Introduction

MQSeries ® Integrator Version 2.0 is a new product in the MQSeries family from IBM ®. It is a framework designed to enable the provision of mission-critical Business Integration tools and processes. This helps you build your own solutions and enhances your existing solutions by adding functionality to existing programs and data without requiring changes to the existing programs and data. A more general overview on the function of this product, and a couple of potential scenarios for the use of this product can be found in the Introductory White Paper on MQSeries Integrator V2 - The next generation Message Broker.

This White Paper is designed to provide a concise look at the architecture and function of MQSeries Integrator V2 from a technical point of view, explaining what the product can do and how it works.

To this end the Sections in this White Paper are as follows:

- Business Benefits of MQSeries Integrator V2
- Technology & Architecture Overview
- Message Flow Framework and Architecture
- Message Dictionaries
- Message Warehousing
- Publish and Subscribe
- Multi-Broker Domains
- Connecting Applications to the Broker
- Internal Architecture and Administration
- Broker Tools and the Control Center
- System Management
- Security
- Migration Issues
- Summary

What are the business benefits of MQSeries Integrator V2?

MQSeries has helped to create the multi-platform asynchronous messaging marketplace. MQSeries applications can pass messages between queues, which can be located anywhere in a customer's enterprise. Applications can also exchange messages with applications from other companies. As customers look to do more complex tasks and to put their data to more productive use, there is a need to process and transform the message data before passing the message on to the next MQSeries application. The purpose of MQSeries Integrator is to provide a simple yet sophisticated way to enable the customer to process messages en-route to their destinations. This will let the customer keep their focus on how to use this enabling technology to transform their business processes, without needing to change their existing applications. This will protect their investment. MQSeries Integrator V2 by fulfilling these functions, some of which are outlined below, is able to function as a ***Message Broker***. This provides both the MQSeries messaging layer and the Message Brokering hub for processing, transformation and distribution of messages, combining these abilities with a Publish/Subscribe function. Also a brand new graphical tool is now provided as the user interface. This allows full control of the product with the functional construction of solutions achieved by use of a wiring diagram concept.

• Potential Business Uses for MQSeries Integrator V2

- ❖ More efficient connectivity for application integration using hub and spoke model
- ❖ Transformation of data while routing between applications
- ❖ Separation of business logic from application logic and data logic
- ❖ Providing added business application functionality such as Publish/Subscribe
- ❖ Integration framework for adding existing and new vendor products to further add value
- ❖ Seamless integration of messages and relational databases
- ❖ Mapping between XML message formats and other data formats
- ❖ Building on existing MQSeries Integrator V1 applications without rework

By being able to provide these sophisticated functions to transform businesses, MQSeries Integrator can act as a Business Integration and Transformation Engine, becoming an application that increases business value beyond that of the individual functions.

▪ Technology & Architecture overview

Message Brokers act as a way station, or a hub, for messages passing between MQSeries applications. Once messages have reached the Message Broker, they can then be processed, depending on the configuration of the Message Broker and on the contents of the message. Within the Message Broker the individual functions are assigned to a collection of interconnected ***Nodes***, where the processing and transformation activities can take place as required.

A key component of this release is the provision of a framework to allow vendors and other partners and customers to write their own processing nodes.
Other components include an extended Publish/Subscribe facility, message dictionaries and message warehousing. These will be discussed in more detail below.
As this version of MQSeries Integrator follows on from MQSeries Integrator V1, full upwards compatibility has been provided by ensuring that the ***NEON Rules*** and ***NEON Formatter*** have been included in the provided set of Nodes.

• Deliverables

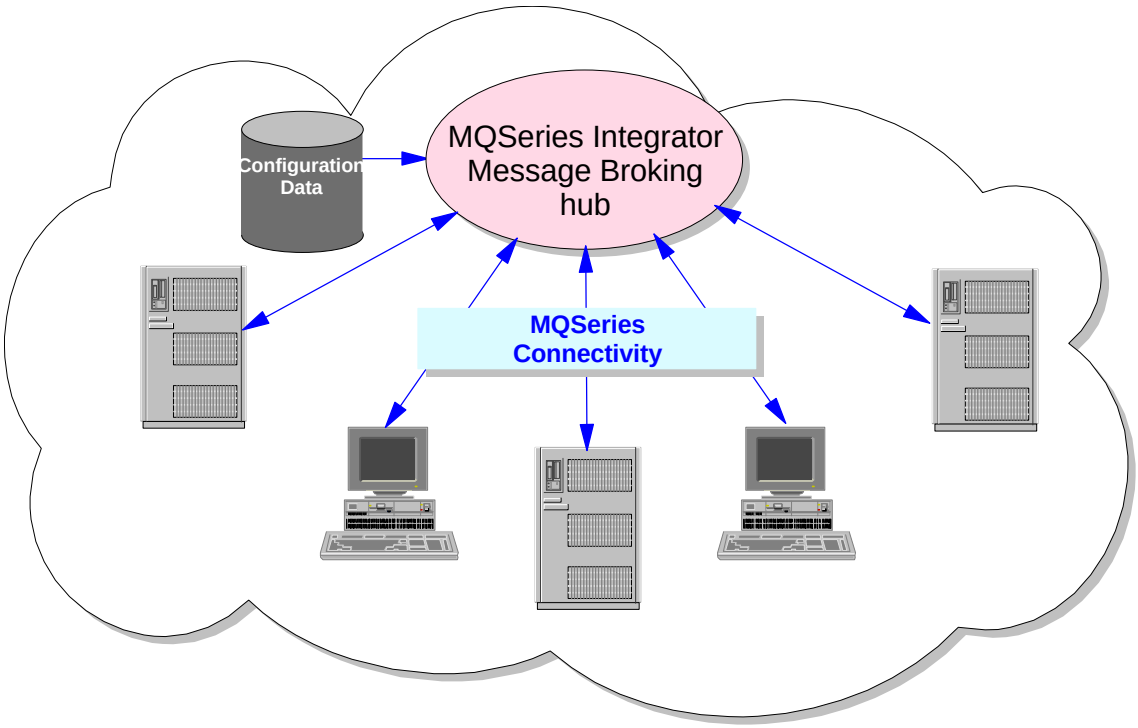
The components of this product can be described as a Message Broker with a ***message flow*** framework, including publish and subscribe management and message warehouse services. These deliverables are combined with the supplied ***Control Center*** tool used to wire together the message flows through the MQSeries Integrator nodes. This allows the fast creation, deployment, and control of message-based business solutions.

This release also includes increased value by adding support for data warehousing, which can be used for audit of message flows, replay and even data mining.
The transformation and reformatting of messages is made simpler through message format dictionaries that provide sample templates of message formats and structures to allow fast handling of parsed messages.

In essence, you are easily creating applications that enable you to transform your business by making use of the application integration framework within the MQSeries Integrator product.

This is instead of creating applications to meet your business need that have to operate within your existing application environment.

Figure 1 - The diagram below shows the routing simplification when interconnecting systems using a messaging hub



- MQSeries Messaging

The Transport layer for the progression of messages into, and out of the Message Broker is the MQSeries product. MQSeries is supplied with the MQSeries Integrator V2 product. Many of the functions of MQSeries, such as assured once-only delivery and transactional support for messages are used in this product. Existing MQSeries documentation, which is widely available, can be used for more information on MQSeries features and functionality.

- Message Flow Framework and Architecture
- Nodes

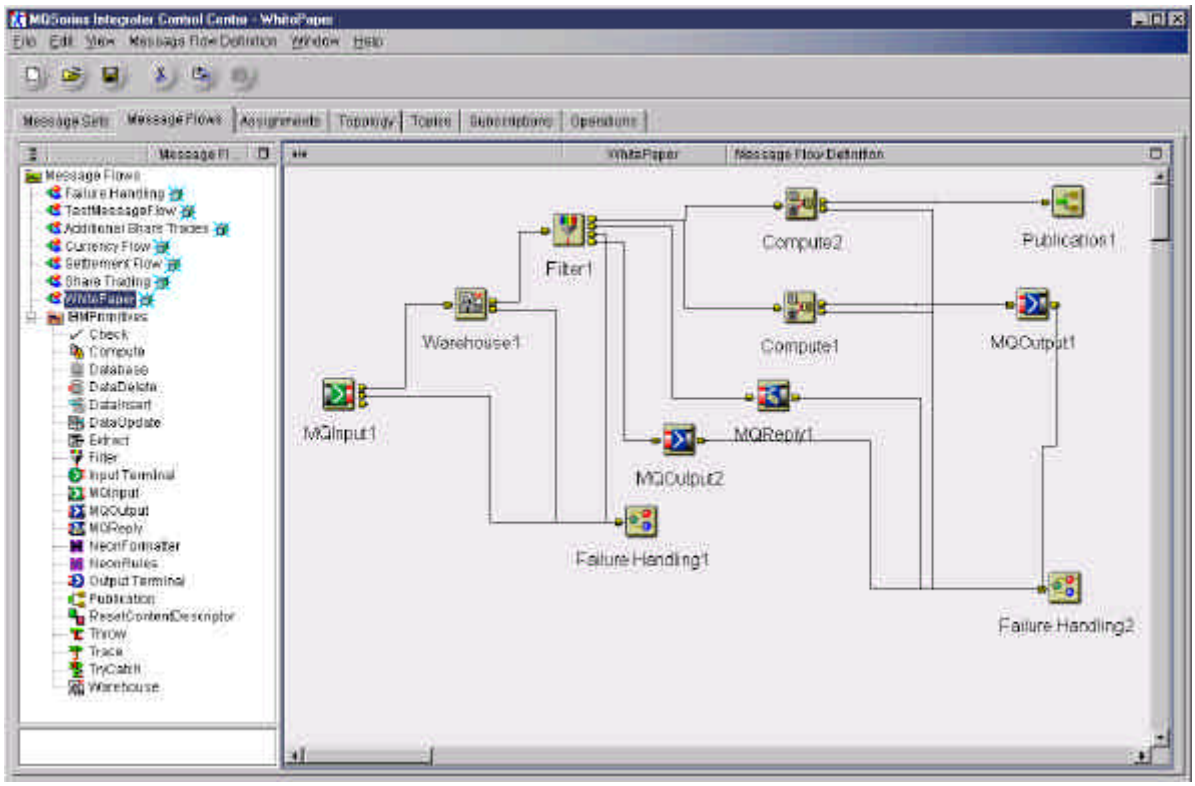
Within the framework of a Message Broker, all processing on the messages is performed within constructs called Nodes, which are actually dynamically linked libraries (DLLs) called from the broker's execution environment. These ***Message Processing Nodes*** perform operations on the message data within the message flow, as well as having the ability to access information outside the message flow, such as accessing a database, or by leaving the message flow by placing a message on a MQSeries Queue. Nodes may have Input and Output ***Terminals***, and the terminals are logically joined up by ***Connectors***. Through the design and configuration of a selection of nodes, a sophisticated processing environment can be built.

