RIB adapters (e*Ways) do not contain many collaborations. The philosophy here was to deploy fewer, message family specific processes as opposed to many concurrent threads. This allows for a better fine-grain control of each message producer and consumer.

This can be accomplished in several ways. First, the number of collaborations per e*Way can be decreased by adding a new copy of e*Way and moving half of collaborations to the new e*Way. This keeps the total number of threads same but decreases threads per each process. It is suggested to keep the number of collaborations under an e*Way from six to ten. For JMS IQ Managers that push the threads per process limit, one may add more JMS IQ Managers and redistribute the messaging load between them.

Component considerations

This section describes some additional considerations for each SeeBeyond component type.

BOBs and e*Ways: An e*Way is an application specific adapter that fully supports invoking all published Applications Program Interfaces (APIs) of that application within the collaborations attached to it. A BOB (Business Object Broker) is an internal SeeBeyond container for holding collaborations. The RIB only uses e*Ways.

The biggest question for is the number of copies and their location. Each e*Way or BOB will process a stream of messages from an application or an application's database. The location of an e*Way is dependent on the availability and performance of its information source or sink: the application or the database the application uses.

The number of copies of each e*Way is determined by the amount of parallel processing desired. In the RIB 10.0 release, sequencing considerations curtail the options here for RIB components. However, these concerns may not apply to other interfaces to external applications. If parallel processing is implemented, then one copy of an e*Way per parallel processing path can be used.

Collaborations and parallel processing

A collaboration is the basic programming unit for a work-slice (message processing logic) in e*Gate. Within the RIB components, collaborations have already been defined and located.

Collaborations have to be deployed and executed within an e*Way or BOB. An e*Way or BOB is an operating system "task" or "process". A collaboration is run as a thread under that process.

The number of copies of collaboration depends on the transaction throughput required. Incremental throughput improvements can be achieved by adding copies of collaboration to the e*Way. This can improve performance by parallel processing. The intent here is that system idle time due to disk I/O is reduced when multiple copies of the same collaboration are simultaneously processing messages.

Connection Points

Connection Points are protocol based communication service provider components. They create a session with some external entity. For example, the Oracle connection point establishes a database connectivity with an Oracle database and the MQ JMS connection point establishes connectivity to an MQ Series server. Only one connection point per schema is needed for a communication service for an application integration point. However, it is not unusual to have one connection point for outbound connections, one for inbound connections, and a third for special purpose connections (such as error handling or control).

Queue storage location

A common design pattern is to locate queue storage along with other schema components on the same server. This pattern increases performance by decreasing network overhead.

However, locating the queue storage on a separate server from schema components is useful for high availability situations. An advantage here is that the queues continue to be available even when the server containing other schema components fail. Furthermore, an IQ manager is a fairly robust component that handles routine database transactions without any user intervention. The probability of the failure of a server handling IQ managers alone is quite low compared to that of a server housing complex application specific code.

A common practice is to locate queue storage on networked enabled RAID disks, so that message data continues to be available even when an e*Gate server fails. The RAID disks are typically configured in RAID 1 + 0 (that is, both mirroring and striping). RAID 5 configuration can be equally effective as messages with in IQs are rarely updated, thus minimizing any update penalties possible in RAID 5 configuration.

Log files

SeeBeyond uses various log files during the recovery of e*Gate schema after a server failure. Logs have to be stored with same care as data within IQs. A popular design pattern is to locate logs on external RAID boxes using the same configuration as that of IQs. If log files are available during a fail-over operation, then some messages may be lost.

Topology and performance considerations

The key to performance improvement of e*Gate is to fully understand or map the source and destination of messages across all applications. The location of servers and schema components affect the distances these message travel. The overall objective of topology design is to minimize the overall network travel path of messages. This is especially important for high volume messages. The topology design has to focus on minimizing travel distances of high volume messages.

