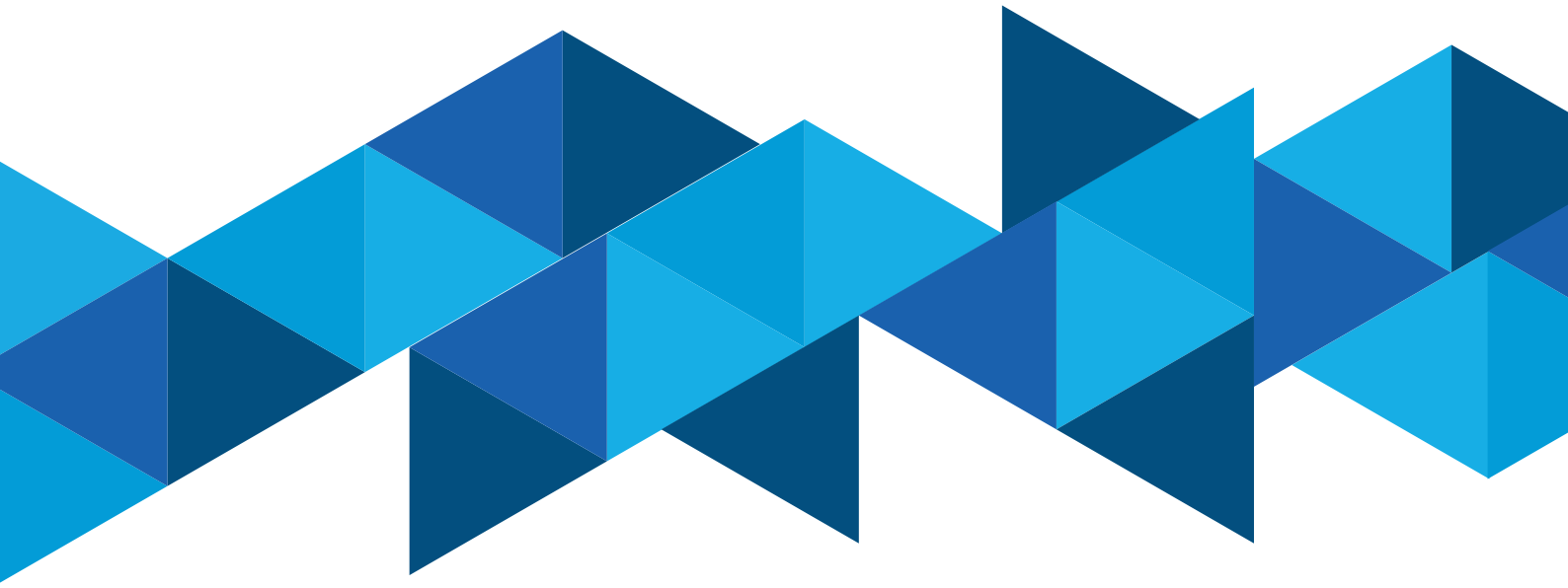
MIRACLE
SOFTWARE SYSTEMS



Best Practices for IBM Integration Bus
for National Retailer

SOA and Connectivity

Bhaskar Gara (General Manager - Integration)
Miracle Software Systems, Inc.

Introduction

This document presents some findings and recommendations concerning the use of **IBM Integration Bus.** These findings were developed from previous experiences of various integration projects. It covers the aspects of IIB development encompassing Message Set guidelines, Message Flow guidelines and Deployment guidelines.

- The first key to successful Technical Design and Interface Development is a firm knowledge of the intended business logic and structure

- The source to the Business Requirement is to acquire a firm and defined Functional Specification.To some degree, the Functional Specification may undergo a few minor amendments while the task of Technical Specification is underway. However, it is expected that the Functional Specification has defined all the messages, components and processes that are required for the interface before the commencement of Technical Design
- There may be (and usually is) multiple components to each interface, and in IIB terms, that boils down to Adapters and Message Flows
- The Functional Specification is used to define where the Business Requirements are met in terms of various components. It is required to consider the various options available and create a design that is based on workable functionality of IIB, and to hopefully end up with a reasonable level of maintainability and performance as well

Interface Design Layout

- The IIB interfaces to be implemented should have certain common characteristics so that re-use of components is optimized
- The design of the message flows should adhere to one or more of the pattern templates laid out for the project

Common Characteristics of Interfaces

- Each IIB Interface designed for the project should have certain characteristics, which are assumed to be common across all implemented interfaces
- These common characteristics generally refer to some common routines and flows or sub-flows, which are recommended to be used globally across all interfaces in that project