Nodes have a set of terminals that are used to receive messages from Connectors or send messages to Connectors. Common types of terminals are ***in*** terminals, which receive messages, ***out*** terminals, which forward on messages, and ***failure*** terminals, which forward on messages when some error has occurred during the processing of the message or an exception has been raised. This allows the System Designer to specify the behaviour of the system when encountering a failure at a particular point in the message flow.

A message processing node is a well-defined processing stage, coded to perform a specific task, or set of tasks on a message flowing through the broker. A selection of predefined nodes comes supplied with the MQSeries Integrator product, including both the NEON Formatter and NEON Rules Nodes. Additional nodes can be created and plugged-in. These nodes can be written in C, or other languages with a C-wrapper, and a sample implementation of a user-written node is provided with the product.

Any number of nodes can be ‘wired together’ using Connectors to form a message flow within an instance of a Message Broker.

Figure 2 - This shows a example message flow with a variety of nodes connected together



- **Message Processing Nodes**

Processing Nodes perform the various operations on messages within the message flow. A message flow is initiated by an input node that starts a message flowing through a message flow. As described below, MQSeries Integrator V2 is supplied with an MQInput node that reads a message off a specified MQSeries Queue. This node will be connected to other nodes. Although the links between Nodes are called Connectors, these are purely constructs to assist in wiring the Nodes together in the Control Center graphical tool. Messages are actually

passed between Nodes by method invocation calls with a pointer to the message object passed between the Nodes.

The properties of the nodes in each architected message flow can be customized. This will enable the function of the nodes to be specific to the messages flowing through the nodes, and to enable the processing required of the nodes in the message flow to be performed.

For example, in a node that will perform a filter operation, the filter statement is assigned by customizing the node specifically for the appropriate message and the filter operation that will take place. In an MQInput node the name of the associated MQSeries queue is given as part of the customized information along with the transactional properties of the message flow.

- **Transactionality and Threading Support**

Processing Nodes do not explicitly maintain the transactional integrity of the messages flowing through the system. Instead the integrity is maintained transactionally within the bounds of the message flow. The message is read from an MQSeries application queue to begin the message flow and placed onto an MQSeries application queue to terminate the flow of the message through the instance of the Message Broker. In between the MQInput node, which begins the message flow, and the termination of the route through the message flow, the message follows the processing route through the nodes in the message flow. If there is more than one connection to an output terminal for a node, and the processing is such that the message will be fired through this terminal, the processing for each different route through the subsequent nodes attached to those output terminals will be handled independently and sequentially. This will maintain use of the same thread within that message flow, until the message flow for that message is complete. At that point the thread is released back into the thread pool.

There can be many instances of the same message flow all processing messages read from the same MQSeries queue. The system is multithreaded and designed to allow many messages to be processed concurrently in multiple instances of a message flow, thus providing multiple instances of each node. It is always in the interests of the application developer designing the messages to flow through the node, and the node developer (if a user-developed node), to ensure that the messages can flow rapidly through nodes. If messages perform lots of different tasks within the message flow, and can become I/O bound, then the performance of the message flow in terms of throughput of messages will be adversely affected.

Each message flow is allocated a pool of between 1 and 256 threads. Each message that comes in is assigned to a separate thread in the following manner.

For each MQInput node in a message flow there is a listening thread. This thread waits for a message to be placed onto the input queue specified to that node. The thread then actions the message, while a second thread is started to take on the role of listening for incoming messages in another message flow instance.

When the message being processed by the original thread completes its processing within the message broker, the original thread is returned to the thread pool for subsequent usage. However, should all threads in the thread pool be used, the next thread to be returned to the thread pool will be used to monitor the input queue.

If a message flow is to be transactional and to include a database operation in the message flow, then the MQInput Node and the Database Node will need to be configured to accept co-ordinated transactions.

- **Supplied Node Description and Functionality**

Up to this point we have been describing the nodes as Processing Nodes. However, the basic supplied palette of nodes are briefly outlined below, with a description of the function attributed to each type of node. All nodes described below have an in terminal, and a failure terminal. Some nodes have a variety of output terminals that vary depending on the type of node. Exceptions will be noted for each node described.

- **Triggering and Initiation**

The only basic node type that offers this is the ***MQInput*** Node.

The MQInput Node uses an MQGET call to receive the message. It then proceeds onwards in the message flow either through the out terminal of the MQInput Node, on to the subsequent node, or to the failure terminal, should an error have occurred. There is a third terminal, called catch. This terminal is fired if the exceptions occur later in the message flow and are not handled closer to the point of failure.

One MQInput Node should be used to take messages off one MQSeries queue. Problems relating to the sequencing of messages could arise if multiple MQInput Nodes were reading from one MQSeries queue and message sequencing is not used.

- **Checking and Filtering**

The basic node types here are the ***Check*** Node and the ***Filter*** Node.

The Check Node checks whether the message's ***'Message Type Specification'*** matches the attributes expected for some or all of the attributes ***domain***, ***set*** and ***type***. This enables evaluation of RFH2 headers and other standard properties. Messages are flowed through the in terminal. Should the check be successful they are flowed through the match terminal; otherwise, they are routed through the failure terminal.

The Filter Node is a content based evaluation of the input message, using an SQL expression as the decision criteria. The possible terminals for this node are in, ***true***, ***false***, ***unknown*** and failure. The explanations for the terminals for in, true and false are hopefully self-describing. The message is flowed to the unknown terminal if the result of the evaluation is indeterminate or unknown. If a failure occurs (such as an arithmetic overflow) during the evaluation, the message is routed through the failure terminal.

- **Message Manipulation**

The basic node types here are the ***Compute*** Node, the ***Extract*** Node, the ***NEON Formatter*** Node and the ***ResetContentDescriptor*** Node.

The Compute Node is designed to be able to transform a message, with the ability to accept one Message Type as an input and after transformation to output a different Message Type. This is done using the contents of the input message and optionally data values from an

external relational database. Each element of the output message, which can be entirely different from the input message, can be derived using a specific formula, with the language used to specify the query closely based on SQL3.

The Extract Node will produce a transformed output message from the input message by selecting, copying, and also modifying elements from the input message to create a new output message.

The NEON Formatter Node invokes the NEON Formatter engine to map the content of the input message to the output message. By using this node a message in one format, defined to the NEON Repository can be transformed into another message format as defined in the NEON format definitions. This node can be used either on its own or in combination with other nodes to build up a comprehensive message flow.

The ResetContentDescriptor Node is designed to allow the message to be interpreted by another parser type to within the same message flow. It performs the same function as if the message was passed from an MQOutput Node to an MQInput Node.

- **External Database Operation**

The Basic Node types here are ***DataInsert*** Node, ***DataUpdate*** Node, ***DataDelete*** Node, ***Database*** Node and ***Warehouse*** Node. All these Nodes are specialisations of the generic function of accessing a database. All of these Node Types have terminals as follows: in, out and failure. All the operations using these nodes can be part of an externally co-ordinated transaction, or they can commit the transaction independently. None of these nodes alter the message the flows through them. The input for the DataInsert, DataUpdate and DataDelete nodes must be typed, but the input for the Database node can be generic XML.

The DataInsert Node can do a single insert of a new row into a specified database. Some of the information from the message may be used as part of the insert or the message may just be used as a trigger, possibly following a filter node. An internally generated SQL statement is used to do the insert.

The DataUpdate Node will update the values in one or more rows of a specified database. An internally generated SQL expression is used for the update.

The DataDelete Node can delete one or more rows from a table in a specified database. The message is unchanged in the process. Data from the input message can be used in the expression to specify what data is deleted from the database.

The Database Node can perform a database operation on a specified database, without changing the message, which passes through from the in terminal to the out terminal. Values from the message can be included in the SQL expression to execute the database operation.

The Warehouse Node, which is similar to the DataInsert Node, is used to store the messages flowing through the Broker in a Message Warehouse. The messages stored in the warehouse may be stored there for audit purposes, or for off-line or batch processing of messages, or for subsequent retrieval and processing by the Message Broker. The message is added to the database in the warehouse using a SQL insert. The message is stored in the database with an

index record built from the message schema. The message itself may be attached to the index record and stored as a ***BLOB*** (meaning that the individual elements can't be accessed).