Best practices & guidelines summary

Minimize message and database connection path length. Locate RIB adapter e*Ways and external application e*Ways on the same LAN as their respective database servers. Locate e*Ways belonging to the same message stream on the same LAN if at all possible.

Limit the number of collaborations per e*Way to between 6 and 10. The RIB uses only 1 or 2 collaborations per e*Way.

Limit the number of IQs per IQ Manager to 6 or under. (Not applicable to JMS IQ Managers).

Locate Queue Storage on disk systems that can failover along with the e*Ways or other schema components. Otherwise, messages may be lost until the system has failed back.

Locate log files on disk systems that can fail over along with the other e*Ways or other schema components. Otherwise, there may be database recovery or lock problems.

Understand your schema design. Retek suggests that a “RIB Messaging” schema is used that contains few, if any, external components. However, whichever schema design is implemented, make sure the motivations and risks behind the design are understood.

Analyze the number of processes and threads needed for a running system. Make sure that these values are configured as part of the operating system.

Chapter 5 – High availability

SeeBeyond's distributed architecture addresses the needs of high availability and robust failover, while providing inherent scalability across all components. The key for High Availability (HA) deployment is to identify and eliminate any single point failure, both in terms of software components and hardware components.

High availability solutions are typically complex, involving redundant software, hardware, and network resources. One or more of the following e*Gate product features described in this section can be used to deploy Retek Integration Bus (RIB) in a "High Availability" configuration.

A major goal in deploying any EAI platform such as the RIB is to strive for 24 x 7 availability. Any prolonged outage in the availability of EAI bus can lead to adverse consequences. When some parts of RIB are unavailable, RIB itself continues to receive events and store them in IQs till the required components are available. However, prolonged outage can create a cascade of failed transactions when IQs are filled up.

E*Gate provides several ways of providing High Availability by enabling automatic failover. Key components in e*Gate that provide this HA capability are:

e*Gate Registry: This directory contains all information related to the run time environments of all the components registered on a deployed e*Gate system. All participating hosts authenticate with the Registry using their schema name and the logical name of their control broker. The registry keeps track of mapping between logical names of control brokers and their physical addresses on the bus. Thus, when a participation host is unavailable, a new participation host can be attached to the network with its control broker taking on the logical name of the control broker on the failed host. Such changes in physical address have to be updated in the Registry to bring up the new host in place of the failed participation host. However, this requires replication of the Registry itself on more than one participation host, so that this information is always available.

Registry Replication: It is important to replicate Registry information to two or more participation hosts to ensure its recoverability, without which e*Gate itself cannot be recovered. One of the participation hosts can be arbitrarily selected as the Primary Registry. Several Secondary Registries can be placed on other Participation Hosts on the network. Whenever a change is made to the Primary Registry, it automatically updates all the secondary copies in real-time to maintain consistency. Components initially request information from the Primary Registry. When that is unavailable, it goes through the list of Secondary Registries till it finds one that is available.

Subscriber Pooling: One option configuring JMS Connection Points is whether to use “queue” or “topic” connections. If “queue” is used, then only one copy of each message is delivered any of the topic’s subscribers. In this design, it is assumed that all subscribers perform the same processing. This allows for parallel message processing, since multiple messages can be processed simultaneously. Furthermore, if the subscribers are distributed on separate boxes, then messages can continue to be processed as long as the queue and at least one subscriber is available. This technique is known as Subscriber pooling and is also available with the standard SeeBeyond IQ implementation.

However, subscriber pooling introduces the possibility that multiple subscribers will process dependent messages in a slightly different order than when they were produced. Consider the following scenario: two subscribers process PO Item update messages and there are two consecutive messages updating the same item. The first message updates the item quantity to 10, the second one updates the quantity to 100. One subscriber is on a heavily loaded system and the other on a lightly loaded system. The subscriber on the heavily loaded system finishes some prior work and grabs the first message. The lightly loaded subscriber then grabs the second message but is able to lock the item database record before the first subscriber. (The first subscriber is running on a slower system.) Now, the first message cannot be processed until the second message has finished its processing. This means that the final quantity is left at 10, not at the last updated value of 100. Because of this and similar scenarios, **configuring JMS connection points using “queue” connections is not recommended.**

Remote data center considerations

The IT infrastructure of an installation might include several data centers geographically dispersed. In such a situation, network bandwidth plays an important constraint in RIB processes such as Registry replication.

