MQSeries® Financial Services Edition

Technical Architecture Book

Version 1.2.2

NOTE:

Before using this information and the product it supports, read the information in “Appendix, *Notices*” on page 56.

Third Edition (November 2001)

This edition applies to version 1, release 2, modification 2 of MQSeries Financial Services Edition (product number 7J0423) and to all subsequent releases and modifications until otherwise indicated in new editions.

Printed in USA.

Contents

Appendix

Figures

Tables

About this book

The purpose of this document is to provide a detailed description of the technical architecture of MQSeries Financial Services Edition (MQSFSE) and does not include information associated with requirements of specific applications, which may or may not be integrated with MQSFSE.

Who should read this book

Systems Architects, Developers, and any other I/T professionals interested in learning more about the MQSFSE technical architecture.

Terminology used in this book

All new terms introduced in this book are defined in the *Glossary* on page 61.

This book uses the following shortened names:

- MQSeries®: a general term for IBM MQSeries messaging products.

Prerequisite and related information

It is assumed that the reader is familiar with with the prerequisite MQSeries products, which are MQSeries, MQSeries Integrator, MQSeries WorkFlow.

This document is not intended to be a primer on the MQSeries family; for more information on the features of MQSeries, please see the MQSeries Family Documentation.

How to get additional information

Visit the following home page at:

http://www.ibm.com/software/mqseries/support/

By following this link you can find:

- The latest information about MQSeries family of products.
- Download Support Packs.
- Access FAQs.
- Access MQSeries family publications library.

How to send your comments

Your feedback is important in helping to provide the most accurate and high-quality information. If you have any comments or suggestions about this book or any other *MQSeries Financial Services Edition* documentation:

- Send your comments by e-mail to **einet@us.ibm.com**. Be sure to include the name of the book and the specific location of the text you are commenting on (for example, a page number).

- Send a letter with your comments to:
  IBM Corporation
  Department: 5EFA/Building: 202
  8501 IBM Drive
  Charlotte, NC 28262
  U.S.A.

Chapter 1
Introduction

MQSeries Financial Services Edition version 1.2.2 (MQSFSE) provides the ability to integrate front-end systems to back-end systems in a non-invasive fashion in a customer's enterprise environment.

Figure 1 illustrates the intended use of MQSFSE:

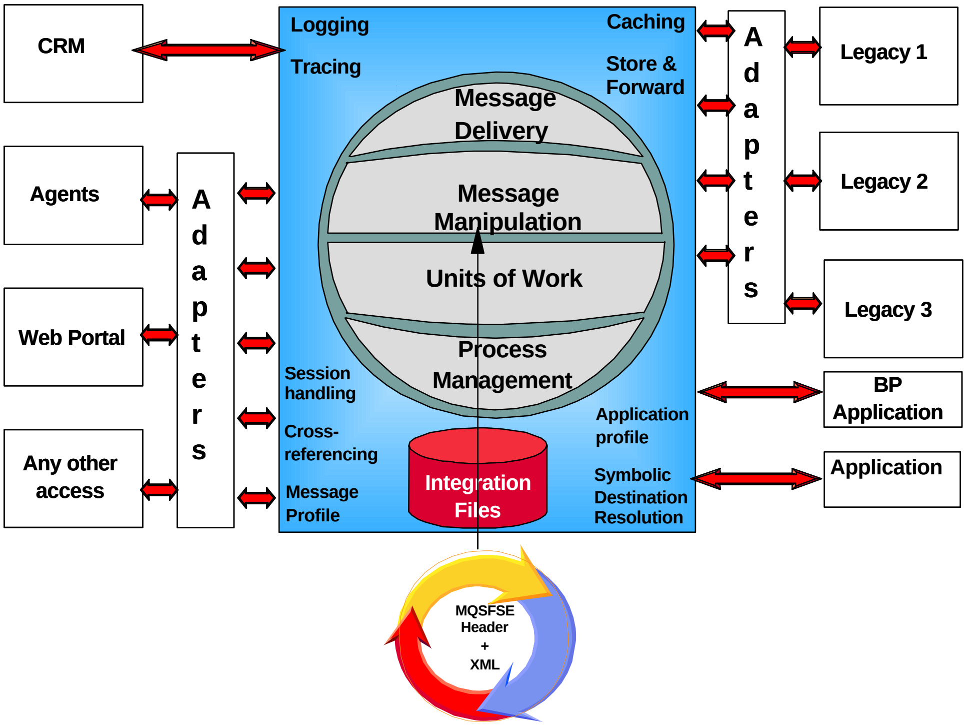

Figure 1: Intended MQSFSE use.

MQSFSE is designed to integrate disparate applications using a hub and spoke architecture to implement a message bus. MQSFSE architecture requires that application-specific information be translated into the message dialect used with an MQSFSE implementation. Within a business event, MQSFSE provides the data manipulation, application cross-reference, message configuration, symbolic destination management, and logging required to assure delivery of both the request XML message to the appropriate target application, as well as the response message to the originating application. Adapters are used at the boundaries to translate application specific information into and out of the XML message dialects understood by MQSFSE.

In many cases, the nature of the applications themselves dictates a great deal about what needs to be done to enable the application to be integrated. Once these specific requirements have been identified for each application, there are a series of additional requirements imposed by MQSFSE, which allow applications to inter-operate within an MQSFSE enabled environment

One of the decisions that is required in order to implement MQSFSE, is to determine what kind of XML vocabulary will be used within the hub. For more information on product implementation see the MQSFSE Development Guide. From an MQSFSE perspective, there are two types of XML vocabularies:

- Internal
- External

Internal vocabularies are generally data model based and are used to develop custom message sets that can be optimized to the specific requirements of the applications being integrated.

External vocabularies are generally the result of the work of a standards group such as the Open Application Group (OAG) or Interactive Financial eXchange (IFX). These messages sets are intended to provide an XML implementation that governs how content must be communicated. While these standard approaches usually support some form of extension, there is much less freedom to optimize the messages to existing application requirements.

Internal vocabulary XML messages can be defined with the Interface Design Model that is supplied as part of MQSFSE. In this version, the IDM supports versions that consist of content derived from OAG related content for generic CRM and supply side integration, Insurance Application Architecture (IAA) for insurance integration, and IFX for retail banking integration. MQSFSE also supports the external version of the IFX and OAG XML vocabularies through the use of message gateways. The message gateways wrap and unwrap the message sets with a header that allows the message to drive MQSFSE functionality, without the originating applications being hub aware.

Figure 2 shows a logical view of MQSFSE:

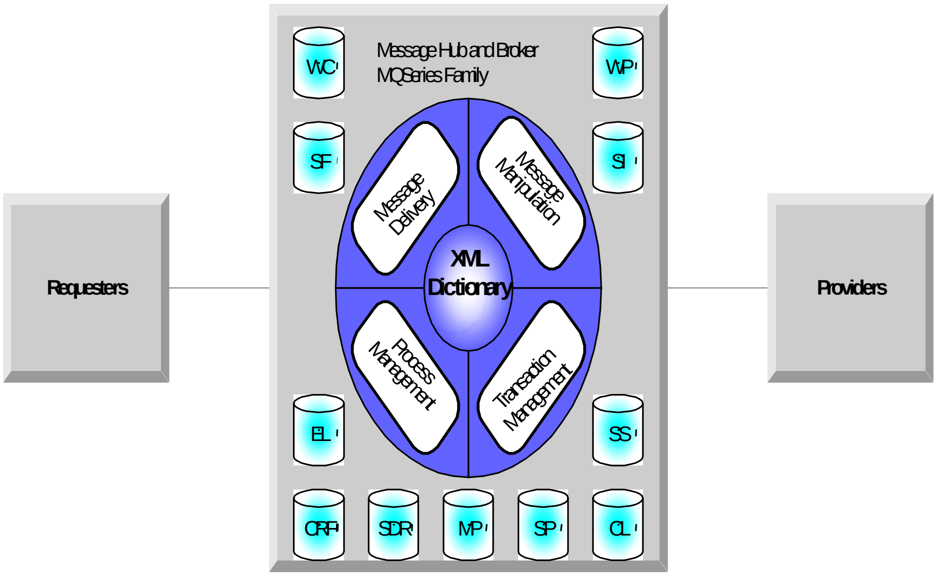

Figure 2: A logical view of MQSFSE.

MQSFSE version 1.2.2 is composed of the following services:

- Assured Message delivery.
- Message content manipulation.
- Transaction management of the activities through which a message is routed.
- Process management of the tasks associated with the activities through which a message is routed.
- Resolution of the destination to which a message is to be routed via Symbolic Destination Resolution (SDR) function.
- Cross referencing of the keys used by the application to access the data contained in the messages via the Cross Reference Function (CRF).
- Logging of message activities and errors via the Logging and Cache (CL) and the Error Message log (EL) services.
- Support for the message vocabulary used as the semantic for communication between the applications integrated with MQSFSE via MQSeries Integrator (MQSI).

- Assured delivery of a response message for each request message.
- Dynamic management of system availability via the System Profile (SP) services.
- Dynamic process management via configurable container definitions using the Workflow Container Profile (WP) services.
- Session and process management via the Session Status (SS) service, including state management of both the business event and cross business event messaging.
- Dynamic availability, rerouting, including Store and Forward (SF) via the Session Interaction (SI) and System Profile (SP) services.
- Dynamic tracing and debugging via the Error Log (EL) services.
- Complex long running transaction handling with assured response via MQSeries Workflow (MQSWF) and the Message Profile service.
- Lightweight message handling.
- Optimization of data content through the use of message caching.
- MQSFSE supports the ability to dynamically define container structures for passing data between MQSI and MQSWF. This architecture provides a number of capabilities including allowing the content of a message to be updated, providing the basis for transitions, and as the basis for collaboration via the MQSeries Workflow APIs.
- MQSFSE supports named versions of messages so that they can be stored and reused within a transactional message process. By combining this feature with the dynamic container feature, MQSFSE now provides support for multiple message types within a message flow.
- MQSFSE supports hub based Logon and Logoff messages that allow MQSFSE to generate a session token that can be used for simple session/state handling for validating message affinities.
- MQSFSE supports a message store and forward option that allows messages to be re-processed when unavailable applications become available.
- MQSFSE provides support for both a lightweight MQSI based message processing supervisor and a transactional message processing supervisor.

- MQSFSE provides a profile that allows applications to signal their availability, as well as the definition of mirror and alternate applications to use as the target for message requests.
- MQSFSE provides message maintenance of internal tables.
- MQSFSE uses the message types of Request and Reply.
- MQSFSE tracks and records the status of messages as they are processed through MQSFSE.
- MQSFSE internally traces activities including workflow template completion to provide comprehensive tracing and problem determination support.

Chapter 2
MQSFSE architecture

New feature highlights for version 1.2.2

The following list describes the new features available in MQSFSE version 1.2.2. These features are explained within the context of the MQSFSE architecture in the remainder of this publication.

- Support for OAG customer relationship management and supply side messages and scenarios.
- Industry neutral Interface Design Model (IDM) for generating internal message sets optimized to specific applications.
- OAG enhanced IDM.
- OAG MQAO based message gateway provides both an example of building MQSFSE support for native external vocabularies, and implements support for the OAG CRM and Supply Side use cases shipped with the product.
- Improved TestDrive functionality for MQSFSE product evaluation and model office work.
- Enhanced support for publish and subscribe. MQSFSE provides explicit support for subscription to topics, and customizable implementation of publication within MQSFSE.
- Enhanced authentication services for hub related messages.
- Alternative Symbolic Destination Resolution service based on LDAP.
- Enhanced and NLS enabled error handling within MQSFSE.
- Enhanced streamlined Cross Reference function.
- Pub/Sub support for errors within the hub.

End-to-end view

MQSFSE provides an enterprise-wide scalable framework that allows multiple front-end applications (e.g. web and call centers) to be connected to multiple back-end applications (e.g. policy administration and claims systems) in an effective and efficient manner, as a starting point for services engagements.

Figure 3 illustrates an MQSFSE implementation:

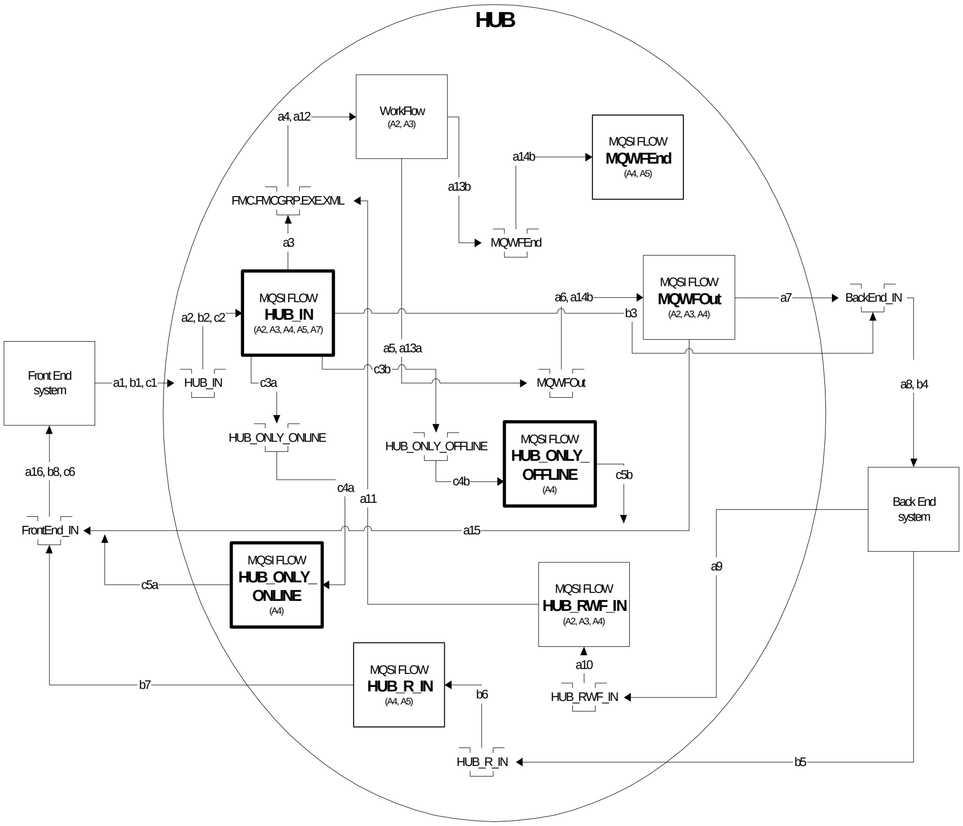

Figure 3: MQSFSE implementation.