- **Decision and Routing**

The basic supplied node types are the ***MQOutput*** Node, the ***MQReply*** Node, the ***NEONRules*** Node, and the ***Publication*** Node.

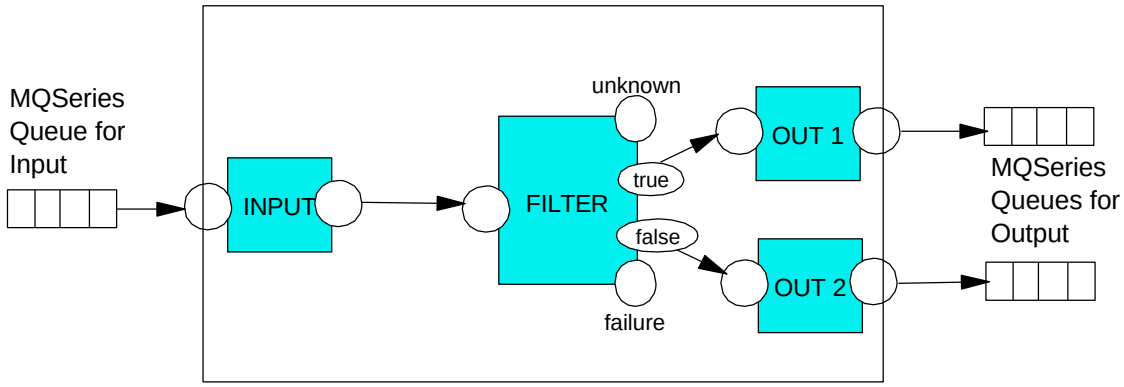
The MQOutput Node is a defined endpoint of a message flow within the Message Broker. On leaving this node, messages are written to a MQSeries Queue using a MQPUT MQI call. They can either be written to a specified fixed queue, or sent to a reply queue, or a list of destination queues can be specified, using information from the message.

The MQReply Node is a specialised version of the MQOutput Node. It is used when the MQSeries Queue that the message is to be outputted to is the one specified by the ReplyTo field of the message header.

The NEONRules node is used when the message needs to be passed to the NEONRules engine for evaluation of rules. The node is therefore a driver for the Rules engine, replacing the Daemon supplied with MQSeries Integrator V1 within the new product. The NEONRules node evaluates messages against the defined rules and will fire one of the terminals depending on the results of the rules. If a message is routed to the PutQueue terminal this message could be routed straight to a node such as MQOutput to provide the same functionality as available in the V1 product. Similarly the failure terminals and noHit terminals could be routed to MQOutput nodes. If a message leaving the node is to be written to a number of different queues, then the destination list feature is used. The message leaving the node need not be written out to a queue immediately but can be routed to any other node in the message flow. There is a fourth terminal, called propagate, on the NEONRules node. This is used when the rules evaluate but no output queue name is added to the message leaving the node and the system designer is looking to route the message through to other nodes in the message flow for further processing.

The Publication Node is used in the transmission of messages to subscribers of the defined Publish and Subscribe service. Messages routed through this node as part of a Publish/Subscribe message flow are matched to subscribers for both topic and content and then forwarded directly to local subscribers or routed to other brokers to match their subscribers that are remote to the publishing broker. Management of the published Topics and the subscribers are handled elsewhere in the Control Center.

Figure 3 - This illustrates the integration of MQSeries Queues with the nodes in a message flow



- **Error Handling and Trace**

There are three basic Node Types for this area. They are the ***Throw*** Node, the ***Trace*** Node and the ***TryCatch*** Node.

The Throw Node has just an in terminal and is used within the message flow to throw exceptions. These may be caught by TryCatch nodes earlier in the message flow or may cause the processing of that particular message to cease and associated transactional activity to be rolled back. The Throw Node may be used to throw an exception based on message content to prevent additional failures downstream in the message flow.

The Trace Node is used to aid in debugging the message flow. It has an in terminal and an out terminal. The out terminal passes on the input message unmodified. However, the Trace Node will format and write out a trace record to a specified destination, assisting in creating a record of the route a message has taken through the message flow. The trace format can be selected using a variety of options.

The purpose of the TryCatch Node is to prevent exceptions of downstream nodes from terminating the processing of messages or transactions, which is likely to happen if the exception drops back to the MQInput Node, which is the root node in the message flow. The message is received by an in terminal and forwarded on unchanged using the try terminal. If the exception is caught by this node it will be propagated using the catch terminal if this is connected. Error handling of the exception can then occur.

If exceptions occur in the message flow have not been caught by other nodes, such as a TryCatch Node, the MQInput Node will catch the exception, as it was the initiator of the thread and the catch terminal on the MQInput node will propagate the message.

- **3rd Party or Plug-In Message Processing Nodes**

The Nodes described above are those which are supplied with the MQSeries Integrator V2 product. Additional Processing Nodes may be provided by 3rd parties or ISVs to enhance the message flow processing within the Message Broker. In the future IBM may provide extra nodes to join the other supplied Primitive Nodes. They may add new functionality or enhance and replace existing functionality. These nodes must be designed to match the requirements of the Message flow Framework that will allow the new nodes to be added to the MQSeries Integrator Design Tool. This compliance is provided by supplying a ***signature template*** in the

form of an XML representation. The Node is likely to be written in C and distributed as a Dynamic Link Library on Windows NT ® or a Shared Library on UNIX ®.

- **Transactionality within the Message Broker**

Given the importance of any work driven through the Message Broker it can be vital that if required, all message flows are given full transactional integrity. Part of the base MQSeries function allows MQSeries messages to participate in co-ordinated transactions, thus ensuring the transactionality of the message within the system. By using this facility, messages within the Message Broker can also gain transactionality for their message flow. The key part of this behaviour is because the message flow is initiated by reading a message from a MQSeries queue, doing some work on the message, or using the message to drive some work such as working with a database, and then placing a message on a MQSeries queue. If there is any failure between doing the MQGET to read the message from the queue and doing the MQPUT to place the message and the failure or the exception is not handled, the entire operation can be rolled back to the MQGET. This can be done as no operation is committed until defined to do so. Operations on external relational databases use co-ordination to ensure transactional behaviour, with distributed systems, that support it, using the XA interface.

The flow of transactional behaviour between the nodes is controlled by setting attributes on the instance of the message flow. The key attributes of interest to this functionality are to enable the system designer to set the transactionality of the message flow. Therefore, in the MQInput node there is a transaction attribute that determines whether messages flowing through the message flow will be handled transactionally. In the MQOutput node there is a persistence attribute to determine the persistence state of the outgoing message. The Database nodes have a co-ordination attribute to determine whether access to the database will participate in the transaction.

These attributes can specify factors such as whether work written to a database will participate within a transaction, how and when a transaction will be committed or rolled back, and whether the message within the message flow will be sent persistently between nodes. A comprehensive approach must be applied by the system designer to ensure that the appropriate level of transactionality is applied to each node and collection of nodes and to ensure that all participants, such as a database are appropriately configured. By using an appropriate combination of these attributes on the message flow, and the relevant nodes, a message flow can be transactional for its entire span, or can be transactional only in accessing databases, or it can even be non-transactional.

Message Dictionaries

- **Message Dictionary Benefits**

As has been established above, a key part of the function of the MQSeries Integrator V2 product is the ability to parse the contents of messages to either perform work on the message data or to allow the message data to drive work externally. In order to efficiently parse this data it is essential to be able to look up message formats in order to identify the relevant fields in each message for every node based function. The Message Dictionaries therefore are used to provide format information giving the ability to rapidly parse information from messages, held in the dictionary as a logical message model for direct access to named fields in the message body.

With a message dictionary format provided to give templates for expected message types, the required element fields can be extracted from messages rapidly.

- **Components and Functionality of a Message Dictionary**

A Message Dictionary provides the Message Services Component with support to parse the different formats of message contents and their associated headers that are defined to the Message Dictionary. These formats can include ***MQMD*** message descriptors, MQSeries ***RFH*** and ***RFH2*** format headers, ***XML*** messages, other wire level formats, and messages built according to NEONFormatter definitions. These formats held within the dictionary can be collectively known as ***Message Type Definitions***. Within the Message Dictionary, these definitions can be grouped together as ***Message Sets***, and the Message Sets are deployed to the Brokers that will be processing the messages within these Message Sets.

The three main components of a Message Dictionary are a ***Message Repository Manager*** (***MRM***), a ***Resource Manager***, and a ***Message Translation Interface*** (***MTI***), but to a User the Message Dictionary is accessed from the GUI and the components are not exposed.

The definition for the format of the messages, with identification of the fields and elements within a model message template, is known as a ***Message Model***. The MRM uses the MQSeries Integrator V2 Control Center tool to define and maintain the Message Model and stores its information in the MRM Database. The Message Models in the MRM can handle many forms of messages such as XML message formats and byte-oriented record structures from C or COBOL sources.

This information is supplied to the Brokers to be held in a ***RunTime Dictionary*** (***RTD***), locally available to each deployed broker. These enable the Message Brokers to maintain a local cache of the format definitions and thus improve the speed of parsing the message formats. If the information in the dictionary is to be changed while a copy is held in the RTD, a new version of the dictionary is added, as dictionaries can’t be modified once deployed.

- **Uses of the Message Dictionary**

The section so far has described the function of the Message Dictionary as a store for message formats for when messages are received by the Message Broker. The Control Center is used as the definition tool for defining the message formats the Broker is expecting to receive. The messages to be received by the Message Broker have their formats defined within the Control Center that is the interface of the Message Dictionary. These defined formats are then used in conjunction with the processing nodes and parsers to provide the logical message formats used by the Message Broker from the wire format messages received via MQSeries.