Identification of data centers

The process begins by taking an inventory of data centers where e*Gate hardware will be located. Locating all e*Gate servers in a central data center facility enables easier administration and tighter physical security. However, the response time for transactions might be hampered in such an architecture with network propagation delays.

The next step is to consider if the e*Gate infrastructure will be dispersed among two or more remote data centers as well. An important consideration here is the network bandwidth between each data center pair. The location of fail over systems and other attended processes will only be successful if message traffic can be physically communicated to the components providing this capability within allotted response times. If the bandwidth is not sufficient for all messages, then it may be necessary to only provide partial fail over capabilities: messages deemed non-critical will stay in queues or have their publishing adaptors shut down.

Disaster recovery considerations

If the client has an alternate data center for disaster recovery, the disaster recovery data center needs to be equipped with sufficient e*Gate hardware and software infrastructure for recovery of the e*Gate platform. Network bandwidth analysis should also be performed, treating the disaster recovery center as another remote data center.

High availability option considerations

One primary consideration for determining the HA architecture is to insure that normal operations are not adversely affected. Another consideration is to fully utilize available hardware platforms. A third consideration may be the likelihood of multiple failures. It is important to understand the trade-offs and the risks for any specific deployment.

The following sections present two alternatives for implementing High Availability configurations. Care should be taken in evaluating these options regarding the following:

- The amount of manual intervention required when a host failure occurs.
- The amount of tolerable down time associated with a failure.
- The complexity of the deployed system.
- The development of special components to insure that messages are processed in the correct sequence, or reduce the risks if certain messages are processed out of sequence.

Hardware preparedness for HA

HA architecture involves replicating schemas across two or more servers. Schema components such as a JMS connection point or Oracle ODBC connection point are often configured to connect with a named application server. A DNS server or an Oracle Names Server resolves the domain names into physical IP numbers. When a server using a physical IP as its domain name fails, all of its schema components have to be reconfigured to point to the new IP number of the fail over server. To avoid such manual intervention and the need to edit the configuration files of schema components, we need to make sure that domain names resolve into logical IP addresses, where we can attach new servers to take over the logical IP address dynamically and seamlessly continue e*Gate processing. We can allocate IP addresses dynamically to servers using the means described below:

Hardware IP Routing: A network router, such as Cisco Local Director or ArrowPoint load balancer, dynamically multiplexes incoming e*Gate events to its attached servers. All the servers in this configuration share the same IP number. Thus, when a server is attached during failover, the new server has the same IP number as the failed server. This ensures that we do not have to make any changes to configuration files of e*Gate schema components.

Clustered IP Routing: We can configure a clustered server network to share IP numbers among its server nodes. The cluster management software is mounted on a shared file system and this storage box typically contains its own Ethernet cards. In addition, the individual server nodes can contain private file systems and private Ethernet cards that allow direct IP addressing. For configuring e*Gate under HA mode, we need to rely on the shared IP numbers of the cluster for receiving events. In this mode, when a new node is added to the cluster, it can automatically take over the shared IP number of the failed server node as all nodes are using the cluster's shared IP number.

Advantages of clustered IP routing

This document suggests the use of clustered servers managed by an HA cluster management package that detects failures at the operating system level. IP routers are not natively geared to recovering the failed server. When an IP router finds a server non-responsive, it merely routes the incoming packets to the next available server. IP routing by itself does not initiate any failover processes. In contrast, HA clustered software, such as Solaris HA or IBM HACMP, provides automatic detection of server failure and initiate recovery by invoking a recovery script.