MQSFSE in it's simplest form requires the following tables in the MQSFSE Database:

Service name/Alias	**Database table name (s)**
Cross Reference/ FSE_CRF	CRF_TABLE
Errors/FSE-ERRL	END_TRACE_TABLE ERROR_TABLE EXCEPTION_TABLE MESSAGE_TABLE ORIGINAL_MESSAGE_TABLE START_TRACE_TABLE
Message logs/ FSE_MSGL	MESSAGE_LOG_TABLE
Message Profile/ FSE_MSGP	MESSAGE_PROFILE_TABLE SYSTEM_INTERACTION_TABLE WORKFLOW_PARAMETERS_TABLE
Symbolic Destination Resolution/FSE_SDR	SDR_TABLE
Sessions and processes/FSE_SESS	SESSION_TABLE SESSION_PROCESSES_AND_SYSTEM_USAGE_TABLE SESSION_AUTHENTICATION_TABLE PROCESS_STATE_TABLE
Tracing/FSE_TRAC	START_TRACE_TABLE END_TRACE_TABLE
Store and Forward/ FSE_STOF	STORED_MESSAGE_TABLE INTERACTION_DEPENDENCY_TABLE
System profiles/ FSE_SYSP	SYSTEM_STATUS_TABLE SYSTEM_STORE_FLAG_TABLE SYSTEM_BACKUP_TABLE
Workflow control/ FSE_WFCO	WORKFLOW_PARAMETERS_TABLE WORKFLOW_CORREL_TABLE

Table 1: MQSFSE database tables.

In addition the following messages queues are also needed:

Queue name	Description
HUB_IN	Input for message requests.
HUB_IN_FAILURE	Failure path for queue.
MQWF_OUT	MQWF to MQSI message link.
MQWF_OUT_FAILURE	Failure path for queue.
HUB_RWF_IN	Response queue for WF driven messages requests.
HUB_RWF_IN_FAILURE	Failure path for queue.
HUB_R_IN	Response queue for Broker driven message requests.
HUB_R_IN_FAILURE	Failure path for queue.
MQWF_END	WF end of process signal graph.
MQWF_END_FAILURE	Failure path for queue.
HUB_ONLY_ONLINE	Input for messages for online hub functions (Logon, Logoff, CRF).
HUB_ONLY_ONLINE_FAILURE	Failure path for queue.
HUB_ONLY_OFFLINE	Input for messages for offline hub functions (File maintenance).
HUB_ONLY_OFFLINE_FAILURE	Failure path for queue.
FMC.FMCGRP.XML.EXE	MQSI to MQWF link.
WORKFLOW_DEFAULT_ACTIVITY_ERRORS	Holds error information.
WORKFLOW_DEFAULT_ACTIVITY_FAILURE	Failure path for queue.
WORKFLOW_DEFAULT_ACTIVITY_IN	Place holder flow.

Table 2: MQSFSE queues.

MQSFSE provides a wide range of functions and features capable of handling a broad range of integration requirements. The intention is that these functions and features are optimized as a part of the engagement to match customer specific requirements. By providing support for the addition, subtraction, and changes to the systems comprising the enterprise environment, MQSFSE diminishes the need for major programming and or operational changes.

The primary functions of MQSFSE provide:

- A mechanism for rapidly automating new business processes through application integration.
- Isolation between front-end and back-end systems that eliminates the need for the various application changes that were historically required when making enterprise applications aware of each other.
- "Impedance matching"-the ability to facilitate integration by translating between different system contexts (protocols, designs, languages, formats).
- A mechanism for leveraging existing applications in new ways, e.g. 24 ×7 availability, and selling/servicing online.
- Synchronization of multiple applications though process management of a common shared business event.

MQSFSE is designed to be transparent to the end user by providing the underlying architecture and infrastructure required for integrating applications. This transparency is accomplished in a fashion that leverages the existing functional value of the integrated systems within the enterprise. Most interactions with MQSFSE are limited to customization efforts by the individuals involved with increasing the scope of the enterprise using automation via process management definitions that describe the business event. As a result, an implementation of MQSFSE is scalable to meet the business requirements of the organization with minimal impact on existing implementations.

MQSFSE can support implementations that are as simple as system-to-system integration, or as complicated as new application development using process automation. MQSFSE is designed to support simple scenarios, and as business needs evolve, add more complex scenarios. By integrating standards-oriented technologies, projects are encouraged to start with simple high yield implementations that can be grown over time as business requirements dictate the need to provide additional support for a wide range of business processes. The architecture is designed to support addition of features that enable business-to-business collaboration as required by the enterprise.

MQSFSE extends the use of basic messaging for application-to-application inter-operation by adding both business process management and process collaboration that allows implementation of increasingly sophisticated types of business automation. MQSFSE provides this support through the use of a hub and spoke architecture. The message level integration is achieved by using a special XML header that operates as a wrapper, resulting in a unifying model that provides the common semantic for the content of messages used to integrate systems. The MQSFSE header provides access to the services designed to transform and deliver the messages for use by the end-point applications.

MQSFSE, through the use of MQSeries Workflow (MQSWF), can drive new and existing processes. The features of MQSWF makes it possible to first capture and implement existing business process definitions, and then provide new levels of automation, collaboration, and integration for existing application environments. MQSWF allows business processes to not only be automated and enhanced through the introduction of both manual and program activities. Once the business process flow has been separated from the business service provider, these new activities can be introduced without requiring changes to the service providers themselves.

MQSFSE can support simple point-to-point integration requirements by allowing the type of process automation to be selected on a message-by-message basis. For complex or long-lived processes, MQSWF base process management can be used. For simple point-to-point requirements, business process management can be bypassed in favor of brokered message Delivery.

Business processes

MQSFSE supports many different types of integration efforts by integrating a series of flexible system components. The goal is to provide a reusable framework for implementing solutions for a wide range of integration requirements. The intention is to use MQSFSE as a starting point for custom enhancements, to tailor the implementation to match client requirements. Use cases are used to document an understanding of the business problems to be solved by providing a basis for representing the system interactions, data flows and manipulations required to satisfy both new and existing business problems. By mapping the use cases against an appropriate business model, a common representation of the business problem can be established that removes ambiguity and allows related activities and information to be integrated across disparate systems.

As an example, the model representations of a business problem can be used for generating the required XML messages and the tasks and workflow scripts used to drive the implementation of the use cases in MQSFSE. The non-functional requirements for scalability, reliability, throughput, and growth rates can then be addressed through the features of the open standards based system components making up the MQSFSE framework. The end result is a set of implemented business scenarios that provide directly executable solutions, act as building blocks for future solution implementations, greatly simplify integration efforts, and reduce the inherent risks of integration efforts. For more information see the MQSFSE publication titled Planning Guide.

Implementation

The MQSFSE framework is built on the MQSeries family of products, with extensions designed for several industry specific implementations. The general design uses XML messages to provide a common semantic for representing domain content in the form of a vocabulary that allows disparate applications to communicate. MQSeries provides the base transport layer for this communication between end-point applications, using a hub and spoke architecture. MQSFSE is comprised of a set of MQSeries Integrator (MQSI) based message flows that operate as generic components used to provide functions that facilitate integration between the end-point applications. For simple integration requirements, the MQSI components themselves provide direct support for integrating the end-point applications. For complex scenarios MQSeries Workflow (MQSWF) scripts are used to exercise the MQSI based components as needed to tie the enterprise together by providing the integration between the end-point applications.

The next sections describe the various components required to implement MQSFSE.

Base transport layer

The MQSFSE framework is built on messaging. The assumption is that if a message is sent from a source application, the message will be processed as soon as MQSFSE assures delivery to the intended destination.

MQSFSE uses MQSeries queues for all communication with front-end and back-end applications as illustrated in Figure 4:

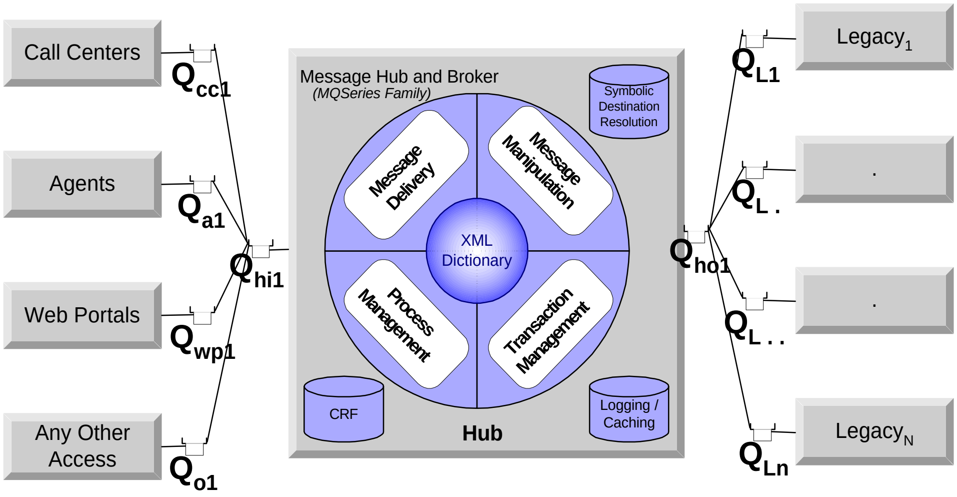

Figure 4: MQSFSE use of MQSeries.

Messaging

The MQSFSE framework uses MQ queue-based messaging as the basis for integrating the environment, and as a by product fully supports integration via the Java Messaging Service specification. The MQSFSE framework supports all of the message types supported by the MQSeries family by wrapping these messages in a message header. The industry specific extensions implemented on the MQSeries family of products making up the MQSFSE framework rely on XML as the architecture defining the base vocabulary for messaging. An MQSFSE header is used to wrap the data and command parts of the XML message. The XML architecture provides the data representation based on the architecture of the XML vocabulary chosen for use within MQSFSE.

Message headers

MQSeries makes extensive use of message headers to drive processing through the MQSeries family of products. MQSFSE uses the standard MQSeries headers and a specialized MQSFSE header to drive MQSFSE processing. Essentially, the MQSeries headers wrap the MQSFSE header, which in turn wraps an XML message.

The basic use of headers by MQSFSE is illustrated in Figure 5:

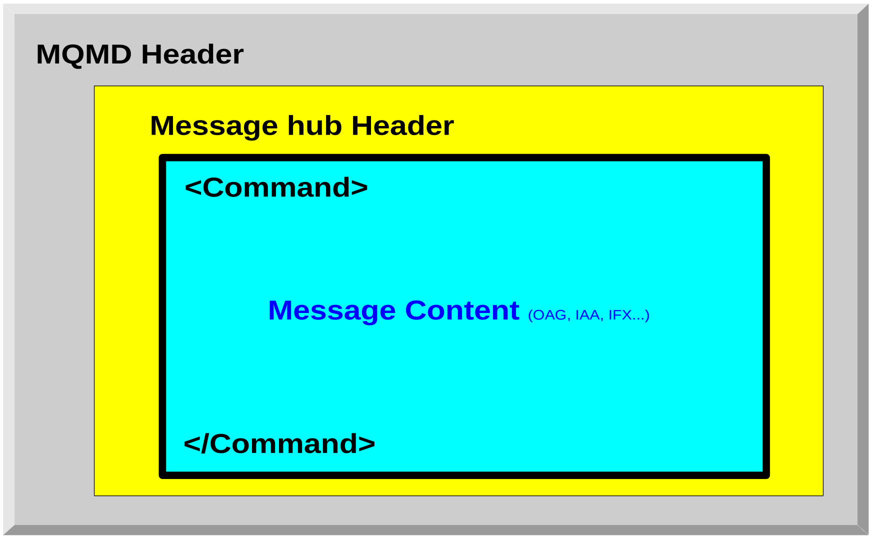

Figure 5: Basic use of headers in MQSFSE.

Message structure

MQSeries	XML <container>	
MQMD header	MQSFSE header	XML commands

Figure 6: Message structure illustration.

MQMD header

An MQSeries message header, called the MQSeries Message Descriptor (MQMD), is placed in front of all MQSeries messages. This header is created at the message creation time by MQSeries. See the MQSFSE publication titled ***Development Guide*** for more information.

MQSFSE header

An MQSFSE header, called message header, is the first piece of information in the body of the message. The header contains information used by MQSFSE to perform message routing, key manipulation, logging, and provide support for authentication functions. See the ***MQSeries Application Programming Reference*** for more information.

Message content

The Message Data contains the business data in the XML message. In the case of IAA-XML messages, there are essentially three sections to the Message Data Header command, aggregate, and property. Since IAA-XML messages are an XML representation of object form of the IAA model:

properties correspond to the attributes of the object model classes.

aggregates correspond to the classes of the object model.

The commands can be thought of as methods on objects of the model. For IFX and OAG on the other hand a data model is not required. Instead, the message data is defined by the associated DTD.

See the ***MB-XML Architecture*** book and ***http://www.ifxforum.com***, ***http://www.openapplication.org*** web-sites for more information.

While MQSFSE does provide forms of the IDM that include content derived from the OAG and IFX messages that have been modeled, the generated messages use the architecture described above for IAA. The intent of the IFX and OAG IDM is to allow internal form XML messages to be developed that can be optimized to the applications requirements of a particular enterprise using the appropriate message content. The IFX and OAG IDM's do not create IFX or OAG messages, instead the appropriate IFX and OAG DTD must be used as a child of the MQSFSE header DTD for this purpose.

Message classes

MQSFSE segregates messages into distinct processing classes based on the type of message being processed. The MQSFSE header contains a bodyCategory tag that is used to indicate the message type. By loading the appropriate value into this tags, MQSFSE will route the messages through the appropriate functionality.

The two available classes are hub-oriented messages and standard content driven messages. Hub only messages are intended to manage the internal MQSFSE function components themselves; while standard content driven messages use hub functions as a means of accessing the applications being integrated by MQSFSE. The XML design of the hub-oriented messages is dictated by MQSFSE, and essentially consists of the MQSFSE header and an XML representation of the table structure. The non-hub-oriented messages are comprised of the MQSFSE header that wraps a structure defined by a specific message architecture such as OAG, IAA, IFX, etc.

Adapters

In order for external applications to communicate with MQSFSE, the MQSFSE framework relies on translation programs also referred to as adapters. Adapters themselves are not part of MQSFSE, but are responsible for translating the application specific syntax into the specific syntax of the XML vocabulary that will be processed by MQSFSE. Generally, adapters must be written by the owners of the applications to be adapted, as it requires experience with the application to map the Application Programing Interface (API) to the appropriate XML message definitions.