The messages are interpreted from the wire format descriptor with the aid of the message definition. The format of the message, whenbeing defined using the Control Center, needs more than the types, the elements (or fields), and the associated lengths of the message in order to match the logical message format to the wire format.

The Message Broker does not just use the Message Dictionary for providing the logical message format from the wire format. The Dictionary is also used in the reverse way to take

the logical message formats received by the MQOutput and Publish Nodes and to create wire format messages from those logical message formats.

This can be explained clearly using an example where the content of a message is generated by a COBOL program and the content of the message is a COBOL record structure. The logical message structure held within the COBOL record can be defined to the Message Dictionary using the GUI. When the message is received by the Broker this definition is then used to deconstruct the message into the relevant fields. However, when processing is complete within the Broker, and the message needs to be sent to another COBOL application as a message, the outgoing message needs to be created in wire format with the logical format becoming a COBOL record structure again. Once again the Message Dictionary is used to map the fields of the message, but this time it maps the message fields in those of the COBOL record, creating the wire format as defined in the Message Dictionary. Naturally, one of the strengths of the logical format is that if the message is not being sent on to a COBOL application, but instead is being displayed on a Web Page and needs to be sent on in HTML, the definition within the Message Dictionary will build the output wire format as required, taken from the definition in the **Message Template**.

- **Message Templates**

With the multiplicity of different formats of messages that could flow through a broker, there must be a way of distinguishing between the types of messages. The definition of each type of message, or related group of messages, is described as a Message Set. These Message Sets are assigned into a Message Dictionary, with a separate Dictionary for each Set. On receiving a Message, the information in the message header will help to identify the correct dictionary to load in order to parse the message. For deployment to brokers, the entire collection of related messages grouped together into a Message Set is assigned to the broker or brokers that will be receiving the messages for processing in the assigned Message Flows.

The information within the message header that provides these details, and is defined to the Message Broker using the Control Center is as follows.

- The message domain, which describes the source of the message definition: that is, whether it has been defined by the Control Center tool or the NEON tool.
- The message set, or project, groups together a collection of messages, elements and types, within the specified domain, going to make up a complete definition of messages relating to a particular flow or business operation.
- The message type that will precisely define the structure of the data within the message, giving such details as the number and location of character strings.
- The message format that identifies the wire format of the message.

These definitions will be required for each type of message (other than self defining formats such as XML), expected to be received by any message flow. When the definitions have been completed for each message, with the message set defined to a Message Dictionary, and the appropriate Dictionaries assigned to the message flow, then when a message is received, its type is identified by the information in the message header, and the appropriate Message Dictionary is accessed and finally the **Message Parser** is called to deconstruct the message.

Careful thought must go into Message definition with the MRM. If messages are just being routed through the Broker with no transformation then only one message needs to be defined,

as the output message is no different from the input message. If there is any transformation, with elements being added or removed from the message, then both input and output message formats should be defined to the MRM. When both input and output message formats have been defined, then when transformation of the message takes place, perhaps in a compute node, then the compute node will take the input format as the inbound message and the message will be transformed into the output message format according to the customized properties of the node performing the transformation. Note that the MRM does not distinguish between input and output message definitions. Whatever the sequence of messages used in the message flow, all messages are defined in an identical manner.

- **Message Parsers**

Once the template has been defined the Parser is essential to establish what is to be done to the received message. The Parser can be one that is supplied with the product, or it can be a user supplied or a 3rd party supplied Parser. The behaviour and the Parser can be different, dependant on whether the message has been defined to the Message Broker using the Control Center or whether it has not been defined using the Control Center.

New messages defined in the Control Center, defined as parts of message sets are created to be logical message structures that can then be used in transmission to other systems. Supplied parsers for these messages will work on messages that use the following types of message headers: MQMD, MQRFH, MQRFH2, MQCIH (CICS ® bridge header), MQIIH (IMS ™ bridge header) and MQSIH (SAP bridge header). When the incoming message has been defined in the Control Center, but is generated by an application as a data structure, the parser can be specially constructed to deal with this or the data structure can be imported using the facilities of the Control Center.

Validation of message content is very useful when dealing with messages that have been defined to the Message Dictionary with attributes called **Valid Values** assigned to the message fields.

Should the message not be defined using the Control Center, there are a number of options. The message could be in Generic XML, in which case it is self-defining and does not require a Dictionary definition. Should this be the case, the message cannot be validated to ensure it is in the correct format.

The message could be defined using the NEON tools and be intended to be processed using the NEONFormatter. In this case the message flow definition will be designed to handle this scenario by routing the message through a NEON node, and the format can be established using the NEON tools. If the message structure is not defined anywhere, it is treated as Opaque content and no content based processing can be performed on the message as it flows through the Broker, but it can be acted on according to the information in the Message Header and the design of the Message Flow.

- **XML and the Message Dictionary**

If no template, or schema, exists for an XML message, but the message uses well-formed XML, the message can be termed self-describing and content based routing, based on the XML descriptors, can be applied. In this case, with a well-formed XML message, the message flow can still be architected, and nodes, such as the compute node can be customized to

handle the elements in the XML message, even when they haven't been defined to the Message Dictionary. By using the message dictionaries for non-XML messages, which are not self-describing, allow content based routing based on the message dictionary's description of the content, once the message format has been defined to the Message Dictionary.

XML is used at the heart of the MQSeries Integrator V2 product. All configuration data for the product is held in XML format. Once a message is defined to the MRM as a message format, the format of the message to be output can be also defined with the data format for the output message being XML as required. Thus XML messages can be generated within the Message Broker from non-XML based message formats.

▪ Message Warehouses

• Reasons for Message Warehousing

Message Warehouses can be used for many different purposes. However, a fundamental use is that they are storing long-lived messages in a standard relational database. These messages may then be used for purposes such as laying an audit trail for a particular type of message (such as high value messages). Message Warehousing may be a standard method of logging the work being performed within a particular broker, possibly for off-line analysis. Another possible reason for Data Warehousing could be to implement a full Data Mining or Data Analysis on the data flowing through the Message Broker.

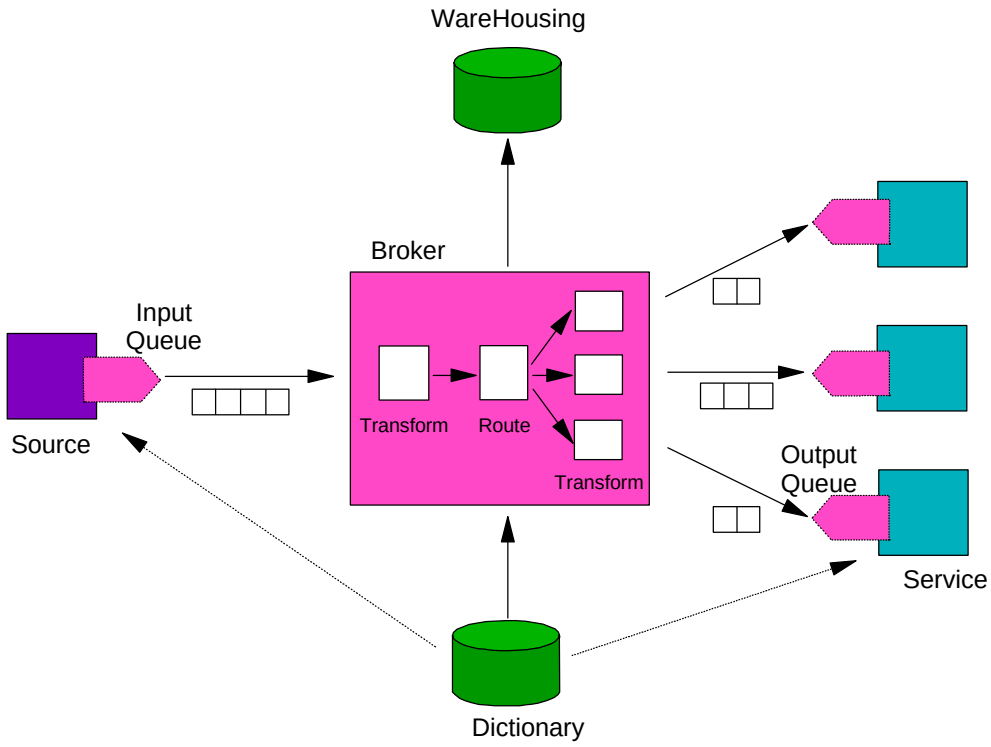
• Warehousing Nodes

In order to actually place a message within a Message Warehouse, a node must be configured to do so. This message flow is likely to perform some calculation or transformation on the message before moving the message to the Warehouse. The Warehouse Node may parse the message, or parts of the message that are to be moved to the Warehouse. The required format of the message and its data contents are created and passed into a built SQL INSERT statement for entry into the database.

• Message Content and Format

The format of the message content can vary widely, and the system designer will have to evaluate the required format from the eventual use of the data in the messages in the Message Warehouse. It is possible that the Message Warehouse could just be used as a temporary store for messages and the database needs no knowledge of the contents of the message. In which case the message could be stored in the database as a ***BLOB*** with no ability to extract information from the message other than by passing the message back to the Message Broker or other messaging services. Alternatively the database could be required to do complex processing on some of the elements within the fields in the message. In this case then the database would need to be aware of the headers and fields of the message in order to work with the stored data. Any design that is put in place by the system designer should be appropriate to the needs of the application, and likely future uses of the data.