Hot standby data center/server

If a deployment has multiple data centers, one may use one of them as operational data center. The remaining data centers will not participate in e*Gate processing as long as primary data center is operational. When primary data center is not available for some reason, one of the remaining data centers will be designated as e*Gate operational data center and takes over e*Gate message handling.

In a clustered Hot Standby option, a cluster of two identical servers is deployed. Only one of them is in production mode at any time. When the production server fails, the standby server takes over e*Gate processing.

Some positive factors that favor Clustered Hot Standby Sever option are as follows:

- HA clustered software can only be used for hardware recovery, that is, an alternate server takes over the IP number and functions of the failed server. The processes and state information in the failed server are lost. At this time, e*Gate does not have HA software layer that keeps track of such state information during failover, so active components will be re-started without any state carryover.
- Shortened recovery periods during failover.
- A cluster's shared disks enables data consistency between the Production Server and the backup server.
- Schema complexity is relatively low compared to load balancing options.
- Scalability and Increased processing power is feasible by adding additional servers.
- Uses the distributed architecture of SeeBeyond e*Gate to a limited extent.
- Preserves message sequencing.

Some negative factors that need to be addressed in Clustered Hot Standby Server option:

- Recovery complexity can still be high.
- The processing power of a powerful Hot Standby Server is either not utilized or is used for some other purposes. If the latter case exists, then this other processing may need to be shutdown before the failover can commence.

- Does not fully utilize the distributed processing architecture of e*Gate platform.
- The failover recovery process might take 10 minutes or more. This depends on the time it takes to recognize that a failure has occurred. Some HA packages by some vendors allow the monitoring of specific running components. If these are used for monitoring e*Gate components, these scripts may go through a series of health checks to bring up failed e*Gate modules before declaring the Production Server as having fatal error. During these monitoring and health checks, the EAI messaging service is not available to applications.

Clustered data centers/servers with distributed load balancing

In this architecture, each server node has multiple copies of all schema components to increase CPU utilization and transaction throughput. All queues are configured using pooled subscribers. Availability is maximized using a "server farm". When a server goes down, all the copies of schema on that server might become unavailable, which can decrease transaction throughput. However, as long as there is at least one copy of a functioning schema on one host in the server farm, messages may still be processed. This option is similar to a Subscriber Pool off of an IQ Manager, except that the subscribers are all within different schemas.

This configuration contains some very positive features, such as significant uptime, scalability, and server utilization. However, there are also some significant drawbacks to this approach, including:

- Complexity of the distributed architecture.
- e*Gate schema will be complex to allow load balancing through subscription pooling.
- Additional programming effort to enable subscription pooling of all active components such as collaborations and BOBs in each node. Typically, 50% of additional programming effort is required for accomplishing load balancing of multiple nodes, compared to the programming effort involved in developing a schema for a node.
- Issues with sequencing and error handling. The RIB's current design assumes that all messages within a single queue are consumed in order. Multiple simultaneous subscribers may violate this. In the 10.0 release of the RIB, all messages within a single message family need to be processed in a single-threaded manner to insure correctness. This is intimately tied into error handling subsystems. Multiple copies of the same publisher or subscriber could cause "out of order" problems unless additional installation specific development is performed.

Note: A 'Message Family' is a set of messages published by a single publishing collaboration. See the Retek Integration Bus Technical Architecture manual for more details on message families.

Because of potential problems with sequencing and error handling during the normal operation of the RIB, this option is discouraged unless a thorough analysis has been performed regarding these issues has been performed and the appropriate risks addressed.

Performance considerations

In order to select an appropriate HA architecture, we need to define selection criteria properly. Some important performance factors for selecting a HA architecture include transaction throughput, infrastructure cost, and recoverability.