Adapter services

The services that adapters must support in order to integrate with and leverage the features and functions of MQSFSE are as follows:

1. Generating the MQSFSE header information, which includes:
    - Symbolic routing information for defining the source of the message and the intended target destination.
    - Cross reference control information for driving the translation process between system-specific keys.
    - Message processing information used to drive a specific message though a predefined set of activities.
    - Correlation of request and response messages through the MQSFSE message correlation infrastructure.
2. Loading the MQSFSE message data by transcoding application specific data into the appropriate XML representation that includes:
    - Creating messages that adhere to the appropriate DTD.
    - Managing data aggregation across one or more messages into and out of application representations so that application data integrity can be maintained.
3. Each interaction between an adapter and the adapted application must be an atomic unit of work. An application cannot rely on successive messages to maintain consistency; instead, consistent state must be maintained through each message interaction.
4. The adapter must be capable of informing MQSFSE of events that change the required cross-reference key entries for entities managed by MQSFSE. All events that cause an add, delete, or update event against managed keys must be communicated to MQSFSE. For example, an application adds a

customer entry. It must send a message to MQSFSE to create a cross-reference entry in MQSFSE's CRF table that can later be used as a part of a translation process when the party is referenced in messages to other applications in the MQSFSE environment.

5. Adapters must define their response queues using the MQMD infrastructure.
6. Adapters must provide support for restart and recovery of the application that they are servicing. Although MQSFSE itself has full recovery capability, if a message has been handed off to an adapter by MQSFSE for processing. Then MQSFSE will treat the activity as a system failure if the adapter does not restart the message processing on its own.
7. Adapters are responsible for handling application security requirements. The adapters use the information in the user identification tags in the MQSFSE and MQMD headers to satisfy application authentication and authorization requirements. In some cases, this use of the user identification tag requires custom application changes, but generally it can be handled via the common logon framework.
8. Adapters are responsible for handling logging for the purposes of creating audit trails for activity in and out of the application that the adapter is driving.

For more information see the MQSFSE publication titled ***Application Integration Guide***.

Approach

There are several ways that MQSFSE compatible adapters can be built. The exact approach is not as important to MQSFSE as is the requirement that the adapters provide support for the basic services as described in the previous section. The following describes some of the basic approaches to adapter implementation.

MQSI

MQSI can be used to create an adapter in several ways. First, since MQSI is XML-aware, it can be used to translate XML dialects if an application to be adapted already speaks XML. Second, MQSI contains a framework that allows custom processing nodes and parsers to be integrated into the product. This framework allows the standard task/workflow scripting tools to be used to generate process flows designed to act as an adapter.

MQSeries Adapter Offering /MQSeries Adapter Builder

The MQSeries family of products include MQ Adapter Offerings (MQAO)/MQ Adapter Builders (MQAB). These contain both a buildtime and runtime environment that is aimed at building adapters that run as native operating system processes on any platform supported by MQSeries. MQAO/MQAB are built around the Visual Age Enterprise Access Builder environment that facilitates generating code from DTDs and/or copybook/header files in a fashion designed to implement the data translation and mapping support required in an adapter. MQAO/MQAB supports creating adapters in Java or C. The code is then added to runtime services for:

- Accessing MQSeries queues.
- Supporting application authorization.
- XML parsing via the MQAK and XML4J products.
- Setting and resolving symbolic destinations via the symbolic destination resolution function.
- Logging.

These adapters support both push and pull scenarios, including request/reply, publish/subscribe, and fire/forget processing modes.

The push scenario is non-invasive in that it does not require the application to be able to access MQ queues. Instead the framework pushes messages into native application APIs.

The diagrams in Figure 7 and Figure 8 illustrate the MQAO capabilities:

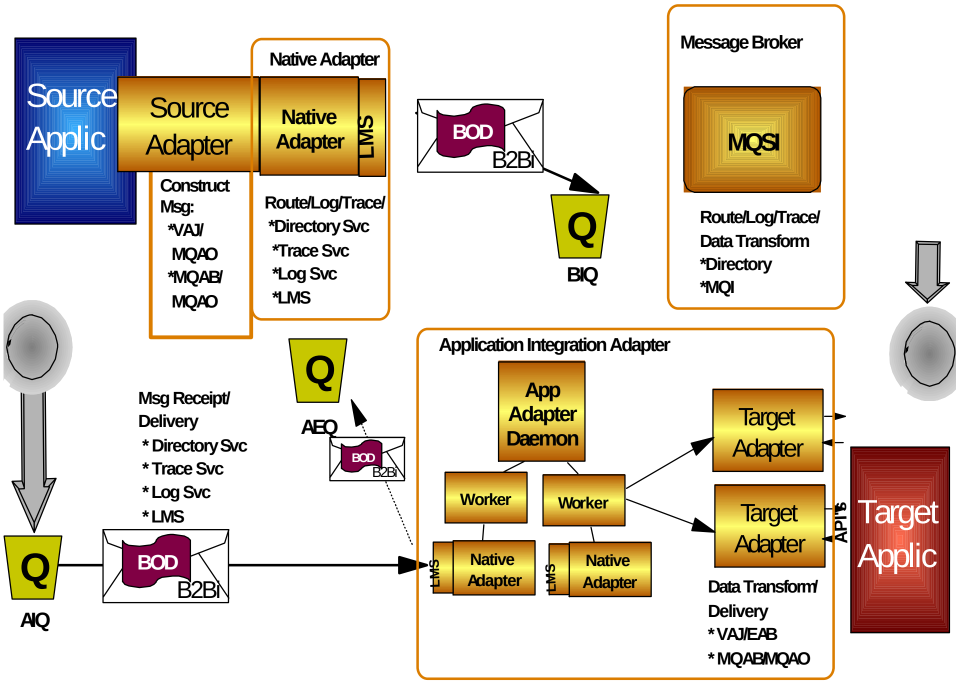

Figure 7: MQAO adapters, push scenario.

The pull scenario illustrates an application that is capable of accessing MQ queues natively.

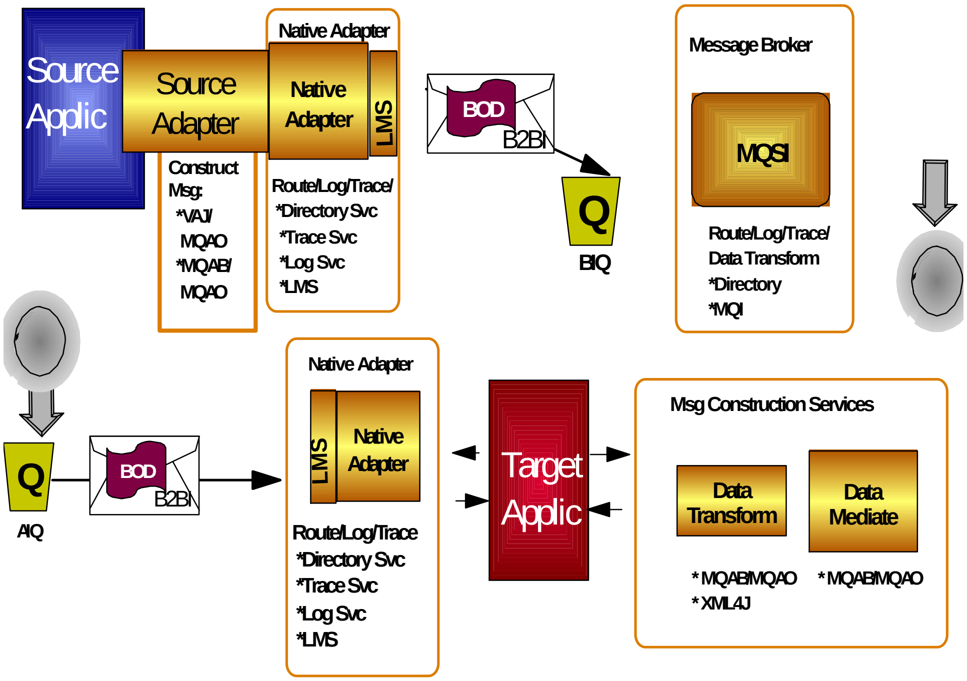

Figure 8: MQAO adapters, pull scenario.

Gateways

There is a special class of adapters that is referred to as a Gateway in the MQSFSE architecture. A Gateway is a simplified adapter that does not have to deal with application to XML message vocabulary transformations. Instead, the gateway is intended to wrap and unwrap an existing message so that it can be processed by MQSFSE. MQAO can be used for this purpose. In this case, the gateway is installing and removing the MQSFSE header to allow applications that already speak using a standard message set (e.g. OAG) to communicate via the hub, without requiring changes to the applications themselves. If an IDM message set is used (IAA, IFX/OAG modeled in the IDM) gateways are not required, because the IDM creates a custom DTD that already contains the MQSFSE header that the gateway is intended to provide.

See the ***Application Integration Guide*** for more information.

An example of the gateway implementation provided for the OAG messages is illustrated in Figure 9:

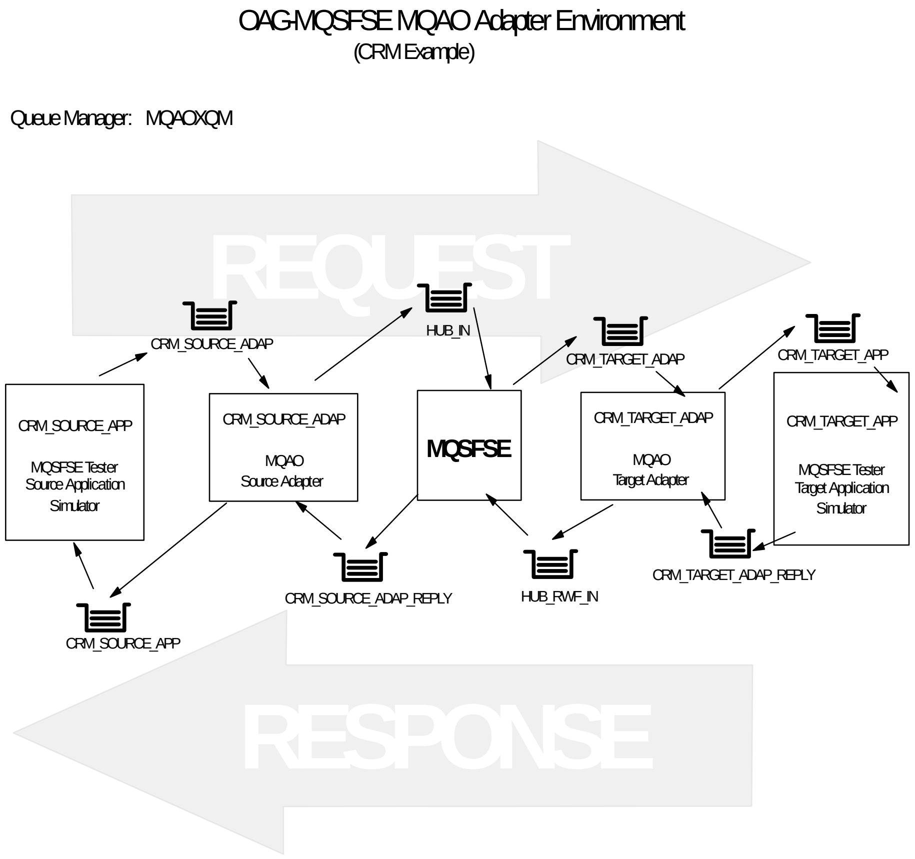

Figure 9: MQAO Adapter Environment.

Cross reference functions

The Cross Reference Function (CRF) of MQSFSE is responsible for providing translation between the system key representations for specific entities redundantly hosted in multiple integrated systems. The CRF function is an optional function that is only required when systems expose their native keys via the messaging model. When the keys are exposed the CRF provides the facility to allow applications to have access to their own keys when commanded via XML to do a particular activity. The CRF uses the source application's key to determine the keys required by the destination applications. The implementation is done using a design pattern that requires all end-points to communicate the state of their keys as business events are processed via the hub and at the adapted end points.

The translation is accomplished using a Universally Unique ID (UUID) generated by the database that stores the key entries. Each integrated system registers its key representation for an entity with MQSFSE CRF function so that it can be attached to a UUID. The adapters are responsible for loading the appropriate information into the MQSFSE header so that the CRF will be engaged to translate a particular system key representation to and from the UUID when a key translation is performed. MQSFSE translates the source key representation into the UUID on the way into MQSFSE and then translates from the UUID to the target application on the way out of MQSFSE. All MQSFSE CRF manipulation is done against the UUID inside MQSFSE.

Figure 10 illustrates how the CRF works:

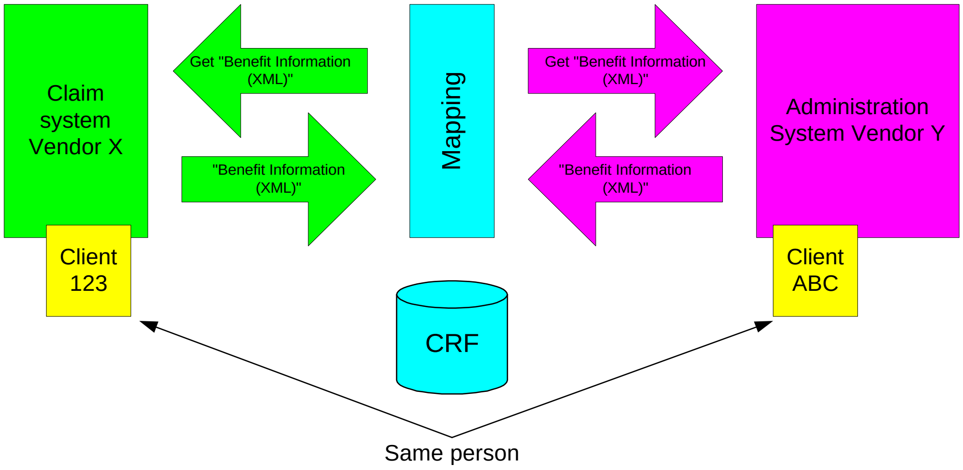
Claim system Vendor X
Client 123
Get "Benefit Information (XML)"
"Benefit Information (XML)"
Mapping
CRF
Get "Benefit Information (XML)"
"Benefit Information (XML)"
Administration System Vendor Y
Client ABC
Same person

Figure 10: MQSFSE CRF.