Figure 4 - This diagram illustrates the message flow driving access to an external database for Data Warehousing



- Publish Subscribe System

- **Content and Topic based filtering**

The addition of a Publish and Subscribe function to the Message Broker increases the functional value of such a system. By combining the strengths of a Publish/Subscribe broker with the functionality of the Nodes within the MQSeries Integrator hub, extra benefits can be realised. For the subscribers of information, they can not only receive just the information they have requested but also the information can be filtered and formatted at any level to meet their needs. For the as with a standard Publish and Subscribe system, publishers it means that their information can be made available without needing to know who will be requesting it and therefore to whom to send the information.

The usefulness of a Publish/Subscribe system is greatly increased by increasing the refinement of the selection criteria. There can be a large number of messages for every topic that are not required by subscribers, even when the topic matches their request. Thus by improving the refinement of the selection of the messages to be sent to subscribers, by allowing content-based subscription, a more selective and therefore more efficient method of distributing information is provided.

The MQSeries Integrator V2 product provides this content-based filtering on subscription, as well as the hierarchical topic based filtering that is available on both publication and subscription. For content-based filtering, the contents of elements within a message are evaluated by SQL expressions to establish the content filtering result. The content filters can be stored in the Dynamic Subscription Table. These filtered messages, when combined with the other supplied or defined Message Broker functions can be transformed for different applications and only the required parts of messages sent to the applications that subscribe to

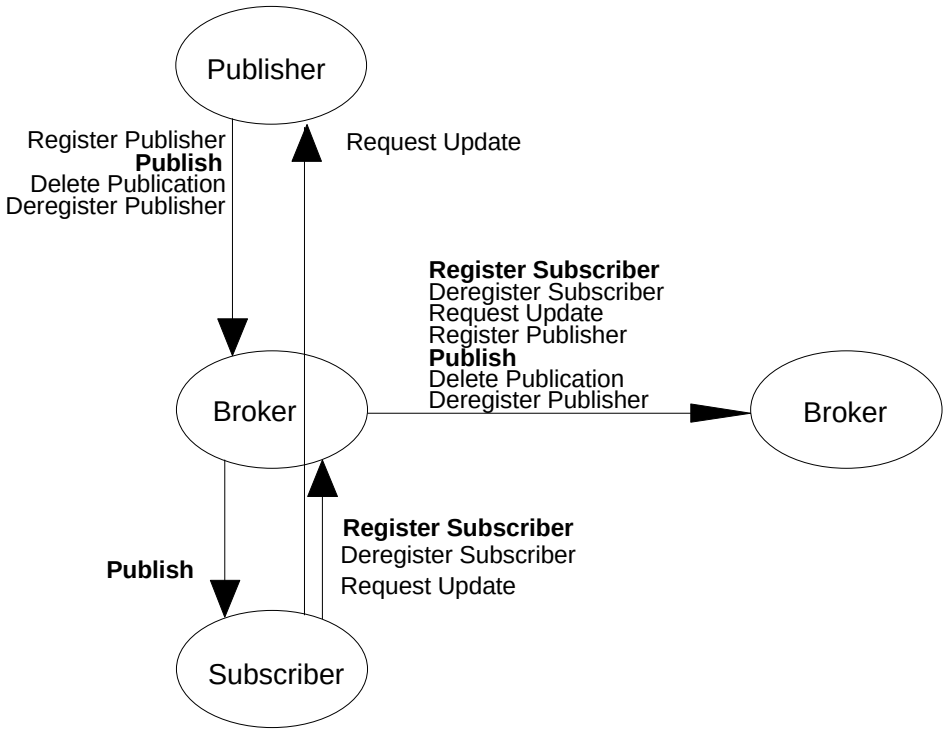
them. The Publish/Subscribe function is represented as a Node, known as a Publication Node within the message flow. The end result of a Publish/Subscribe Node in action is the placing of a message on one or more MQSeries queues.

- **Subscription Handling**

If subscribing applications for published data are known in advance of the publication, this is known as ***static subscription*** and routing can be well defined in advance. If the subscription set can be changed by subscribing applications adding subscriptions or changing subscriptions, this is referred to as ***dynamic subscription***, which is very flexible in terms of changing business requests within a system during runtime with no need to pre-register interest in particular message types. MQSeries Integrator V2 supports both types of subscription.

The requests of subscribers to the Publish/Subscribe function take the form of messages known as ***control messages***. These give the subscriber full ability to create, delete and change their subscriptions. The names for these messages are: ***Register Subscriber***, ***Deregister Subscriber*** and ***Request Update***. For publishers there are different messages to meet their different needs. These message names are: ***Publish*** and ***Delete Publication***. These control messages are only required if the client application is using the MQI programming interface and not the MQSeries Application Messaging Interface or the MQSeries Java™ Messaging Service, as the MQI will need to build the headers explicitly to call these functions.

Figure 5 - Here is an illustration of the flows involved in the handling of dynamic publication and subscription requests



The list of subscriptions is held persistently within the Message Broker. Changes made to the subscription list are updated dynamically, and will take effect as soon as the message reaches the broker. The key items in determining a Publish/Subscribe action are as follows: topic,

content, subscription point and destination. Any combination of these four pieces of information can be used to create a unique subscription.

Where multiple Message Brokers exist, subscription information can be shared between brokers to ensure that messages flowing into other brokers, which fulfil the needs of subscribers on another broker, can be forwarded to the interested parties. In the case where multiple brokers have subscriptions to a message flow, the updating of subscriptions is also dynamic, with implementation of changes in subscriptions being passed as quickly as a message from one broker to another.

- **Collectives**

When defining brokers within a Publish/Subscribe network where users may publish information at one broker and other users may subscribe at other brokers it can be more efficient to connect and group brokers together. In MQSeries Integrator V2 the architecture to do this is termed a Collective, and there is a section on Multi-Broker configurations and Collectives at the end of this paper.

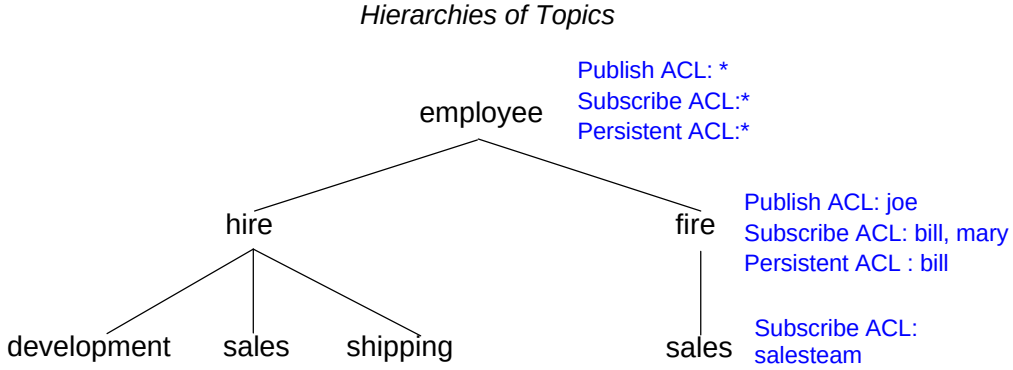
- **MQSeries Integrator V2 Publish/Subscribe Security**

The ability of users to publish information, or subscribe to information depends on the setting of the Access Control Lists (ACLs). The ACLs are set on topics to which the message is published. Publishers must have ACL permission to publish to the required topic. Subscribers must have ACL permission to subscribe to the required topic. Subscribers may request to receive persistent messages, but if denied by the ACLs they will still receive the desired messages, but will not receive them persistently.

Topics are organised into a hierarchical tree structure. This leads to downstream topics inheriting ACLs from root topics, unless explicitly stated. This can mean that if a leaf topic does not explicitly state the ACL permissions then the ACLs are derived from the higher topics, ultimately using the root ACLs if no other ACLs have been found in the topic tree.

Wild card topics can be used by a subscriber, and the security policy handles this by applying the policy to the individual leaf topics that are matched by the wild card topic, and ensuring that the ACLs for each particular topic are fulfilled.

Figure 6 - This diagram is to demonstrate the hierarchy of ACLs through the Topics



- **Comparison with MQSeries Publish/Subscribe**

In MQSeries Version 5.0 the Publish/Subscribe function was added. The function of Publish/Subscribe in MQSeries Integrator Version 2 is enhanced beyond that supplied in the base product and therefore there are some functional differences.

Streams are supported purely for migration purposes. No new applications should use streams as they have been replaced with structured topics. Stream names must always be included and are matched to a topic name.

Topics in the base product were arbitrary character strings, with structure only visible to the application, whereas the new function has structured topics, delimited by a forward slash and externalised both to the application and to the broker administrator.

Control, Response and Request Messages are broadly supported in the same way between the base Publish/Subscribe and the MQSeries Integrator V2 Publish/Subscribe. However, some of the options on the messages may not be supported. The ***PCF*** format, used in MQSeries base Publish/Subscribe is not supported, with only ***RFH*** format messages being supported.

The use of ***correlID*** is supported for migration purposes, as is support for ***direct requests***, and is thus not recommended for use in new applications.

There is an enhancement in the support of ***local publications*** and ***local subscriptions*** giving more flexibility for both publishers and subscribers.

The ***Register Publisher*** message, along with the ***MQPSRegOpts*** value on a Publish message, are only supported for migration purposes in the area of publishing messages on previously undefined streams or on topics with unused top level names.

None of the ***operator control commands*** from the base Publish/Subscribe function are supported in the MQSeries Integrator V2 Publish/Subscribe function. Also the ***routing exit*** in the base function will be replaced by a message flow node.