Transaction throughput

To improve performance, it is possible to extend the capabilities of the server by increasing its processors, processor speed, memory, and network bandwidth. However, performance gains may be limited due to the nature of any specific performance decreasing bottleneck. For example, if the specific disk I/O sub-system used is slowing the overall system performance, then adding additional CPUs or upgrading the existing CPU to a faster model will not increase performance. Alternatively, options such as moving a schema to a new host, distributing the components in a different configuration, or improving the disk I/O subsystem used may result in the desired throughput.

Infrastructure cost

The infrastructure cost is dependent on the number of servers in an HA cluster and the specifications of each server node. This has to be computed based on transaction volumes, number of schemas, transaction processing time, memory, IQ size, archiving needs, CPU count, CPU speed etc.

A major consideration here is the choice between deploying a cluster containing a few high-end servers versus a cluster containing a large number of mid- or low-end servers. High-end servers can have more memory and processing speed. However, the performance of e*Gate is dependent on the number of operating system processes and threads, which can be maximized with a large number of nodes in a cluster. Hence, in many cases, it is more cost effective to use a cluster of larger number of mid- or low-end servers rather than a smaller cluster of high-end servers.

Recoverability

The ability to recover after a system failure is the primary reason for HA architecture. During failure, certain transactions can be abandoned after partial processing. Unless these transactions are rolled back properly, there is a danger that transactions are dropped without proper delivery or that non-idempotent operations are performed multiple times.

Definition: *Idempotent operation:* An operation that can be performed multiple times with the same result as the first time it is performed. Setting a thermostat to 70 degrees Fahrenheit is idempotent. Setting a thermostat 5 degrees warmer is not idempotent.

The EAI bus has to guarantee "exactly once" processing of an event by all subscribers. The complexity of transaction recoverability is similar in all architecture options described in the previous section. The essential steps involve detection of a failed server in the HA cluster management software layer and invoking a recovery script. This script must insure that all standard and JMS IQ managers are brought up under the control of the fail over server and partial transactions are rolled back.

Best practices & guidelines summary

This section describes several design patterns that have been successful in designing HA architectures.

Failover of registry using replication

The e*Gate Registry provides directory services for all the components on the EAI bus. To prevent registry becoming a single point of failure a preferred design pattern is to have a registry in each server. This will also minimize network roundtrips for information that components such as the Control Broker need. One of these registries can be designated as primary registry and remaining secondary registries can be replicated automatically. If there are frequent changes to registry, such as relocation of component schemas across servers, replication can increase network traffic. Since changes to registry are infrequent, replication related network traffic is often insignificant.

Use a high availability clustered software

The use of HA cluster management software is highly suggested. The cost of HA cluster management software is minimal compared to the hardware cost of servers in a cluster. A major advantage of the HA cluster management software layer is the detection of server failure. The interrupt from such HA cluster management software can be used to trigger a recovery script to initiate and complete proper recovery tasks. In addition, the presence of a shared file system in a cluster allows all servers to share same resource libraries etc., which eliminates version incompatibilities that might arise when servers have their own resource libraries in their private file systems.

Failover of IQs with external RAID

As long as events are in IQ, transactions are guaranteed to be recoverable. However, when the file system used by IQ itself fails, there is a severe danger of transaction loss. A preferred design pattern here is to achieve hardware failover of IQs using external RAID banks. It is typical to configure IQs under RAID 1+0 (mirroring and striping of IQ data).

Failover of e*Gate Monitor

It is important to ensure that e*Gate Monitor does not become a single point of failure. The e*Gate Monitor application needs to be installed on two or more systems so that failure of any single host does not remove the ability to control the operations of the RIB.

Illustration of HA architecture with clustered hot standby

This HA architecture requires two node cluster managed by HA cluster management software. One of the nodes is operational and other is in hot standby mode.