As messages are passed between applications, the CRF translates the representation based on the information coded in the MQSFSE message header, so that the source and target applications can maintain a relationship using their own representations of what is in actuality the same entity. The CRF is used to store the relationship between all keys used to represent entities in MQSFSE.

Table 3 below illustrates the type of information stored in the CRF.

UUID	Value	System Id	Key Type	Attribute
1	123	CIIS	Party	
1	zx3452	Legacy A	Party	aaa
1	gssdgdsf	Legacy A	Party	bbb
1	2345	CRM	Party	
2	2345	CRM	ContactPoint	
2	345	CIIS	ContactPoint	

Table 3: Sample information stored in CRF.

UUID: Enterprise-wide unique entity ID defining the specific entity that is described by the local system key entry and Message type.

Value: System specific key entry defined by the message type.

System Id: Symbolic identifier defining the origin of the key entry.

Key Type: System key type identifier (e.g. policy number, client number) as defined by the XML message vocabulary used in MQSFSE. This corresponds to the smallest atomic elements defined by the vocabulary.

Attribute: An adapter specific piece of information that can be used by an application to further delineate the key type. The value of this field is not used by the CRF.

The UUID defines a key relationship for each instance of an entity defined to MQSFSE as a means of providing linkage between the integrated applications. A CRF entry is required for any entity that requires key translation as a result of processing an XML message. An indicator and the associated data tags in the MQSFSE message header define how and if the CRF operations are used. Not all message flows require CRF functionality. For instance, a query usually only returns an application view of an entity, and so doesn't affect any other systems also containing a reference to that entity. However, all add, modify, and delete activities require an instance of the entity to be defined in the CRF, if the entity is shared throughout the enterprise. It is the responsibility of the system owning the data, also known as the system of truth, to publish messages related to add, modify, and delete activity for an instance of a particular entity stored in the CRF. So that other applications in the enterprise can use the information to maintain data integrity and synchronization within MQSFSE.

Figure 10 on page 22 illustrates two important concepts and benefits of the MQSFSE CRF.

The first benefit is that the design allows MQSFSE to normalize the granularity of the keys defined by the aggregates within the XML vocabulary to the level of detail actually implemented in the integrated applications. As an example, the table shows a 1 to N relationship illustrated for **UUID #1** As can be see from the table entries, CIIS is shown having one key entry for the **party keytype**; where as, the system called **Legacy A** actually has two keys for the **party keytype**. What this indicates is that **Legacy A** actually uses more granularity to store the data associated with the party aggregate as defined by the XML vocabulary. In addition, the Attribute column is provided as a way for an adapter to differentiate between the keys. This differentiation would allow **Legacy A** to further define for itself what and how the keys are used. MQSFSE stores but does not use the data stored in the **Attribute** column.

The second benefit is that the design allows MQSFSE to support the type of normalization when an application stores data associated with multiple aggregates, as defined by the XML vocabulary, using fewer keys. This type of normalization is illustrated by the **CRM** entries in the table. In this case the key for the **party** and **contact point** keytypes are the same, which indicates that this system stores data at a different level of granularity than what is defined by the XML vocabulary. The important thing to notice is that the **CRM** application is still creating key entries at the level of granularity specified by the XML vocabulary. This level is important because each of these **keytypes**, or **aggregates** are considered an atomic unit of data, and each integrated application must be prepared to handle data at the level of detail of this aggregate even if it does not internally store data with the same level of detail. The reason for this is that since the XML vocabulary defines this as an aggregate, all applications need to be prepared to communicate at this level of granularity since it is supported by the message architecture.

Figure 11 provides an illustration of how the MQSFSE CRF can be used:

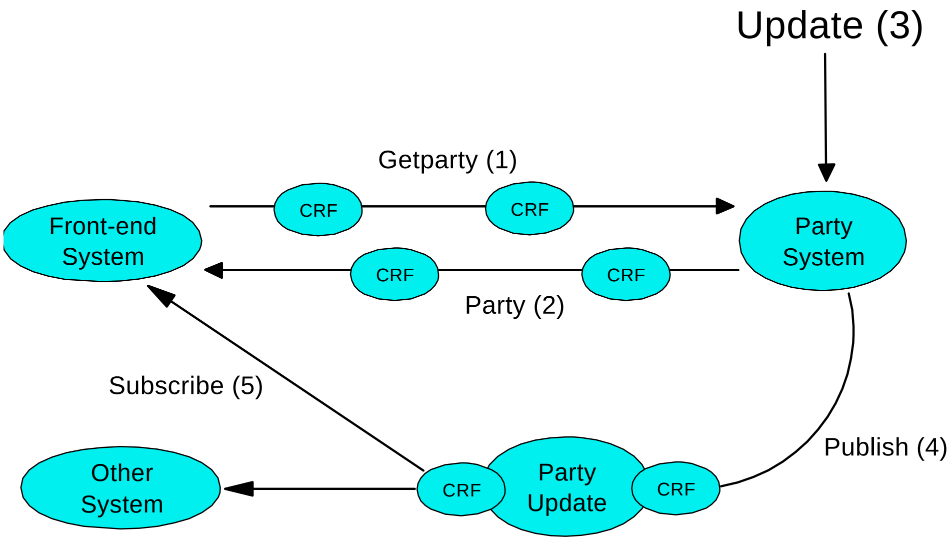

Figure 11: System of truth update/publish.

Front-end and back-end applications communicate through MQSFSE using the CRF to handle key translations. By design, the system of truth is responsible for controlling adds, updates, and deletes to the actual entities. When these activities take place, the system of truth publishes the event through its adapter. One possible way of doing this is by using the MQSFSE support for Publish/Subscribe, built on the functionality of the MQSeries Family of products; another way is via MQSFSE process management.

When Publish/Subscribe is used, all systems interested in this information must subscribe to the events via their adapters, and be prepared to take the appropriate action against a publish activity for an entity in order to synchronize their local copy. It is the responsibility of the system of truth to maintain the data integrity of the information it manages, as MQSFSE does not monitor changes or versions of entities for the purposes of maintaining data integrity. In the above example, if the update (3) was actually a create or delete, then the party system, because it is the system of truth, would be responsible for informing CRF. One thing to keep in mind is that existing Publish/Subscribe functionality only assures access to the current version of a topic. As a result, it is usually necessary to publish verbose messages containing sufficient content to provide the context required to re-synchronize a subscriber.

In addition, subscribing applications may need to use CRF messages to insure that the CRF remains synchronized with business events. The process management approach on the other hand, choreographs delivery directly to the interested applications. This approach allows MQSFSE to evaluate and handle the response the message creates in each adapted application and as a result manage CRF synchronization as a part of handling the message response for each of the applications integrated via the message delivery process. In some cases a hybrid approach is useful.

An example is when one or more applications play the role of "system of truth". In this case process choreography is used to handle delivery to the systems that own the data to insure proper synchronization of the environment. Once the "system of truth" have been satisfied, the message is then published to the remaining interested systems who do not have any sort of veto authority of the message process.

Figure 12 illustrates system not defined by the system of truth:

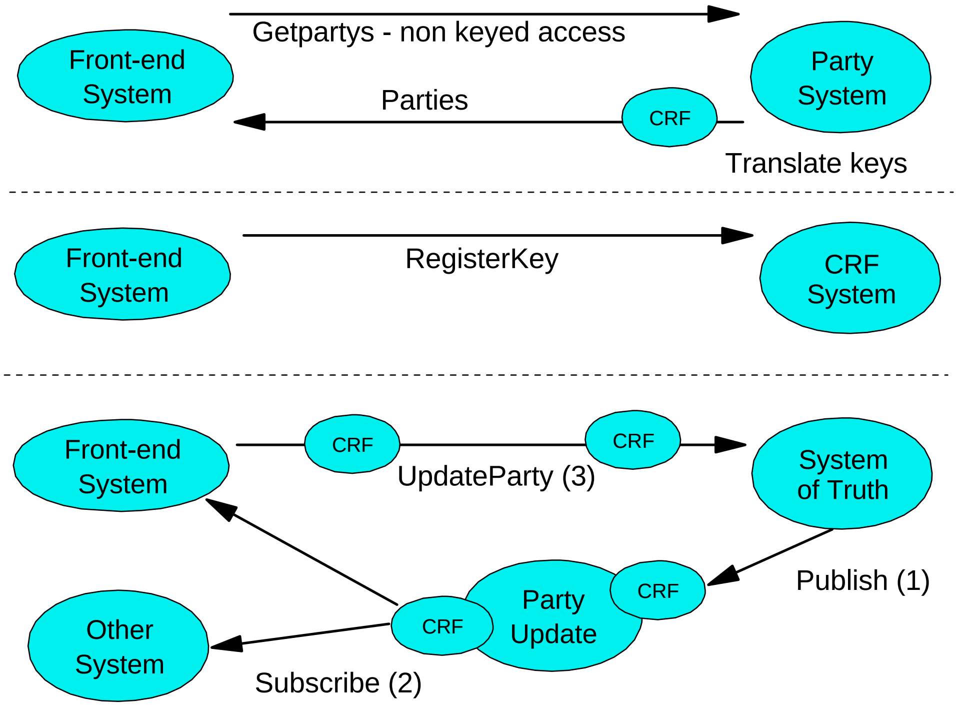

Figure 12: Inquire/update; not defined by the system of truth.

As an example, this scenario describes the activities associated with applications not defined as the system of truth.

It is possible for a front-end system to request information from a back-end system in a fashion that does not require a CRF operation, because it is an inquiry and does not affect entity relationship information. An example would be a query such as a GetPartysByAlphaSearch message. In this case, the request is sent and the party system of truth returns the appropriate information. The returned information is translated via the CRF based on the MQSFSE header. Once the front-end system has a copy of the information, it can keep it or discard it as appropriate. If the front-end system decides to use the information for an update, then it must first attach its key to the UUID originally returned from the party system of truth by registering with the CRF, before processing the update. When the front-end decides to process the update message, it must send the message to the system of truth so that the system of truth can process the update (as previously described), including a subsequent publishing of the event. MQSFSE takes care of the actual routing of the message based on the MQSFSE message header information. The adapter for the front-end system is required to set the MQSFSE header information to indicate that it is an appropriate message type sent to an appropriate symbolic destination using an appropriate CRF (e.g. UpdateParty(3) to the Party System).

For more information including other options on how to use CRF, see the MQSFSE publication titled ***Application Integration Guide*** for more information.

Intelligent routing

MQSFSE contains intelligent routing based values contained in the MQSFSE message header. The routing of messages is based on using the adapters and the capabilities of MQSI and MQSWF within MQSFSE. In all cases, routing is based on translating the symbolic destinations loaded into the MQSFSE header using entries stored in a symbolic destination resolution table or alternative LDAP. This architecture provides the means of determining the actual destination queue and queue manager names for the messages. There are many variations within the hub for how to set the actual value contained in the header within the MQSFSE framework. The value can be set by the adapters, by workflow, or based on entries coded in the message profile. Regardless of how the value is set, the goal is to determine the appropriate MQSeries queue for message delivery. Once the appropriate queue has been determined, an MQSI flow can put the XML message into the queue via a destination list so that the message can trigger the appropriate actions, by another component of MQSFSE, or an adapter.

Adapter responsibilities

As stated in the adapter section, it is the responsibility of the adapter to initially load the symbolic source and destination identifiers into the MQSFSE header. This information is used by MQSFSE for message routing. The MQSFSE routing uses MQSI to interrogate the symbolic names and determine the actual queue names via entries in the Symbolic Destination Resolution table.

In addition, adapters built using MQAO can provide direct support for routing, without requiring MQSI. The MQAO framework provides classes capable of looking up destination ID's via LDAP or configuration files, so those messages can be routed independent of MQSFSE. This routing is useful for high throughput scenarios, such as loading payments from a clearinghouse. Additionally, when Gateways are used in the implementation it is often necessary to use MQAO based routing because multiple applications provide support for specific message types. Unless duplicate adapters are deployed, specific messages must be routed through a common adapter, with the target destination determined after the fact. In this case, custom routing can be configured via MQAO, or the MQSFSE symbolic destination can be passed to MQAO as the basis for finally resolving the true destination of the message.

See the ***Application Integration Guide*** for more information.

MQSeries integrator

By its very nature, MQSeries Integrator (MQSI) is a message broker that is capable of doing content based routing. This content based routing means that custom flows can be generated based on the content of a message, or the queue that a message was read from. MQSFSE uses a symbolic destination resolution table where MQSFSE symbolic names and an MQSI flow to do routing.

MQSeries workflow

By putting an XML message into the XML MQSeries Workflow (MQSWF) queue, MQSFSE engages the appropriate process management script. As a result, it is possible to achieve a wide range of routing options based on using the features inherent in the MQSWF engine. MQSeries Workflow based routing in MQSFSE is based on the workflow message container content manipulation of the symbolic destination name.

Additional routing features

In addition to the ability to set destinations based on the features and functions of the MQSeries Family of products themselves, MQSFSE provides additional routing capabilities through the use of a System Profile, Message Profile, and System Interaction table. These tables give MQSFSE the ability to determine if the intended target of a message is available to process the message being sent to it. In a situation where the intended target is not available, it is possible to define alternate destinations for a message. In the case where no alternate destination is available, it is possible to indicate on a message-by-message basis whether the message should be discarded with an "application unavailable" reply returned to the originator or have the message stored until it is possible to reprocess the message. When the intended target is available, the message is reactivated for delivery. Applications can send messages to MQSFSE to indicate their desired status. MQSFSE will respond back to the system when it has determined there is no current activity intended for the application, and that all subsequent activities can be diverted to another target.

Message manipulation

When processing messages throughout the enterprise, it is sometimes necessary to manipulate the data contained in the message and adjust the format of the data to match platform requirements. There are many ways to handle these requirements within the MQSFSE framework, some of which are discussed next.

Enrichment and manipulation

Message data enrichment can be done in a number of ways. The approaches used in MQSFSE are either MQSI or MQSWF based. MQSI provides direct access to database information, including the ability to directly reformat messages via custom and existing nodes. MQSWF also provides the ability to affect the data stored within messages. MQSWF uses a container metaphor for passing data between activity points. At each activity point, the data can be remapped into a new format, and/or passed out to a process execution server for direct manipulation by a custom process or even a legacy system. In order to use MQSWF or MQSI for this purpose, the appropriate XML message is delivered to the appropriate MQSeries queue to trigger the MQSWF and MQSI process graphs.