The new MQSeries Integrator Publish/Subscribe Broker will replace an existing MQSeries Publish/Subscribe Broker, with a migration command, ***'migmqbrk'*** available to migrate the state of the MQSeries Broker into the new MQSeries Integrator Broker. This is available on those platforms where the MQSeries Integrator is available. The new MQSeries Integrator Broker will be able to coexist with a network of Brokers consisting of both MQSeries Publish/Subscribe Brokers and MQSeries Integrator Publish/Subscribe Brokers, acting either as a leaf node or a new root node.

▪ Multi-Broker Domains

- **Uses of Multiple Broker Domains**

Only one Configuration Manager is required in a domain, whatever the number of brokers. There can be more than one **Control Center** to drive the Configuration Manager if required. Multiple message flows can exist in one message broker at any time. Message flows within a

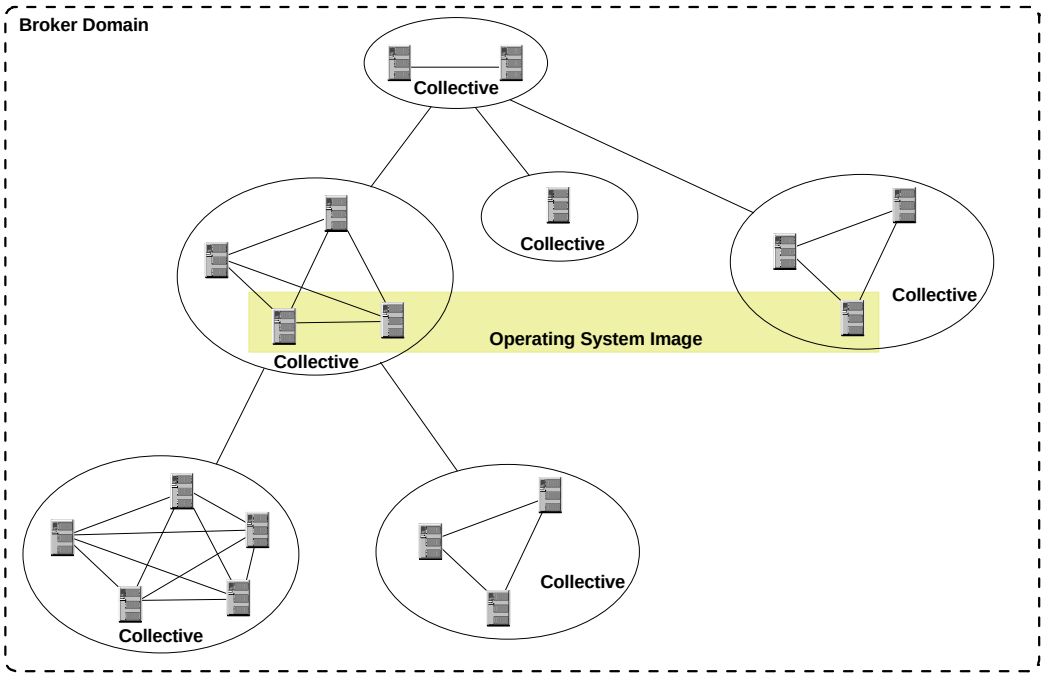
broker may be distributed into separate execution groups within the broker, or they may all exist within the same execution group. In addition, multiple brokers may be defined to give the system designer the ability to have many different message flows running on different physical machines at the same time, with each broker having a uniquely identified Queue Manager, shared with no other brokers. Users may also want to be able to have many separate instances of the same message flow to enable greater throughput of similar messages through a specified message flow definition.

- **Collectives and Publish/Subscribe**

When publishing a message, the user does not want to have to publish the message to every broker where there might be a subscriber. A key necessity for any scaleable Publish/Subscribe system is that where the same message can be subscribed to in multiple brokers, the message should be published only once in one broker, and then the brokers can forward the published message to other connected brokers where there are subscribers.

Traditionally systems could be connected together to form a tree hierarchy or a highly interconnected point to point system. A tree hierarchy can lead to long chains of systems, which can be inefficient if messages need to be passed between systems at opposite ends of the chain, with the need to be passed between many systems en-route to the destination system. With MQSeries Integrator V2, multiple brokers can be grouped together as collectives. These brokers are all connected together in a point to point manner, and the collectives are then interconnected in a tree hierarchy. This allows less individual connections than a fully point to point environment but shorter chains than a tree hierarchy.

Figure 7 - The graphic below illustrates a group of systems connected together as collectives



▪ Connecting Applications to the Broker

- **Past and Present Choices**

Some users of the MQSeries Integrator V2 Message Broker will be existing MQSeries customers who are looking to bring added value to their existing MQSeries applications. These applications will have been written using the MQSeries MQI. The messages will have a header that may have been constructed by the client programmer to pass information to the server or to the receiver of the message. Some other users will be existing users of MQSeries Integrator V1 or users of the MQSeries Base Publish/Subscribe function. These users may have messages that have the MQRFH header. This carries extra information that is used by the NEON Rules and Formatter and also by the MQSeries base Publish/Subscribe. The existing RFH header has now been extended and enhanced and this new enhanced header is the RFH2 header. The RFH2 header can be used to define publish/subscribe flows or to define the message set to which the body of the message belongs. It can also be extended by the client application to include additional information.

- **Additional Functional Interfaces**

The MQSeries Application Messaging Interface (AMI) can be used to build client applications, and the AMI will automatically build any required headers as specified using the AMI, including the new RFH2 headers. The AMI is designed to simplify the task of the application programmer, while enabling the more advanced functions and message broker facilities to be used.

The MQSeries Java™ Message Service (JMS) implementation will use the Java standard API for Enterprise messaging to build messages that communicate with other JMS services and applications written using the MQI and the AMI.

It is IBM's intent to provide the MQSeries Common Message Interface (CMI) which will provide a logical message construction API to be used in conjunction with the MQI, AMI or JMS that will construct or parse messages more easily, allowing interrogation and modification of the messages, all independent of the layout of the messages. There is comprehensive support for XML messages and the message formats can be constructed and validated by supplied tools.

Broker Internal Architecture and Administration

- **Architecture and Framework**

MQSeries Integrator V2 is designed to act as a hub within a multiple system environment allowing many systems to connect together through the Message Broker instead of connecting directly to each other. By having logical connections defined procedurally, rather than having to define each connection between pairs of systems separately, it will become easier to add systems into an established network without large amounts of additional work.

Multiple brokers can run on a machine, but each broker defined must run with its own queue manager. These can all be administered within an ***administrative domain***. The domain is the administrative scope of a set of brokers, which can exist on just one machine or on a number of different machines. All brokers within a domain share the same topology information and can communicate with each other. The information to maintain this domain administratively is held within the ***configuration manager*** of which there is one and one only in each

administrative domain. The configuration manager is accessed using a GUI where the existing system definition can be viewed and altered by the configuration manager.

All the information required to configure the Message Broker is held in a database, which is called the ***configuration repository*** by the configuration manager. The configuration manager does not directly configure the Message Brokers within the domain, but will access the ***administrative agent*** that exists in each broker. After a Message Broker is initially installed on a machine, the configuration manager will use the administrative agent to get the required configuration information from the configuration repository and deploy this definition. This definition will be stored in a cache within the broker and any updates will be queued to the agent and then used to refresh the broker's configuration.

The administrative agent, which essentially is just a first level parser of the commands from the configuration manager, is maintained by the ***Controller***, which runs in a separate process. There is only ever one controller process for each broker, and it is designed to be completely reliable, ensuring that the administrative agent and the other defined broker processes are running as required. The administrative agent will pass it requests such as to restart a ***message flow execution engine***, and the controller will execute these requests. If the controller process fails, a new controller process is started. This new process will look to tidy up previous processes by killing the orphaned processes such as the administrative agent and any message flow execution engines and will restart new instances.

The administrative agent, as directed, will also restart a message flow execution engine (also called an ***execution group***) of the Message Broker should it fail. This could happen if one thread within an execution group throws an uncaught exception, which may cause all the threads in that execution group to be shut down. The Agent could then restart the execution group. The execution group is the term for the environment that supports the execution of business message flows through a system of nodes within a broker, such that if message flows execute in separate execution groups then they are guaranteed to run in separate processes. As discussed above, the flow of a message through a set of nodes is done within a single thread. By operating multiple execution groups, you can ensure the separation of the work running within the message flows. The threading behaviour of the system, detailing how threads are allocated has already been discussed in an earlier section.