Schematic overview

The figure below shows the schematic overview of Clustered Hot Standby HA architecture:

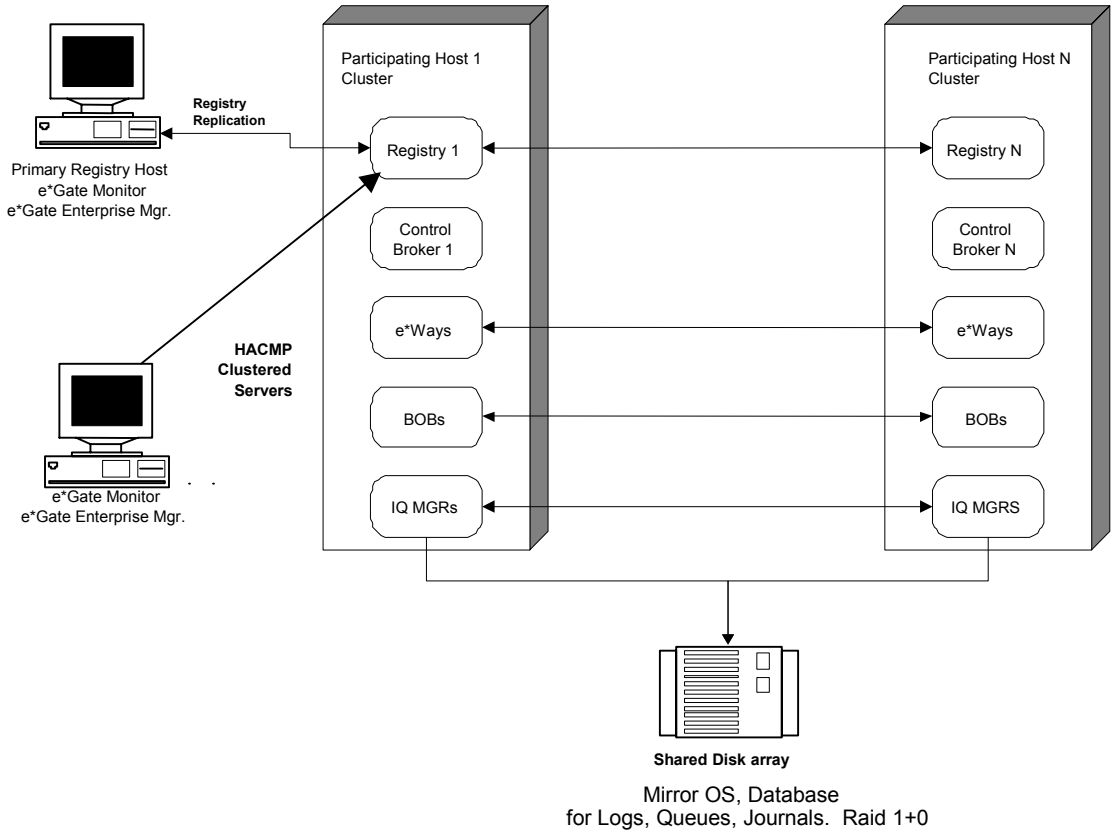


Figure 5-1: Schematic Overview of Clustered Hot Standby HA Architecture

Deployment diagram

The deployment of e*Gate components for the clustered hot standby architecture is shown below:

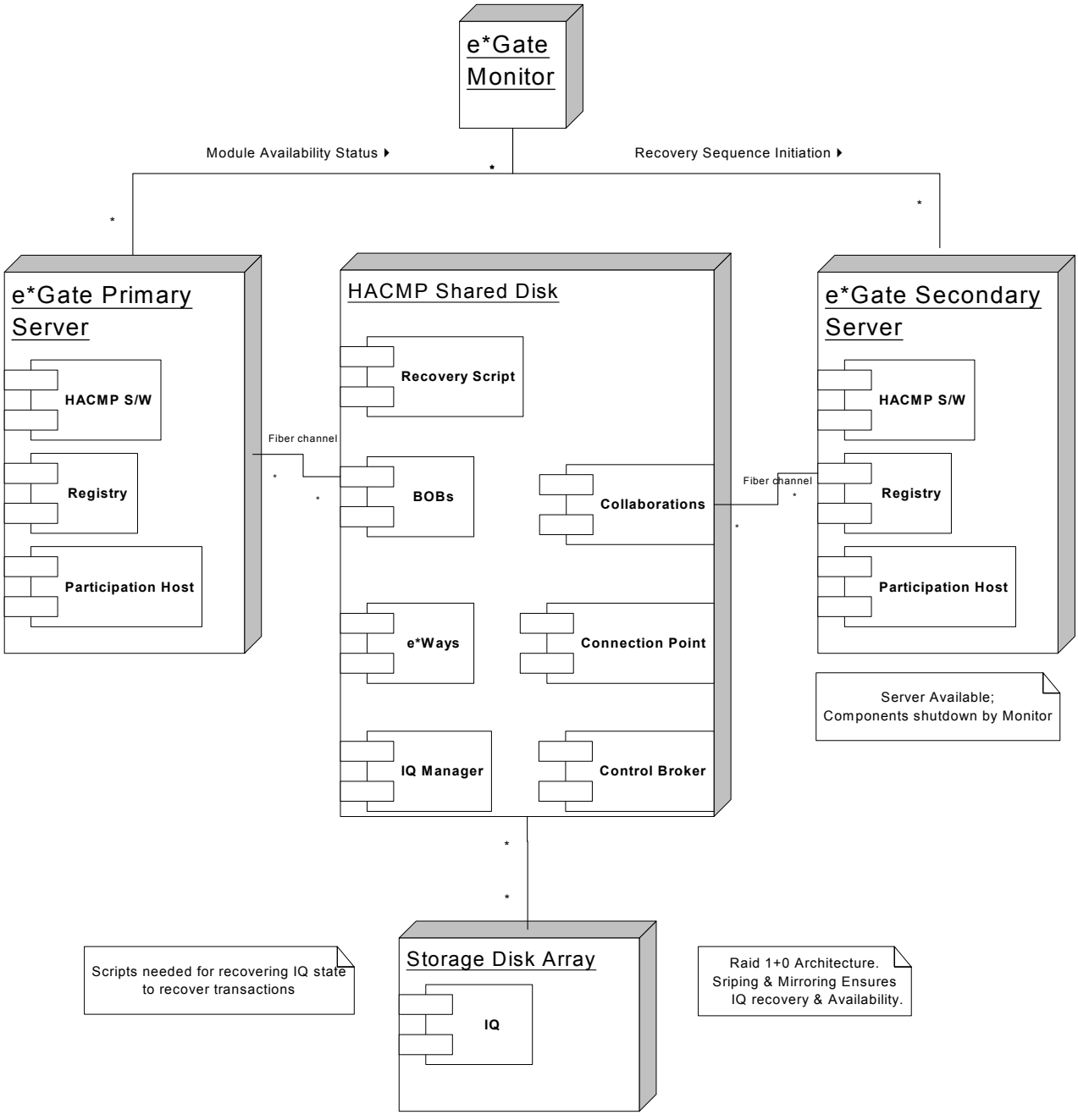


Figure 5-2: Deployment Diagram of Clustered Hot Standby HA Architecture

Appendix A – Parallel processing deployments

This appendix describes methods of configuring e*Gate with multiple copies of e*Ways for the purposes of increasing throughput or providing for seamless and immediately available fail over. However, these topologies are not suggested with currently available RIB components due to risk that messages may be processed out of order. Additional custom development may be needed for RIB components if some type of parallel processing is needed.

Subscriber pooling

Subscriber pooling is a technique for parallel message processing. When an SeeBeyond Standard or JMS IQ Manager is configured for Subscriber Pooling, then all messages on its queues are delivered to a single subscriber. However, in this design, it is assumed that all subscribers perform the same processing. The intent is for improving throughput using parallel processing.