Additional message manipulation features

MQSFSE provides the ability to define custom containers for activities within a workflow using the Message Profile Database. This table stores the specifications for generating a dynamic message container that is customized to support the requirements of a specific workflow process template. The purpose of these dynamic container definitions is to give MQSWF direct access to the specific message content required to implement the message flow. In addition, MQSFSE also provides the ability for an activity to request a new "on the fly dynamic container" be returned when processing results are returned to an activity. To further enhance this capability, MQSFSE allows each container to be given a name so that it can be recalled for later use within the MQSWF process graph. This combination of features provides a high degree of flexibility for generating messages in the formats required to integrate all aspects of the enterprise.

Code page and data formats

As stated above, both MQSI and MQSWF support enrichment and manipulation of messages. These functions can be used to satisfy data format requirements. In addition, MQSFSE uses MQSeries native code page translations to handle cross-platform data manipulation requirements. Adapters can also be used to handle data format and code page translation requirements. Adapter based translation is generally required when a transport layer other than MQSeries queues are used to deliver information to an adapted application, or to solve volume and/or throughput requirements.

Processing modes

MQSFSE supports three basic forms of asynchronous messaging.

- MQSeries Publish/Subscribe
- Request/Reply
- Fire/Forget

Essentially these are processed as an agreement between the end-point applications. The type of processing mode used is usually dependent on the process being implemented to solve a specific business requirement. However, in general, MQSeries Publish/Subscribe can be used to notify enterprise applications of changes when those applications have no veto authority over the change. When applications have veto authority, as is the case with a system of truth, then request/reply should be used to ensure that processing is completed in a timely and appropriate fashion. The last option is fire/forget, which should generally be used when a response is not required for a message request. The following describes the three types of processing modes available in MQSFSE.

MQSeries publish/subscribe

MQSFSE is compatible with the native MQSeries Publish/Subscribe feature, including MQSI publish and subscribe functionality. One approach to executing a publish/subscribe function is by a source adapter recognizing that a message should be published and puts the message into a queue managed by an MQSI flow. That flow in turn implements the publish process and publishes the information under the appropriate topic to the appropriate queues. The topic should be hierarchical in nature and could be a function of the message type. Applications looking for specific information must subscribe to the various topics published by MQSI.

The other approach is that MQSFSE provides direct support for defining subscriptions through a hub only message. Additionally, the MQSFSE architecture provides for directly defining the timing and message definition that should be published within the definition of a standard message flow. This approach relies on the administrator for defining the topic hierarchy and messages are published using body category as the lowest level topic in the previously defined hierarchy.

NOTE *In certain circumstances where a transaction is of high value (e.g. reinsurance), high impact (e.g. new claim), high risk (e.g. type of business), or high volume (e.g. batch processes), the integration design is likely to be made more effective by directing the message to an application-specific queue rather than to a subscribe queue. In some cases, it may even be appropriate to completely bypass MQSFSE, and in these cases care must be taken to ensure CRF integrity.*

Request/Reply

A request/Reply message set is a process where a source application sends a request message and asks for a response message. It is the responsibility of the target adapter to ensure that a response message is sent back to the source adapter, according to the rules coded into the XML message vocabulary defining the command being processed. The message vocabulary generally defines the format for a response to a specific request. The response message is returned to MQSFSE after the message has been processed, by putting the response message into the reply-to queue and associated queue manager as defined in the MQSFSE message header.

In some cases, it is necessary to send a message directly to CRF, if the message is of a request/reply process mode. In this case, CRF will respond with the appropriate XML response message.

It is up to the application and the associated adapter to determine if processing should wait for the response in Request/Reply processing mode, or handle the response asynchronously when it arrives.

Fire/Forget

A fire/Forget message is one that is sent without expecting a direct response. For example, this processing mode might be used when a system that is not the system of truth initiates a change to an entity. In this case, the application does not require a direct Request/Reply-processing mode, the application would already be a subscriber in order to receive the eventual notification of the processing results once it is published by the system of truth.

Messages processed in Fire/Forget mode should not send a reply to the application after processing the request.

Message processing supervisors

Message-processing supervisors are responsible for guiding a specific message to completion. MQSFSE provides access to two different message-processing supervisors. Lightweight message processing is based on MQSI, and assured reply transactional message processing is based on MQSWF. The MQSFSE Message Profile contains the definition of which supervisor to use on a message-by-message basis. Either supervisor can be used, but the default message processing supervisor is the assured response mode based on MQSWF. The alternative lightweight version is based on broker based delivery of a message by MQSI and does not engage MQSWF. This lightweight version does not support the same level of functionality as is supported by the transactional version, in favor of better performance.

Lightweight message-processing supervisor

In some cases, the lightweight message processing supervisor requirements are simple and straightforward, the case where an implementation using workflow only requires one activity. Essentially, the requirement is point- to- point in that the requester needs to send a message to a specific target. MQSFSE supports this by routing the messages directly from the source to the target while still providing access to the standard hub functions for CRF, SDR, and logging. The benefit is that this supervisor is functional yet lightweight; the disadvantage is that the implementation is stateless.

Unlike the transactional supervisor, which maintains states and can detect failures by an adapter in a fashion that allows the hub to automatically handle any problems, the lightweight supervisor optimistically assumes that the adapter will respond in a timely fashion to any messages it receives. Since this is not always the case, the source adapter has to be prepared to handle a failure, even though there is no way for it to determine the nature of the failure. In many cases, this outcome is acceptable, making the lightweight supervisor an excellent choice for many circumstances. In other cases, where the environment is more complex, the messages more important, and the exact status of the message processing activity required, then the transactional message supervisor is a better choice.

Transactional message-processing supervisor

The transactional message-processing supervisor provides access to all of the functionality in MQSFSE. This supervisor is designed around MQSWF which manages the states associated with processing a message within the enterprise. MQSWF drives the message through a series of activities that have been predefined as a means of handling the events associated with processing a specific message. The transactional message-processing supervisor is stateful; and therefore capable of managing the contingencies associated with failures during the message processing cycle. As a result, compensatory logic is centralized in the hub using a user oriented flow definition language, instead of hard coded in a requesting applications adapter. This makes the transactional message processing adapter far more robust and flexible.

Transaction support

By design, MQSFSE classifies applications into three categories.

1. Primary applications that are considered the *system of truth for a specific business activity*.
2. Applications that have some sort of veto authority over changes to an entity.
3. Applications that must just be updated so that they can be kept synchronized with the changes.

Depending on the class of an application, different types of transactional support are required to ensure that the environment stays synchronized. The following sections detail the types of transactional support available in the MQSFSE environment that are used for supporting specific business requirements of the scenario being implemented. The correct choice of support is the one that will ensure that processing always leaves the enterprise in a consistent state.

MQSeries sync point

MQSeries provides support for both single and two-phase commits. This support means that in the scenarios where there are both primary and secondary applications, it is possible in many cases, to coordinate the updates to ensure the integrity of the transaction across a wide range of platforms like CICS, IMS, and DB2 that support the XA interface. MQSeries exposes access to the sync point interface through the MQSeries API, which allows adapters to use this infrastructure when deemed necessary to meet the business requirement. The MQAO supports participation in the commit process. In addition, both the MQSI and MQSWF components of MQSFSE can participate in the commit process. However, care must be taken to determine the commit scope across all participants. In some cases, it may be necessary to wrap multiple applications under a single adapter in order to enable them to participate within the same unit of work.

Compensatory transaction processing

Applications play primary and secondary roles in the MQSFSE framework. In this case, MQSI and/or MQSWF processing frameworks can be used to determine how to handle failure conditions in order to drive updates into the applications. When a failure occurs during the execution of an MQSWF process template, both MQSI and MQSWF will roll processing back to predefined error nodes, and initiate predefined processing scripts, allowing compensatory transactions to be targeted at the failing applications. Since the adapters are responsible for maintaining a consistent state for the application, the compensatory transactions are essentially used to reverse entries that were successfully applied to ensure that the environment is left in a consistent state at the conclusion of the transaction.

The use and design of compensatory transactions is primarily determined by the design of the application being adapted. When one or more applications do not support the XA specification, care must be exercised to ensure the integrity of the business event. In some cases, it may be necessary to wrap multiple applications under a single adapter in order to enable them to participate within the same unit of work.

Long running

In many cases, the applications being integrated may not be able to respond in real time due to the nature of the business transaction or the activities to be performed. The MQSFSE framework supports this through the use of MQSWF, and its graphical design tool. The workflow templates that can be executed to implement a business process automation solution provide the functions required to handle application wait time. MQSWF lets the user define parameters framing how to interact with applications, including time-outs, and resulting compensatory actions.

Complex

In many cases, there may be a large number of individual tasks required to implement the automation of a business scenario, where tasks can be dependent on or independent of each other. MQSFSE supports complex transaction designs through the combination of the graphically defined MQSI and MQSWF process scripts. Using MQSI and MQSWF, tasks can be run serially, in parallel or they can be run conditionally depending on data content, and they can entail standards of service like long waits.

Collaboration

Through the use of MQSWF, MQSFSE supports the notion of collaboration. Not all processes in the environment are supported by systems. For systems that require manual intervention, MQSWF provides the infrastructure to support collaboration with end users, in a fashion that seamlessly integrates these collaborations with existing automated activities.

System management

In order to manage the complex distributive environments associated with many-to-many integration, MQSFSE uses facilities inherent in the MQSeries family products to provide system management support. The Tivoli products cover the following aspects:

Deployment

In distributed environments, applications often comprise multiple elements that must be configured for each platform and then distributed to remote nodes, servers and clients. The TME 10 Module for MQSeries automates the distribution and configuration of MQSeries across the enterprise. For example, before distributing and installing MQSeries, a check is performed to ensure that adequate resources exist on the target platforms.

Availability monitoring

Availability monitoring ensures the availability of applications and that those applications are operating optimally for end-users. The TME 10 Module for MQSeries monitors all MQSeries objects, the systems and networks on which they rely, while at the same time automating necessary responses to events such as application or server restarts.

Centralized administration

Centralized administration provides a means of configuring, controlling, and monitoring the entire application network from a central console or consoles in a secure fashion. Tivoli's role-based administration model supports the management of MQSeries networks from any managed node on the network. Different administrative roles with varying permissions can be assigned to administrators based on geography, specific business units, or departmental units.

Additional system management features

MQSFSE uses internal tables to track and record the status of a specific message flowing through the enterprise. Among other pieces of data, these tables contain the message id, session id, and process id associated with each message. This information allows correlation between the internal system management information contained in the Session Status tables and other MQSFSE tables, in order to provide a comprehensive snap shot of the processing environment at the time a particular message was processed. Since these tables are standard database tables, they can be accessed by a wide range of database tools, to obtain information about the MQSFSE processing environment.

Logging

In addition to the system management facilities provided by the third party products that may be used to support the MQSFSE environment, MQSFSE provides audit and error log information that can be used by system monitoring tools. The adapters and the CRF support logs so that an end-to-end view of the message processing status and results are available. The MQAO adapter kit uses a common framework for ensuring consistent audit and error logging. In addition, the CRF provides logging of translation activities in MQSFSE. MQSFSE provides access to all iterations of a particular message in the Message log; as well as detailed error information and trace results in the Error log.

Error handling

All message flows within MQSFSE are handled transaction-by-transaction so that the hub has the ability to roll back any changes whenever an error is detected. Messages that contain errors are routed to failure queues. The hub uses a series of tables to track the progress of a message, that can be used to determine the type and location of an error. When errors are reported, they are accessed by keys that provide National Language Support (NLS). Whenever an error message is created, it can be sent directly to a database error table, or it can be sent to the Pub/Sub Flow to be published to interested subscribers. The Message Profile contains a flag indication for the approach to use.

The NLS support is illustrated in Figure 13:

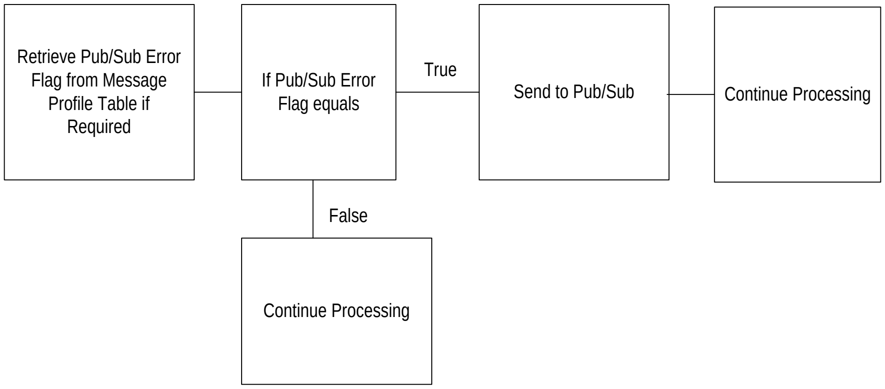
Retrieve Pub/Sub Error Flag from Message Profile Table if Required
If Pub/Sub Error Flag equals
True
Send to Pub/Sub
Continue Processing
False
Continue Processing

Figure 13: NLS support.

NLS support

MQSFSE uses an NLS architecture for generating messages that are sent to log files. The NLS Architecture describes a message that contains 9 digits as follows:

Position 1-2:	Language
Position 3:	Error Type
Position 4:	Architecture
Position 7:	Sub Module (i.e. Flow or Subflow)
Position 9:	Error Number

The first 2 positions of the number are language. Language will be stored in the System_Status_Table. In order to provide multiple language support from a single physical application attaching to the Hub, that application would need to define itself to the Hub using multiple entries in the System_Status_Table. Each entry would define a different language default.

The remaining 7 positions will be assigned to identify an occurrence of an error. It is possible that an error code could be utilized by multiple flows. In such cases, the sub-module portion of the error code will be set to a value that indicates a "generic" entry.

The result is a 9-digit error code. Each error code is associated with a single end user. Error messages may consist of text only messages or may be augmented with user supplied runtime values. An error code is associated to a message that assumes a fixed number of user-supplied values (0 or more).

The error message text is constructed by concatenating text strings stored in the proper sequence in a database table with user-supplied values. The values, if required, are inserted, in the order received, into the message text. If more values are provided than the message requires, the extra values are ignored.

Formatting rules are used to help present the error message in a readable fashion.

The format of an error message as displayed to the end user is as follows:

MQSHUB ######### location messageText

Where:

MQSHUB is a fixed string that indicates the HUB application.

######### is the 9 digit error code.

location is a String that indicates a failure log point. This value is optional. If provided, it must not contain any language specific content.

messageText is the language specific error test associated with the error code and augmented with user supplied values.