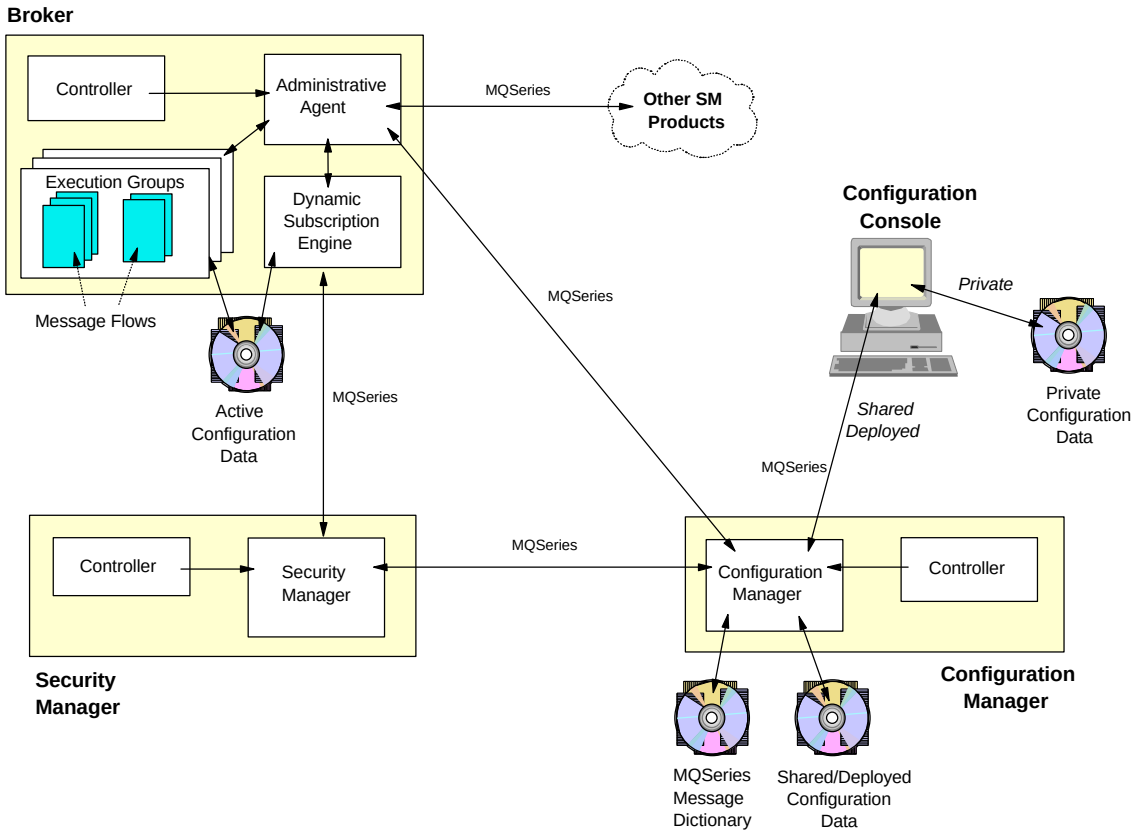
- **Maintaining State Information**

In order to provide continuity between Message Broker instances, there is a need to preserve some state information in a persistent store. A relational database is used as this persistent store. The information that must be retained is: a local cache of the broker configuration data, optimised to enable a fast restart; the operational state of the broker when last running, with a track of which message flows were enabled; a table of the durable subscriptions; and a table of the retained publications. The state information is written to the database and initially the databases supported will be DB2 ® and SQL Server. State is not maintained for messages flowing within the message flows, only for the state of the broker. Message state can be maintained by an external product such as MQSeries Workflow. If preferred, it is possible that state could be preserved by creating a node to hold state, by writing entries out to a database.

- **Broker Queues**

The agent receives the configuration messages from the configuration manager into reserved queues. Each Message Broker has a one to one mapping with a queue manager (for input messages).

Figure 8 - Here we display a simplified architecture diagram of the product components



- Tooling - The Control Center GUI (Windows NT only)

- **Concepts**

As with any product, the features are only good if they are able to be used. With MQSeries Integrator V2 designed to be used to create and deploy potentially complex and mission critical message flows within a domain that could encompass multiple machines over a wide area, the product requires a well-designed tool to provide the ability to configure and control the required functions. A Graphical User Interface is provided with the product. This allows the definition of Messages and Message Flows for use in the Broker, as well as the definition and deployment of Brokers and Execution Groups.

In fact all required functions of the Message Broker for all users can be exercised within the **Control Center**. The Control Center can be used by multiple concurrent users. Each user operates within their own workspace. All items to be worked upon by a user are **checked out** from the **Configuration Repository** and are then available for update within the user's workspace. This item is then locked to that user for update. When the user wishes to deploy the changes then the item is **checked in**. When an item is checked in, it is referred to as being shared, as opposed to being in the user's local workspace.

Different classes of users have different views within the Control Center. All operations are available to Superusers. Other classes of users are Message Flow and Message Developers, Message Flow and Message Assignors, Operations and also Security Administrators. The class of the user will give access to different tabs within the GUI for the different permitted operations.

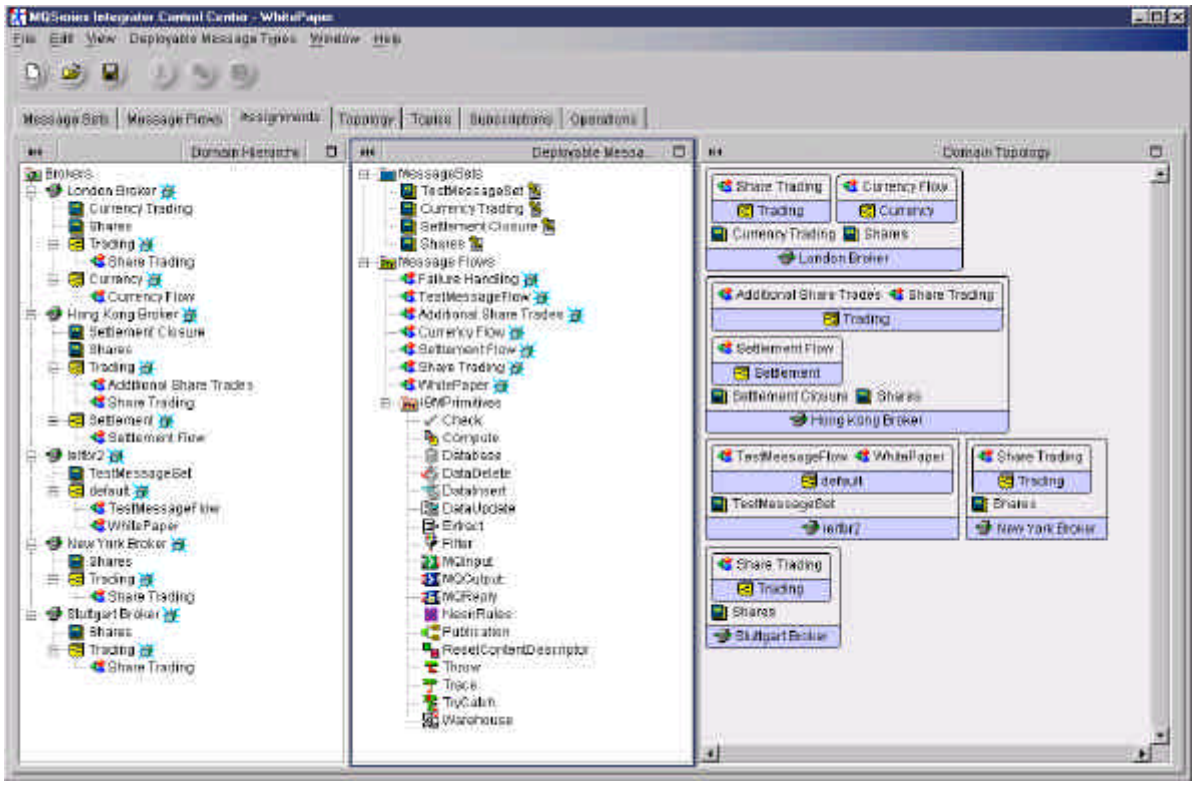
All the information held by the tool is accessed from the configuration repository. This is held in XML format and accessed using the WebDAV protocol. By using the various WebDAV commands the local and shared repositories (WebDAV Resource Servers) are updated and kept in sync.

As already mentioned previously the tools that were used in MQSeries Integrator V1 to configure the Rules and Formatter are still shipped with the product, and the use of these tools is still the appropriate way to configure the NEONRules and NEONFormatter nodes. All other functions are controlled using the new GUI.

- **Format of the Control Center**

For a Superuser, with access to all functions of the Control Center, the GUI will look as shown below:

Figure 9 - A capture of the Control Center while viewing the assignment of message flows to brokers



The Message Sets Tab allows users to define for the Message Dictionary the format of Message Sets used within the system. The messages stored within the Message Sets are also defined here, as are the **Compound Types** used and the **Fields** and **Elements** that make up the messages, along with associated **Valid Values**. The Message Tab integrates into the Message Repository manager, and will allow for full definition of the format and structure of messages. This will include the specification of the Custom Wire Format, and importing and exporting C structures or COBOL copybooks.

The Message Flows Tab displays a split screen, initially showing a palette with IBM Primitives (IBM supplied Nodes) and the workspace to deploy the primitives when defining a Message flow. To define a Message Flow, the system designer specifies a Message Processing Node Type and then drags primitives from the palette onto the right hand part of the screen and connects the nodes by wiring the terminals together. A defined Message Flow that has been identified as a Message Processing Node Type can then be used as an individual Node in subsequently defined Message Flows.

The Assignments Tab (as displayed in the picture above) allows the System Designer to create and allocate Execution Groups to Brokers. Already created message sets can be assigned to Brokers, and created Message Flows can be assigned to Execution Groups. As has been explained previously in this paper, the Message Flows run within Execution Groups of which there can be a number in a Broker. This tab allows the user to set up each broker, by means of Execution Groups, Message Sets and Message Flows, in the configuration architected by the user. Once satisfied with the selections the configuration can be deployed to the Broker, or Brokers affected.

The Topology Tab is designed to allow a broker’s definitions to be registered and be assigned a Queue Manager for deployment to machines where the appropriate installation of MQSeries Integrator V2 has been performed. In addition to creating brokers, this tab also allows for the creation of Collectives. Once a broker has been created within the Control Center, this tab can be used to assign it to a Collective, for it to be a part of a connected Publish/Subscribe Network. Subsequent to assigning a broker to a collective, it can then be connected to a broker in another collective to ensure that the network of collectives is connected together. Once assigned, the tab will display all the defined brokers and collectives. In addition, it will display which broker is in which collective, as well as a graphical representation of which broker in each collective is connected to a broker in another collective.