This is not recommended for standard RIB messages and components because of the risk that messages will be processed out of order. This restriction may be lifted in later releases. However, for other, non-RIB or non-sequence dependent messages, subscription pooling may make sense. Because of its utility in other areas, this appendix provides an example of how subscriber pooling may be used.

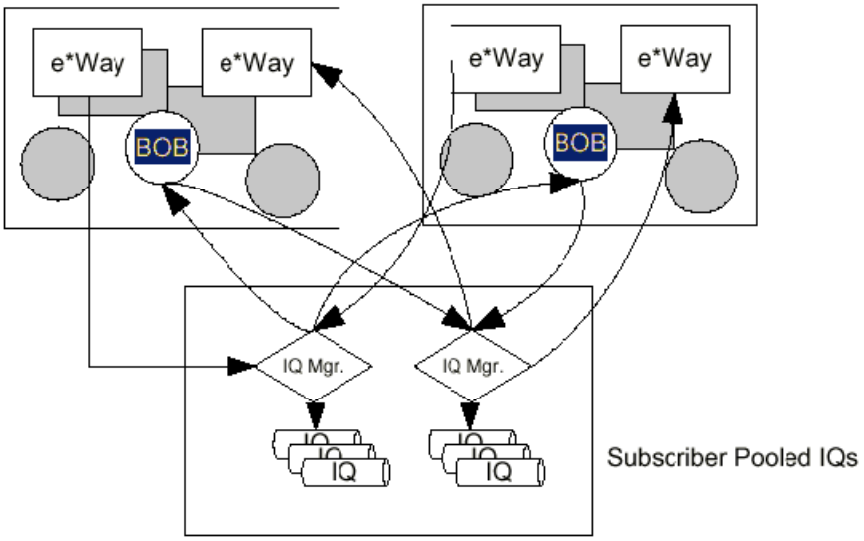


Figure A-1: Subscriber Pooling

This example has three host machines, and two of them are using e*Ways to process messages. The third machine is for message queuing only. The subscriber components are redundant and run from both host machines. The IQs are subscriber-pooled so their operation is load balanced across software processes and also across host machines.

The second machine takes over processing seamlessly from the failed server as shown below:

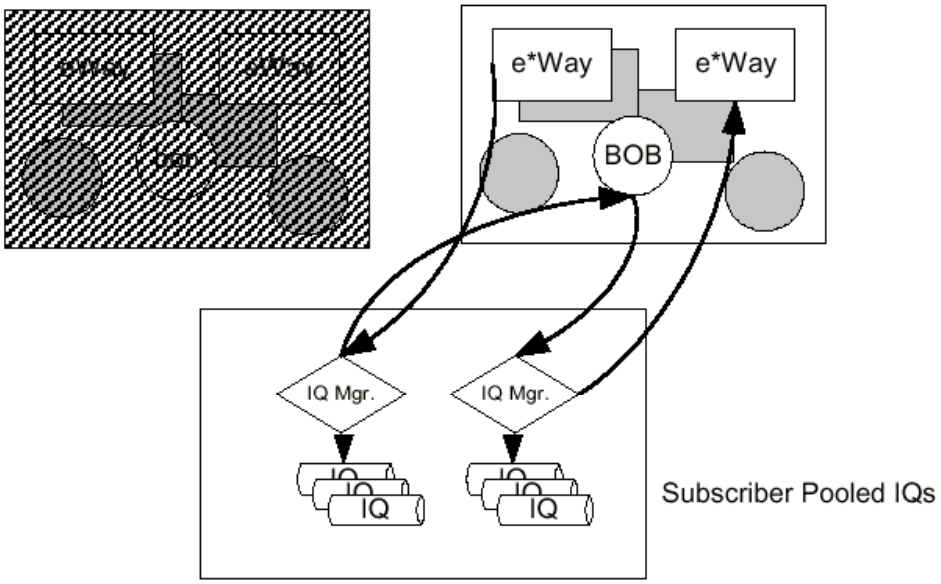


Figure A-2: Failover through Subscriber Pooling