Processing is illustrated Figure 14:

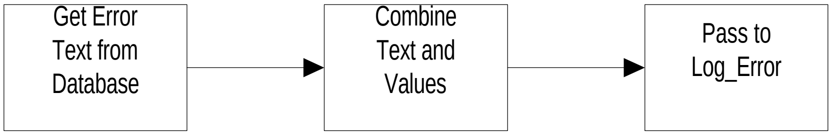

Figure 14: NLS process.

Directory management

IBM Secure Way Directory Server Version 3.2 is a LDAP server used to create a directory structure in a format that supports the Hub's SDR design. MQSFSE uses a MQSI flow and the LDAP as a way of providing an alternate approach to storing system symbolic destination resolution entries to the existing DB2 table approach.

The implementation uses a host definition name of systemsdefs that contains an object group called SystemSymbolics. This, in turn, contains System objects whose names represent the SystemSymbolics. Each System object contains the queue and queue manager name.

NOTE *LDAP support is limited to Windows NT systems only.*

MQSFSE directory maintenance is illustrated in Figure 15:

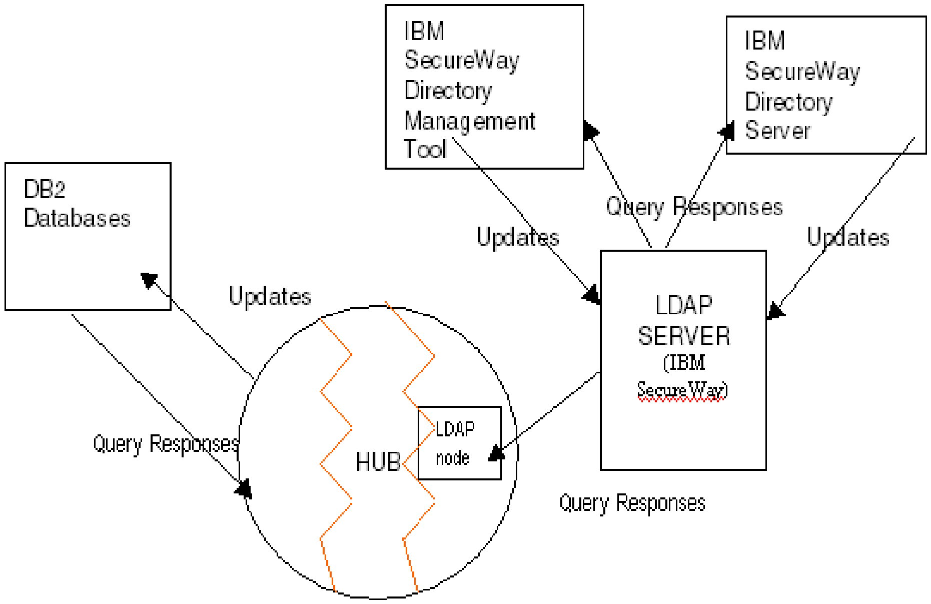
IBM SecureWay Directory Management Tool
IBM SecureWay Directory Server
DB2 Databases
Updates
Query Responses
Updates
Updates
LDAP SERVER (IBM SecureWay)
Query Responses
HUB
LDAP node
Query Responses

Figure 15: Directory maintenance.

Security

Security is an important aspect of any environment. MQSFSE assumes that it is running in a trusted environment, and relies on the MQ Series family of products to provide support for authentication and authorization.

Authentication

MQSeries is the base transport for MQSFSE, and in the default configuration only provides authorization services to protect access to queue and queue manager services. In order to ensure that messages are authentic, MQSeries provides the programming exits that allow custom installation of PKI certificates. This allows MQSFSE to validate the identity of the process that sent a particular message. MQSeries allows messages to be digitally signed and/or encrypted via additional user exits, so that MQSFSE can be certain that the messages have not been tampered with.

Additional MQSFSE authentication features

In addition to the support provided by the MQSeries family of products and associated add-ons, MQSFSE provides simple session management support. As delivered, MQSFSE provides simplistic authentication support via a database table lookup function. However, the expectations are that this function will be replaced by a custom integration to whatever authentication service exists in the enterprise.

Regardless of how authentication functions are implemented, the MQSFSE Message Profile (MP) indicates whether MQSFSE internal session management is enabled for a particular message. When session management is enabled for a message, MQSFSE expects that the client application has initially processed a Hub specific Logon message that contains the appropriate credentials in the MQSFSE header authentication tag. This message is routed through MQSFSE and at implementation time should be integrated with whatever authentication system is available in the enterprise. If the authentication system indicates a successful authentication based on the return of a predefined response message, MQSFSE will generate an entry in its own Session Status table (SS) and return a Session Id in the MQSFSE header.

For all subsequent messages that have the session indicator set in the Message Profile, MQSFSE will expect a valid session entry to be loaded into the MQSFSE header as a part of the request message initialization. If the Session Id is invalid or missing, the message will be rejected. MQSFSE also performs a time-out check to determine if the session was inactive longer than the specified inactivity period. Inactive sessions require a new logon process. If the Session Id is active and valid, then MQSFSE will accept the message for processing.

In addition to the validation of sessions, MQSFSE can execute simple message-to- message affinity checking. The implementation checks to see if a pre-requisite message has been processed immediately prior to processing the current message. Since the session management feature tracks the previous message by intercepting the message at completion, this implementation illustrates how to manage states across independent messages. It is expected that this function can and would be extended as a means of managing state related content across message flows. This is the mechanism intended for helping solve impedance problems between applications designed to support specific message standards and those applications that are being adapted, but where not initially designed to support, these message standards.

The authentication process is illustrated in Figure 16:

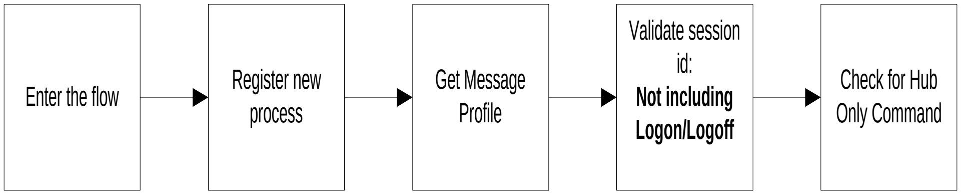
Enter the flow
Register new process
Get Message Profile
Validate session id: **Not including Logon/Logoff**
Check for Hub Only Command

Figure 16: Authentication process.

Authorization

MQSeries provides Access Control List (ACL) support to ensure that the process that sent the message has access only to functions for which sufficient authorization exists. Further MQSFSE is compatible with the the MQAO based logon framework that uses LDAP to store credentials for accessing adapted systems, based on the credentials carried in the XML messages. In addition, the Message profile Defines whether a particular message requires a valid session token as a means of authorizing access to a specific function.

Scalability

MQSFSE provides scalability by leveraging the inherent features of the MQSeries family of products. Queues between front-end and back-end applications can be clustered, MQSI and MQSWF components of MQSFSE can be duplicated, multiple adapters can target adapted applications, and adapters are multi-threaded. As a result, MQSFSE can be scaled in an existing topology, but also can be scaled by migrating to new additions to the topology.

Reliability

The MQSeries family of products are high availability enabled. MQSeries queues assure delivery and provide round robin clustering, adapter frameworks provide redundancy and restart/retry capabilities, and MQSFSE components like MQSI and MQSWF, are transactional and fully restart/retry capable to ensure that the environment produces consistent and timely results.

MQSFSE reliability illustrated in Figure 17:

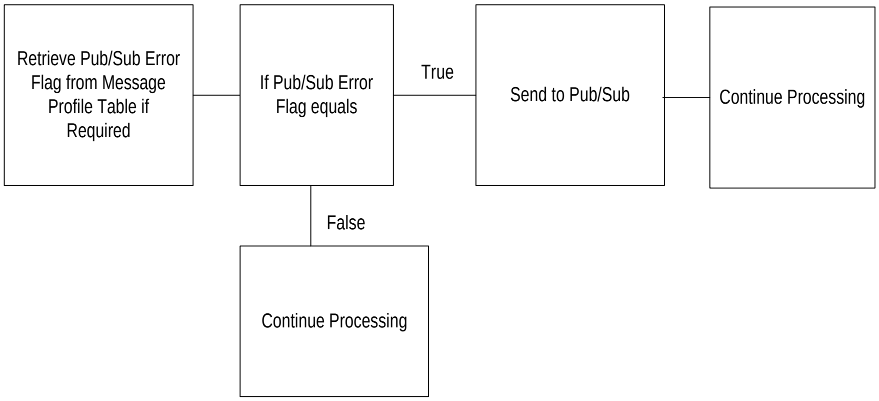
Retrieve Pub/Sub Error Flag from Message Profile Table if Required
If Pub/Sub Error Flag equals
True
Send to Pub/Sub
Continue Processing
False
Continue Processing

Figure 17: MQSFSE reliability.

MQSFSE provides additional features like store and forward and the definition of alternate/mirror applications; as well as comprehensive logging to provide as much flexibility as possible in integrating disparate applications and technologies.

Chapter 3
Recommended implementation

MQSFSE uses a combination of MQSeries services for message transport, message routing, message manipulation, message formatting, lightweight message processing, and for transactional message processing in order to drive the activities associated with implementing business scenarios. An MQSFSE implementation starts with a requesting application , then MQSFSE in the middle, and some number of target applications as the end-points, depending on the business scenario being satisfied. The following sections detail a sample MQSFSE implementation designed to balance complexity with functionality.

Requesting application

The requesting application is any application that requires services from MQSFSE and connected target applications, and is considered the source or originating application. The source application determines which XML messages are generated to trigger a desired set of predefined activities within MQSFSE and related target applications. The source application uses its adapter to format XML messages.

Formatting messages includes populating the MQSFSE message header and MQMD header with information pertinent to the initiating message request. An adapter puts the message on the MQSFSE input queue. If the message is intended to be a fire and forget type message, then processing is complete. If the message is a request/reply type message, then the adapter should monitor its response queue for the correlated response. Monitoring the queue can be done in an asynchronous or synchronous mode depending on what is most suited to the requesting application. Depending on the implementation, the adapter may also have to monitor subscription queues in order to obtain information about add/modify/delete activities on entities for which the application is interested, or to learn about errors detected by the hub during message processing. In this case, the application can use a hub specific message to create the appropriate subscriptions. MQSFSE will create the subscription entries from this message.

In its simplest form, MQSFSE uses entries in the Message Profile, the Symbolic Destination Resolution table, and the Cross Reference Function table to determine how to process the message.

See the MQSFSE publication titled ***Installation and Setup Guide*** for more information on these tables. In order to optimize processing, the content of the XML message is stored in the message cache.

Figure 18 illustrates the application view of MQSFSE:

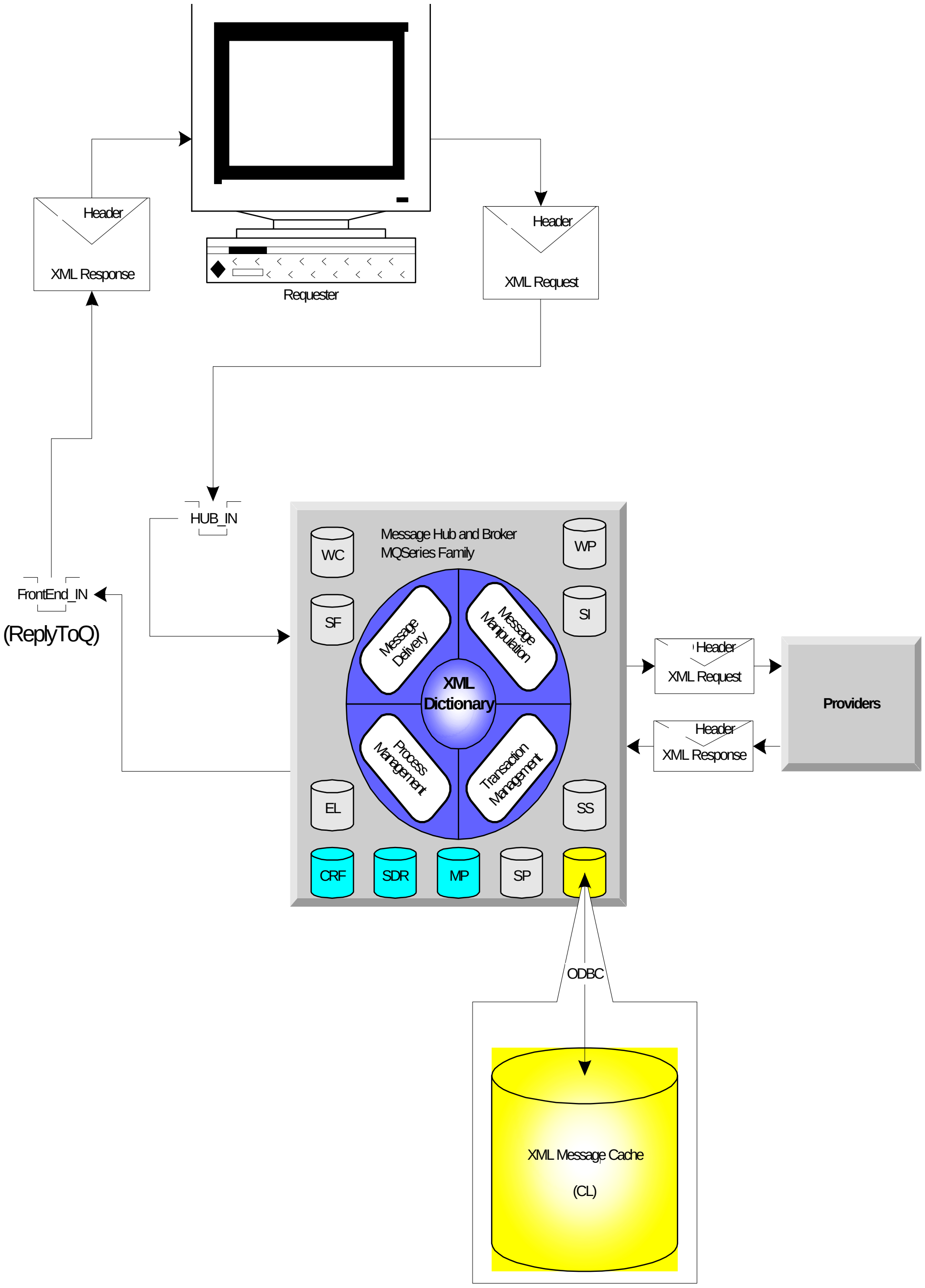

Figure 18: Application view of MQSFSE.

XML process initiation

Once an application has put an XML message on a queue, MQSFSE must interrogate the message profile in order to determine how to handle the message.