The Topics tab is designed to help the configuration of the Access Control of the Publish/Subscribe Service. There are two alternative views available when using this tab. One view is centered on the Topics and then displays on the right side the access for Groups and Users. Another view is centered on Users and Groups and displays on the right side the Topic Tree accessible by the Users and Groups. By using the functions of this tab the ACLs for any point of the Publish and Subscribe topic tree can be set to allow, deny or inherit the setting for users and groups to publish or subscribe to any topic or sub-topic. The ability to publish, or subscribe to Persistent messages is also set here.

The Subscriptions tab is designed to allow the administrator to display the list of Subscribers and associated subscriptions. This will then allow the administrator to delete subscriptions. This could be useful in the case of a subscriber leaving the company, and wanting to cancel the subscriptions, or in the case of an application failure, preventing any build up of unprocessed messages to an unresponsive application. This tab allows items of information about subscriptions to be tracked. These are Topics, Clients, Brokers, Subscription Points and Registration Date.

The Operations tab provides a basic Systems Management interface for managing the Broker Operations. This tab will allow an authorised user to see whether Brokers, Execution Groups and Message Flows are active or quiesced, and will be able to perform the appropriate operations on these entities. More sophisticated operations than those provided within this tab will be available using plug-in modules available from system management vendors.

▪ System Management

• Installation

There are a number of components to be installed before the system can be configured and used. This list of components will vary depending on which platform is being installed. A full list of components and hardware and software requirements will be available in the product documentation and the announcement letter.

If required, the installation tool can also be used to uninstall the product, removing both the product code base and any entries in the system registry database.

• Configuration and Set-up

Once the product has been successfully installed, it needs to be configured before use. The tasks that product configuration will need to perform are as follows:

Definition of message flows

This requires the designing of the components along with building new components by wiring together existing components.

Definition of message sets/formats

This will define the logical definition of a message format and the assignation of it to a message set.

Definition of brokers and broker topology (including execution groups)

This will set the association between the message flow and an execution group, and also the association between message sets and defined brokers.

Definition of publish/subscribe topology within a broker

This will cover all the connectivity issues between brokers and collectives to propagate publish and subscribe messages across a broker domain.

Definition of Access Control to topics and policies

Security for users and groups is handled by the underlying operating system security. Publish/Subscribe topics are defined to MQSeries Integrator V2 and principals (groups and/or users) are associated with them. ACLs for publish/subscribe and other information are then associated with principals.

Deployment of the defined configuration

Once a configuration has been defined, it should then be deployed to the specified broker to load the runtime configuration. The configuration manager process will be notified if this deployment fails.

- **Interfaces for Definition and Deployment**

All configuration changes are routed through the configuration manager. This is required to enable the configuration manager to provide information to recover/restart any set-up that needs to be recreated, even if the changes have only been applied during runtime.

Communication between user interfaces and the configuration manager, and between the configuration manager and the broker components, uses MQSeries Messaging.

The messages for the various interfaces are designed to be best suited to the needs of those using the interfaces. For example, the monitoring and reporting interfaces are defined as a set of XML messages that are published by the MQSeries Integrator V2 broker, along with an associated set of system meta-topics. These can be subscribed to by applications needing to monitor the state of the broker. As appropriate to the information it will either be retained or be available on a request/reply model.

- **Deployment of Changes**

As already stated, for all brokers in a domain, the configuration data is held by the configuration manager, which is unique in a domain. This holds two stores of data, one of which is the shared working version and one of which is the deployed, or runtime, version. The deployed version is populated with data from the shared working version by ***deploying*** the data. By working with the Control Center, the shared working data can be deployed. The Control Center can only be used by authorised local users of the machine on which it is installed.

Individual objects cannot be selectively deployed, but entire sets of objects instead must be deployed. These sets could be:

- All relevant message flows, execution groups, message sets
- All topics and associated ACLs
- All Pub/Sub topology information

or a combination of the above. However, the deployment can be selected to deploy just the changes rather than redeploy the entire configuration, thus improving the performance of the deployment. In order to deploy the data to the brokers, the format of the XML as used by the tools must be ***compiled*** into the XML format understood by the broker. Once this is complete the data is transmitted to the broker using a message to the Administration Queue for that broker, where it is handled by the Administrative Agent in the broker.

The agent deploys the data to the affected components. These update the active runtime cache and activate the changes. Messages on the success of the update will be processed through the Administrative Agent and then the Configuration manager, where the report can be further processed.

As an example, when changing a Message Dictionary that is already available in a Resource Manager, a new version of the dictionary is deployed, using WebDAV for versioning. Thus the version that might be held in a RTD can be identified by an older version number than

that held by the Resource Manager. New dictionaries and new dictionary versions are inserted into broker administration messages and published to all relevant brokers using broker administration messages. The dictionary can then be passed on to the RTD in order to replace the existing deployed version.

Security

Security Architecture and Methodology

An important part of any critical system is the ability to secure the assets by providing an effective security mechanism. The security of the system must be effective in protecting the assets from those unauthorised to use them, and it must allow access to those who have been given authorisation to access the assets. The administration of the security system must also be able to be used easily enough to ensure that the required security functionality is achieved without giving the administrator a workload that might lead to errors.

MQSeries Integrator V2 combines a product specific ***User Name Server*** to define and control lists of the users and groups authorised to the product, taken from the underlying operating system security definitions. To simplify matters, the User Name Server will only go to a single nominated machine within the domain, which need not have any brokers defined on it, and will retrieve from this machine the complete list of users and groups against which the Access Control Lists will be built.

Security Function

The key aspects of the security function for MQSeries Integrator V2 are as follows:
The ability to alter the operational characteristics of the broker is limited to authorised users and groups only. A suitable set of groups is created during the installation process to enable authorisation to be given to appropriately named groups. Then, identified users can be given membership of these groups when they are approved for access to the specified function. Also the MQSeries queues used for internal broker communication have their permissions set for the appropriate groups during the install.

Standard MQSeries security functionality restricts the ability to put messages to a queue that will be used as the input to a message flow. The same function will restrict the users' ability to get messages from the output queue of a message flow.

Access Control Lists are the mechanism by which the security manager can permit or deny the ability to either publish information to a particular topic, or subscribe to a particular topic. A related area of control is that of controlling the persistence of the information delivered for the subscription, which is also managed by ACLs.

MQSeries Integrator V2 can have a need to flow messages that contain sensitive data between the Brokers and the Configuration Manager. If this happens, the sensitive data will not flow in the clear but will be scrambled. This solution is not as sophisticated as using a specific encryption routine. If this is required, the MQSeries channel exits can be used to provide this function.

Migration from earlier versions of MQSeries Integrator

Principles of Migration

MQSeries Integrator V2 provides a framework where applications that run using the IBM MQSeries Integrator V1.1 product should run unchanged when using IBM MQSeries Integrator V2.

As previously described, the NEON Formatter and Rules functions from the original product exist in the new product as nodes in the message flow. To use existing applications, some configuration of the broker is required but no alteration of the existing applications is needed, unless specified below. The migration of the function from the V1.1 product is provided by the replication of the earlier functionality in an encapsulated form (as the nodes) so that the existing processing will be reproduced in the existing product.

The tools for the existing MQSeries Integrator V1.1 product, to define the NEONRules and NEONFormats are not merged into the new tools available with MQSeries Integrator V2 but are kept separate to allow existing skills to be used and to prevent the difference between the product versions being reflected with a confusing interface.

Migration Details

Certain configurations of the MQSeries Integrator V1.1 product may lead to problems when migrating to the MQSeries Integrator V2 release. These problems may not adversely affect the operation of the product and the migrated configuration but the scenario may produce uncertain results.

Customers may have programmed user exits in their existing MQSeries Integrator V1.1 configurations. The implementation of the MQSeries Integrator V1.1 function is replicated in MQSeries Integrator V2 and the existing user exits written for the MQSeries Integrator V1.1 function will continue to operate. Again, this should be tested by the customer when migrating to the new environment. User exits written for MQSeries Integrator v1.0 were statically linked rather than dynamically linked and will not work in the new environment.

A key difference in the internals of the MQSeries Integrator V2 release to the internals of the MQSeries Integrator V1 release is that the NEON functionality is single threaded and not thread safe, whereas the MQSeries Integrator V2 release is fully multithreaded. This can lead to problems when putting work through the NEON Rules and Formatter Nodes, as they will become thread-bound, slowing the workload through the broker and providing additional threads to those nodes will not alleviate the situation. If a NEON node is part of a critical path for a longer message flow, the lack of multithreading ability will create difficulties in driving the expected throughput of work.

Summary

MQSeries Integrator V2 is a strategic open framework that builds on, and extends the function from the earlier version, building on its strengths and adding value across the Business Integration space, making it easier for customers to connect varying applications together in a simple framework, which is both open and extendible. This will provide opportunities for vendors to enhance the product, further increasing the value of the product to customers.

In providing this open framework, IBM is enabling customers to create infrastructures in a relatively short amount of time. This allows them to take advantage of new opportunities in the expanding marketplace, as well as increasing efficiency when connecting their existing applications. As additional nodes become available in the future, the message flows built using this product will allow increasingly sophisticated solutions to be deployed quickly and easily.
In conjunction with the other enhancements to the MQSeries family, this release of MQSeries Integrator meets the needs of Enterprise Application Integration going into the 21st century.

- **For further information**

More information on MQSeries or other IBM products can be found by contacting your IBM marketing representative or see the MQSeries Internet homepage:
http://www.ibm.com/software/ts/mqseries/
or the main IBM Internet homepage:
http://www.ibm.com