- Hub messages are processed against the hub function that they target, with a response message immediately returned to the originator.
- Messages intended for the lightweight message-processor are processed by the HUB_IN_Flow and then delivered to the target application.
- For complex messages, MQSFSE must attach the incoming message request to an appropriate MQSWF process, by using the MQSI HUB_IN_Flow and then activating an MQSWF based workflow template.

Figure 19 illustrates the message flows depending on the configuration for the message in the Message Profile:

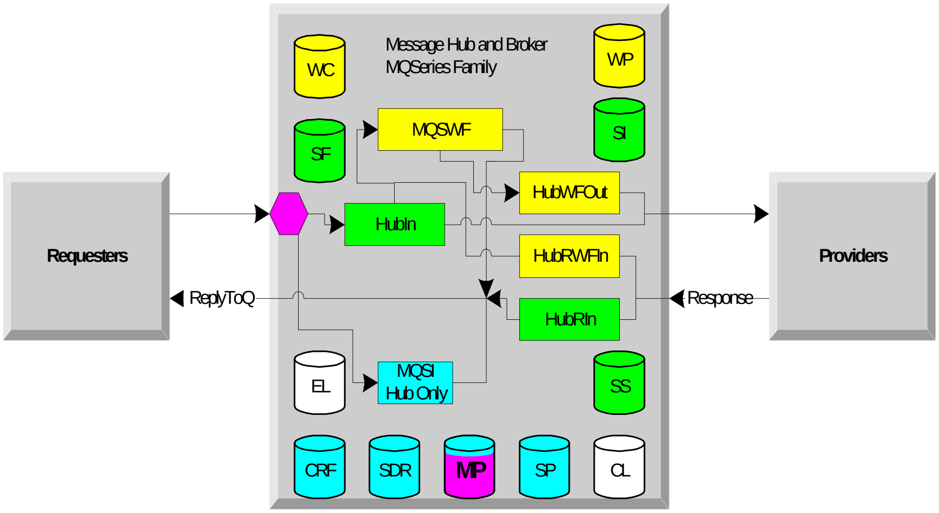

Figure 19: XML message paths.

For both lightweight and transactional message processing, MQSFSE uses the MQSI based HUB_IN_Flow to provide access to MQSFSE functionality. The Functionality provided by the HUB_IN_Flow determines the following:

1. Should a session be validated, created, terminated, or bypassed?
2. If there is an active session, should message state validation take place?
3. Should we check for available systems?
4. Are the appropriate systems or alternatives available?

5. If the appropriate systems are not available, should the message be terminated or should it be stored until a system is available to process the message?
6. Should the message use transactional workflow or should it be sent through the cross referencing and symbolic destination functions so that it can be delivered directly to the target application (lightweight message processing)?

Figure 20 illustrates the processing for both the lightweight and transactional message processing:

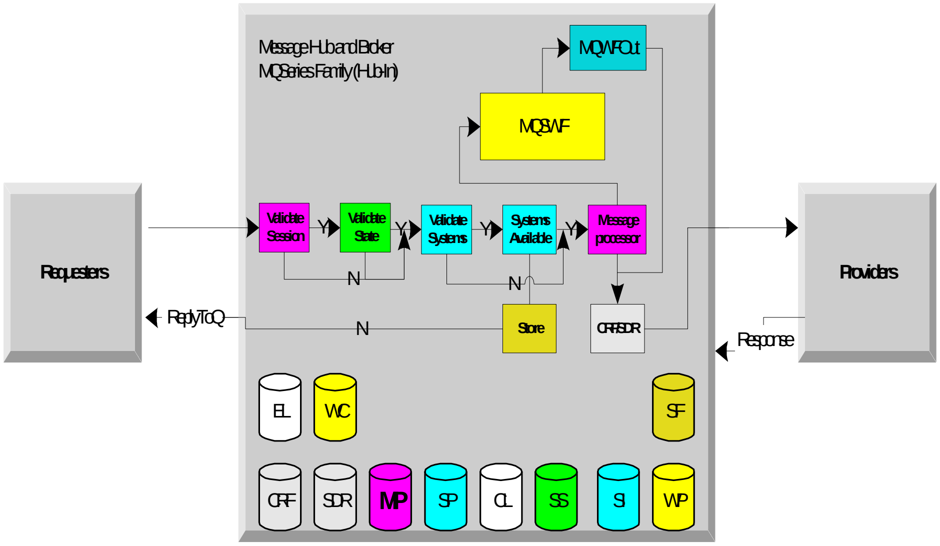

Figure 20: XML message entry to the message processing supervisors.

At this point, the lightweight message-processing supervisor has delivered the message to the target queue. In this case, the processing is dependent on the target adapter to respond in order to activate the return trip component of this supervisor. On the other hand, in order to activate the transactional message processing, the XML message must be wrapped in a MQSWF XML header that provides MQSWF with the information required to initiate MQSWF processing. The MQSI XML to MQSWF XML wrapping process copies control information including the content specified in the Workflow Profile from the MQSFSE message into the MQSWF message.

In general, the contents of an XML message is not needed by an MQSWF process, so MQSFSE stores the message in a database, with the key stored in the MQSWF XML header. This storage allows access to the XML structure when required but maintains a lightweight message container for MQSWF. Once this wrapping process is completed, the newly formed MQSWF XML message, plus any specified message content, and system availability information is put into the MQSWF XML queue so that the appropriate MQSWF process flow can be initiated.

Process execution

After execution begins within the MQSWF workflow template, the response to the original XML message request is obtained by executing the steps within the predefined process graph. There are three types of activities that can take place.

1. Execution of an external application via an adapter XML message.
2. Execution of a workflow function.
3. Execution of a workflow client related activity.

The workflow functions and client activities do not require special support within MQSFSE, but they can be used to provide powerful features and functions as a part of the solution to an MQSFSE implemented use case. Client related functions include implementing manual procedures within the automated message flow. Workflow functions include using the existing Workflow Management Coalition standardized API's to directly access programs and processes. This type of integration can include message content by specifying the correct content in the dynamic container definition stored in the Workflow Profile of the message being executed. Execution of external applications via an XML message is described next.

Execution of external functions

MQSWF process flows are defined at build time to satisfy the request associated with the command name loaded into the MQSWF XML header. The command name is loaded from the MQSFSE Message header bodytype tag in the original XML message that instantiated the workflow. This process graph definition includes steps for invoking the appropriate activities to satisfy the use case associated with the message being processed.

At each activity point, the activity can define a new custom container designed to deliver the information required to satisfy the exit conditions of the activity and or provide data for the next activity. Additionally, the Activity can override content such as the symbolic destination resolution name in order to affect the delivery of the message to a specific destination. If the activity invokes a native MQSWF interface, then MQSWF based implementation rules are used. If the activity is intended to leverage MQSFSE functionality to activate an integrated component of the enterprise, then an XML message is routed to the MQSI MQWF_OUT_Flow using a standard User Defined Process Execution Server.

The MQWF_OUT_Flow provides access to the MQSI based symbolic destination resolution and cross- referencing services required to deliver the message the target application. The MQSI routing capabilities translate the symbolic destination name, coded within the XML message, to the actual queue names associated with the destination applications responsible for providing the services described by the symbolic destination name. For instance, the symbolic destination name may be "customer system of truth". The resolution would be a queue name that has been predefined in the enterprise to deliver messages to a customer system like CIIS. This approach allows applications to be flexibly introduced into the environment without changes to either the application or the workflow template. The only time the process flows are required to change is when the a business process is changed. At this point, container definitions can be changed, messages types can be saved, updated, and or reused, destinations can be added, changed, or removed, and transition checks can be manipulated, all without affecting or requiring changes to MQSFSE or the enterprise itself.

Additionally, MQSFSE will invoke other functions like the MQSI based cross-reference function to translate the keys associated with the data content of the XML commands, based on the XML symbolic destination name. These functions allow the keys to be dynamically translated to the target destination application format and value as needed to support the integration process.

The CRF, logging, and symbolic destination functions are built into the MQSI portion of MQSFSE. They are designed to understand processing associated with both the lightweight and transactional message processing supervisors. One big difference between the two supervisors is the ability to manipulate versions of the message, as well as the content of the messages sent by the transactional supervisor. Regardless of the message sent, every change to a message is logged and stored in the message cache.

An additional option is to publish the message under a topic corresponding to the bodycategory of the message. This allows the message to be directly delivered to the system of truth, and some unknown number of additional interested parties. MQSFSE also provides support for creation of the subscriptions for required topics.

Execution continues once the target adapter returns a response or when the complex message-processing supervisor terminates waiting for a response. In the normal case, processing continues based on reconciliation of the Correlation ID stored in the MQMD header. MQSFSE handles the MQSWF Correlation ID for all MQSI based functions. It is the responsibility of the adapters to process the Correlation Id in the response message based on the parameters coded in the MQMD.

For the lightweight message-processing supervisor, the process is completed with the delivery of the response directly to the originating application, Although if a publication was made, there might be some sort of related activity by a subscriber.

For the complex message-processing supervisor, the workflow template is responsible for interrogating the returning MQSWF XML to ensure that processing completed correctly.

Figure 21 illustrates the steps associated with an MQSWF activity using MQSI based functions:

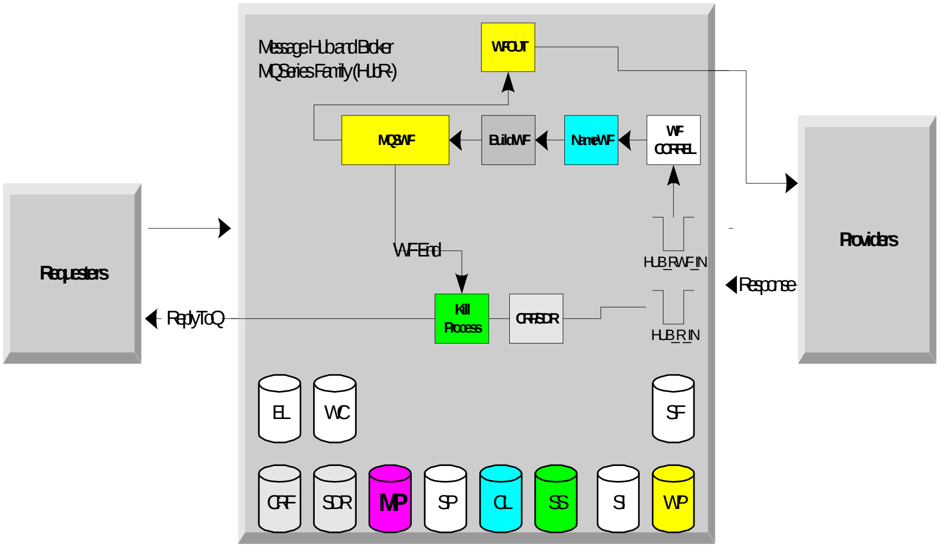

Figure 21: Message processing Supervisors processing a message.

Target application

The target application receives the request for processing. It must respond to the request by providing feedback in the form of an XML response for the specific XML request, based on the semantics of the XML vocabulary being used. It is possible to have the MQSWF workflow process template determine how this response is handled, either by publishing the response so that all subscribers can view the activity, or if a direct response should be routed back to the originator of the request. The appropriate approach depends on the requirements defined in the use case that is being implemented.

It is the responsibility of the target application to ensure that the MQMD Correlation Id is set based on the parameters coded into the MQMD, so that the source application can correlate this response to the original request.

When the complex supervisor is used, a number of activities can be defined for processing a message, and the contents of the message can be used to interrogate the response from the target adapter. When the process graph determines that the appropriate response has been received, then an activity is used to send that response to the source adapter. In this case a flag is set to indicate that this is the response to the source adapter, and the process is handled in a fire and forget fashion. It is additionally possible for the hub to publish this response message so that interested subscribers can gain access.

Chapter 4
Sample scenarios

This section of the document describes a set of scenarios that illustrate how the various pieces of the environment are used to implement an MQSFSE solution. The illustrations are just that, and may never be implemented, or may be implemented differently in MQSFSE depending on the use case details. Some of these examples use MQSI to drive processing, others use MQSWF. This decision is based on the needs of the enterprise in which the implementations are intended to operate.

Client update from front-end system

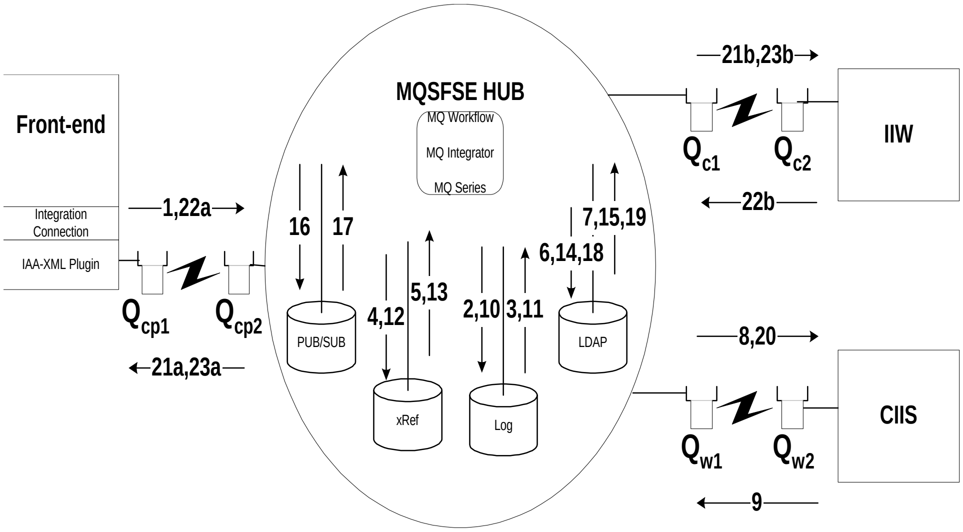

Figure 22: Client update from front-end.

The table below explains how this process works:

Step #	Explanation
1	Front-end sends MQSFSE a client update event.
2,3	MQSFSE logs request, logged.
4,5	UUID translation requested, translated.
6,7	Request resolution for “system of truth” request, queue id returned.
8	Client update request to CIIS or client system.
9	Client information publish request; presume that update is acceptable.
10,11	Request for client update detected, logged.
12,13	UUID translation request, translated.
14,15	Request resolution for publish, queue Id returned.
16,17	MQ/MQSI publishes the client update, published.
18,19	Request resolution for return system, queue Ids returned.
20	CIIS notified that publish request processed.
21a,21b	Subscribers notified of client update via publication.
22a,22b	UUID translation requested, translate.
23b,23a	UUID translation requested, translate.

Table 4: Party update from front-end explained.

This example presumes that the Front-end has a fresh copy of the party, but is not the system of truth, and that IIW, CIIS, and Front-end do not use the UUID as their internal key for the party being updated.

Request for client authentication

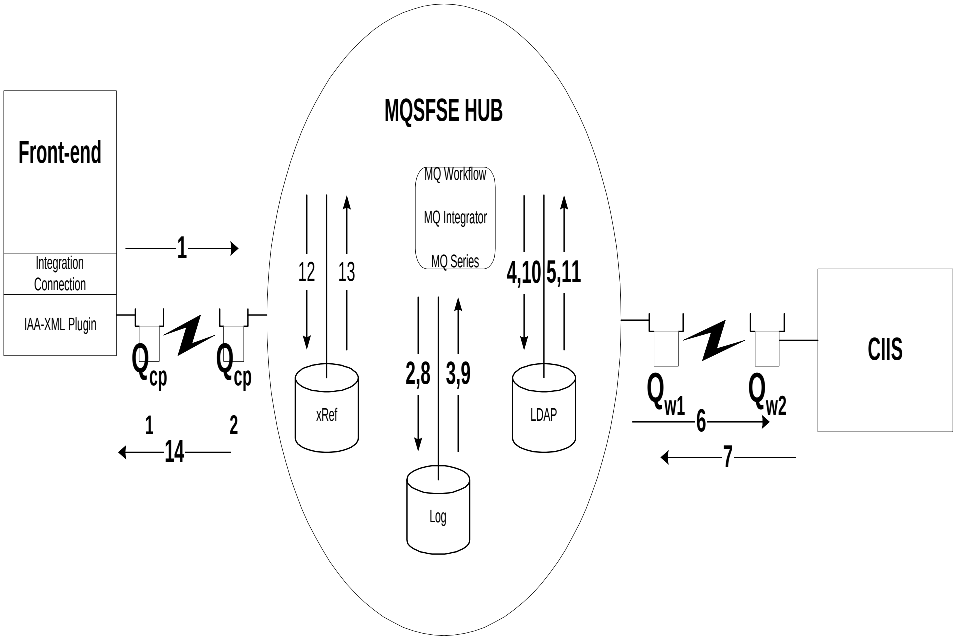

Figure 23: Request for client Authentication.

The table below explains how this process works:

Step #	Explanation
1	Front-end makes a request/response for user authentication.
2,3	MQSFSE detects request, logged.
4,5	Request resolution for authentication request, queue Id returned.
6	Request for authentication.

Step #	Explanation (Continued)
7	Authentication result returned; CIIS validates UID, PWD, domain tuple, returns UUID for client.
8,9	Detected result of authentication request, logged.
10,11	Request resolution for return system, queue Id returned.
12,13	UUID translation request, translate.
14	Authentication request reply returned containing client key.

Table 5: Request for client Authentication explained.

Presumes that front-end system does not use UUID as key for party returned on authentication request.

Client update from front-end with MQSeries Workflow

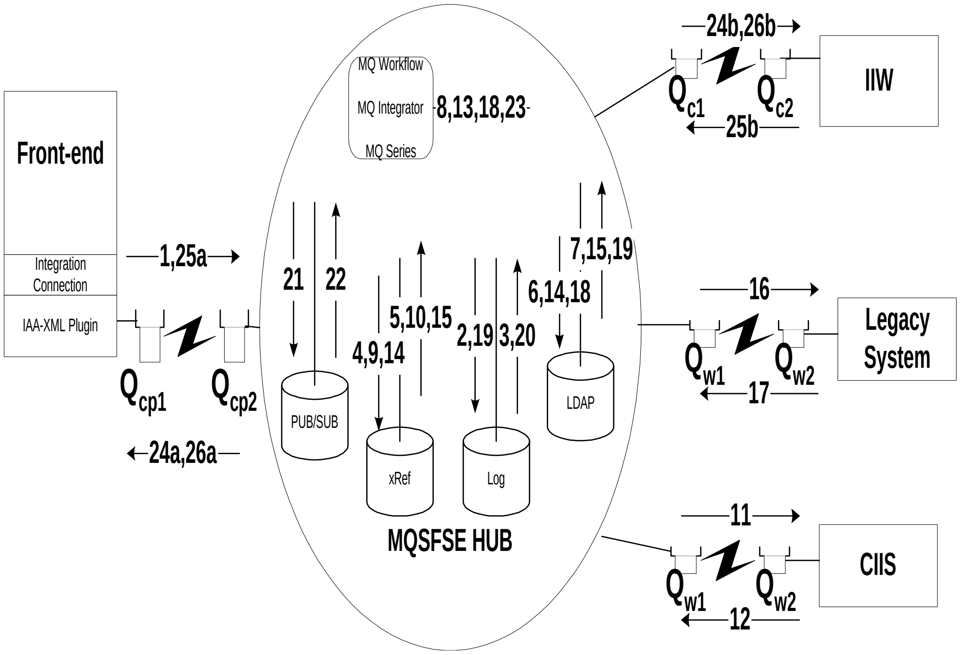

Figure 24: Client update from front-end with Workflow.

The table below explains how this process works:

Step #	Explanation
1	Front-end sends MQSFSE a client update event.
2,3	MQSFSE logs request, logged.
4,5	UUID translation request, translated.
6,7	Request resolution for “system of truth” requested, queue Id returned.
8	Client update request workflow engaged.
9,10	UUID translation request, translated.
11	Client update dropped in CIIS queue.
12	Returned to workflow; presumes that the update is acceptable.
13	Workflow engaged to next activity.
14,15	UUID translation request, translated.
16	Client update dropped in legacy queue.
17	Returned to workflow; presumes that the update is acceptable.
18	Workflow engaged to next activity.
19,20	Request for Client update detected, logged.
21,22	MQ/MQSI publishes the client update, published.
23	Workflow notified that publish request processed; workflow disengaged.
24a,24b	Subscribers notified of client update via publication.
25a,25b	UUID translation request, translate
26a,26b	UUID translation request, translate

Table 6: Party update from front-end with Workflow.

This example presumes that the front-end has a fresh copy of the party, but is not the system of truth, and that IIW, CIIS, and Front-end do not use the UUID as their internal key for the party being updated. In addition, both CIIS and Legacy must be updated cooperatively to stay in sync.

Appendix
Notices

This information was developed for products and services offered in the U.S.A. and Europe. IBM may not offer the products, services, or features discussed in this document in other 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
500 Columbus Avenue
Thornwood, NY 10594
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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 the 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 Corporation
Department: 5EFA/Building: 202
8501 IBM Drive
Charlotte, NC 28262
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 information 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 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.

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

All IBM prices shown are IBM’s suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples 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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© Copyright IBM Corp. 2000, 2001. All rights reserved.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

Trademarks

The following terms are trademarks or services of IBM Corporation in the United States or other countries or both:

- IBM®
- MQSeries®
- DB2®
- IAA®
- Insurance Application Architecture®

OAG is a trademark of the Open Applications Group in the United States or other countries or both.

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

Permission statement

Redistribution and use of this material for both commercial and noncommercial purposes are permitted subject to the below-stated conditions:

1. This Permission Statement shall be reproduced in its entirety in each copy of the material;
2. This material is provided AS IS without warranty of any kind, including but not limited to, any warranty of noninfringement or any warranty (express or implied) of merchantability or fitness for a particular purpose; and
3. The material may be modified provided
   a. Prior written notice of each modification is provided to the Interactive Financial eXchange Forum at the address listed below,

      Interactive Financial Exchange Forum, Inc.
      333 John Carlyle Street
      Suite 600
      Alexandria, VA 22314
      U.S.A.

   b. Any redistribution of modified materials shall be accompanied by a notice that modifications have been made and a clear description of the modifications, and
   c. The party making the modifications assumes all responsibility for the consequences of the modifications.

Glossary

This glossary defines terms and abbreviations used in this book. If you do not find the term you are looking for, see the *Index* or the ***IBM Dictionary of Computing***, New York: McGraw-Hill, 1994.

A

ACL: Access Control List
A list of the services available on a server- each with a list of the hosts permitted to use that server.

Adapters
(1) A part that electrically or physically connects a device to a computer or to another device.

(2) A circuit board that adds function to a computer.

(3) Event Adapter: In a Tivoli environment, software that converts events into a format that the Tivoli Enterprise Console can use and forwards the events to the event server. Using the Tivoli Event Integration Facility, an organization can develop its own event adapters, tailored to its network environment and specific needs.

API: Application Programming Interface
(1) A software interface that enables applications to communicate with each other. An API is the set of programming language constructs or statements that can be coded in an application program to obtain the specific functions and services provided by an underlying operating system or service program.

(2) In VTAM, the language structure used in control blocks so that application programs can reference them and be identified to VTAM.

Attribute
(1) A characteristic that identifies and describes a managed object. The characteristic can be determined, and possibly changed, through operations on the managed object. (2) Information within a managed object that is visible at the object boundary. An attribute has a type, which indicates the range of information given by the attribute, and a value, which is within that range.

B

Business Event:
The series of activities that define the steps required to complete a useful unit of work. Examples are : Add policy, get loan balance, etc. The business event can contain both automated and manual activities, including collaboration.

BPM: Business process management
MQSFSE feature for managing a message through the life-cycle of activities associated with a business event.

C

CAF: Common Adapter Framework

CIIS: Client Information Integration Solution
An implementation of a Party Management System based on the IAA model.

CL: Message Cache
MQSFSE table used to cache versions of messages as they are processed though MQSFE, serves as the historical log.

Class
A UML class. A description of an object.

CRF: Cross Reference Function
MQSFSE function that translates keys from one application into keys for another application as a means of aiding integration between applications.

D

DB2
An IBM relational database management system that is available as a licensed program on several operating systems. Programmers and users of DB2 can create, access, modify, and delete data in relational tables using a variety of interfaces.

DTD: Document Type Definition
The rules that specify the structure for a particular class of SGML or XML documents. The DTD defines the structure with elements, attributes, and notations, and it establishes constraints for how each element, attribute, and notation may be used within the particular class of documents. A DTD is analogous to a database schema in that the DTD completely describes the structure for a particular markup language.

E

EL: Error Logging
MQSFSE function that tracks and records the progress for messages through MQSFSE

I

IAA: Insurance Application Architecture
IBM's business model for the insurance and financial services industry.

L

LDAP: Lightweight Directory Access Protocol
An open protocol that (a) uses TCP/IP to provide access to directories that support an X.500

model and (b) does not incur the resource requirements of the more complex X.500 Directory Access Protocol (DAP). Applications that use LDAP (known as directory-enabled applications) can use the directory as a common data store and for retrieving information about people or services, such as e-mail addresses, public keys, or service-specific configuration parameters. LDAP was originally specified in RFC 1777. LDAP version 3 is specified in RFC 2251, and the IETF continues work on additional standard functions. Some of the IETF-defined standard schemes for LDAP are found in RFC 2256.

LMPS: Lightweight message processing supervisor
Broker based message delivery. Supports all MQSFE functions but only one target.

Log
A record of a sequence of operational activities on a computer.

M

Message Broker
A set of execution processes hosting one or more message flows.

MP: Message profile
MQSFSE table that contains the operational definition of the messages processed through MQSFSE.

MQAB: MQ Adapter Builder

MQAO: MQ Adapter Offering

MQMD: MQSeries Message Descriptor
The MQSeries Integrator (MQSI) header that contains basic control information that must travel with the message.

MQRFH
An architected message header that is used to provide metadata for the processing of a message. This header is supported by MQSeries Publish/Subscribe.

MQSeries
Pertaining to a family of IBM licensed programs that provide message queuing services.

MQSFSE: MQSeries Financial Services Edition
A complete scalable messaging and information integration add-on to the MQSeries family of products. Especially designed for the needs of the financial services industry, MQSeries Financial Services Edition can integrate front-end systems with back-end systems using a hub/spoke architecture using XML as the common vocabulary across systems.

MQSI: MQSeries Integrator
It provides graphical tools for constructing how critical data or business events are handled, by visually connecting a sequence of processing function to dynamically manipulate and route messages, combine them with data from corporate databases, warehouse in-flight message data for auditing or

subsequent analysis, and distribute information efficiently to business applications.

MQSWF: MQSeries Workflow
A business process management system, which facilitates the rapid development and management of the business processes that integrate the IT and organizational infrastructure of a company. It is a client/server system used to design, refine, document, and control a company's business processes using a graphical editor in one of its primary components to facilitate such modeling.

O

Object
Instance of a class.

P

Party
Any person or organization that the insurance company has, or had, or may have a business interest in.

Property
A data value of a type.

Q

Queue
An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages: they point to other queues, or can be used as models for dynamic queues.

S

SDR: Symbolic Destination Resolution
MQSFSE function that translates a symbolic name to specific queue and queue manager name.

SF: Store and Forward
MQSFSE function that stores and retrieves messages in a database table.

SI: System Interaction
MQSFSE function that determines based on parameters stored in a database, whether MQSFSE can deliver a message to the preconfigured targets.

System of Truth
This is a system that is accurate at all times. Any data that is added/change/verified comes from this system. The system of truth is defined for use by MQSFSE, which is the primary system or the system that would hold the most accurate data at any point in time for the systems attached to MQSFSE. It is regarded as the authority for any data being referenced and is the primary system for receiving any data updates.

SP: System profile
MQSFSE table the contains the operational definition of the systems attached to MQSFSE.

SS: System status
MQSFSE function that is used to track the availability of systems defined by symbolic names, using a database table.

T

Tag
An XML construct <Tag....>.

Tivoli
Management Software made by Tivoli Systems Inc.

TMPS: Transactional message processing supervisor:
Assured response message processing function provided by MQSWF for MQSFSE. Supports all MQSFSE functions and multiple targets.

Type tag
An XML tag representing an IAA type.

U

UUID: Universally Unique Identifier
This is a key used by the MQSFSE to uniquely identify the entities which outside systems need to reference.

W

WC: Workflow correlation table
MQSFSE table used to correlate requests and responses for MQSWF activities.

WP: Workflow profile
MQSFSE component of the MP that provides the definition of the initial container used by workflow.

X

XML
eXtensible Markup Language. XML is a markup language for message definition, and is an open and public domain standard. XML is a subset of SGML designed for easy implementation in commercial and web environments.

XML attribute (or just attribute)
Appears in an opening tag, used to specify values in the tag. <Tag attributively'...>

Index