Common Error/Exception Handling

- Each IIB interface shall have a mechanism to handle errors/exceptions.These errors/exceptions refer to both Systems generat-

ed ones as well as User Generated ones

- It is recommended to have a common Error Handling Procedure/ Module to handle all errors/exceptions that arise during execution of an interface
- This Procedure/Module shall be extensively re-used by all interfaces of the project

Common Control Handling

- The flow of each message through a IIB interface should be 'Control Handled' for audit and logging purpose. For this reason,there is a requirement to have a 'Control Mechanism' in each and every message flow
- It is recommended to have a 'Common Control Handling' routine/component that can be re-used extensively by all interfaces of the project

Common Routines and Sub-Flows

Many message flows require to perform certain functions like re-formatting a date, getting current date, getting a unique identifier, etc which can be laid down as common routines for re-use by the interfaces. Also there may be certain processes like adding an RFH2 header to a message,etc which can also be laid down as common sub-flows for re-use by interfaces.

It is recommended to have a set of common routines and sub-flows for the project that can be extensively re-used by the interfaces. The contents of this set cannot be defined exhaustively at any point of time. This set shall continually be gaining members from time to time in order to meet requirements.

Location of Integration Logic

- An interface is comprised of Adapters and Message Flows
- The integration logic is therefore contained wholly within the adapters and message flows

- The interface designer has the option of building any required integration logic in the adapters or message flows
- It is recommended that as much business logic as possible will be performed in message flows rather than in the adapters. In essence logic is only to be located in adapters if that is the only option for the interface
- In other words, adapters will me made as simple as possible. The core of the business logic should be kept in WMB.

The benefits of this approach are:

- The interface solutions are pattern based, with the patterns represented by WMB message flows. Being pattern based, enhances reusability.
- Scaling and resilience is simplified with fewer complex components to consider.
- Maintenance requirements are easier to define and execute, by largely avoiding the need to change one component to accommodate the change in another requirement

Understand the Input and Output Messages

It should be possible to examine the Functional Specification and determine what the input is, and what the output would be. From this standpoint, the WMB message flow can be regarded as the instrument to convert the input message into output message(s).

Determine how many Message Flows are needed

- First, at an application system level, it must be determined how many flows will be required for an interface. Out of this determination, it will be possible to determine how many types of messages a flow must handle
- Sometimes there is just one input message format and one output message format. Other times there are multiple messages for a given flow and there must be logic within the flow to sort them out.

Complete the processing with as few nodes as possible

- The most fundamental recommendation to consider when writing a message flow is to be as concise as possible. There is a performance cost associated with passing through each of the built-in nodes of IIB, so it is best to write message flows in as few nodes as possible
- In other words, there shouldbe as few as possible nodes between the input and output formats for a given business requirement

Message Routing within a flow

- Routing within the flow should be minimized where possible. When routing is required, then the rule about minimizing the number of nodes still applies
- For a number of reasons, some message flows process more than one message type. In this case it is important to understand the distribution of these different message types and their importance so that the most critical or most frequent messages go through the cheapest path; i.e. the one with fewest and simplest nodes
- When more than one message is processed in a flow, or when there is a requirement for dynamic logic inside a flow based on message content,there are at least two ways to direct the processing logic inside the flow
- One way is through the use of RouteToLabel node. This node makes use of the Destination List array of the LocalEnvironment Tree of a message flow. When the message arrives, the DestinationList is populated according to the business requirement, and the RouteToLabel and Label nodes are used to direct subsequent processing

Sub-Flows

- Message Flows can be divided into SubFlows.Essentially, a Sub flow is a sequence of nodes that begins with an Input node and ends in an Output node. The object of using SubFlows is that functions can be logically segregated in a way that increases maintainability and offers opportunity for reuse

Transactionality and Persistence

- As a rule, all messages should be handled as a Transaction
- In the Input, Output and Compute nodes, the Transactionality should always be set to 'Automatic'. This practice allows the WebSphereMQ messagessage attributes to control the Transactionality of the flow; i.e. if message is read from a queue with an attribute of 'Persistent', IIB will supply the appropriate logging to guarantee the Transactionality of the message
- It must be considered that overhead is incurred by making a message transactional.This is caused by the need to save all the data necessary to enable a roll back should a failure condition occur in a message flow

Use of CARDINALITY

The use of the CARDINALITY function should be restricted to minimum as it requires the parser to process a large portion of the message and thereby hindering performance.This built-in function should not be used inside a loop unless it is inevitable.

ESQL REFERENCE Variable

- Reference variables can be used in ESQL to store a part or an entire message tree location and can be thought of as being similar to a C/C++ pointer to a message element. The use of REFERENCE variable is recommended in ESQL for two reasons:
- They reduce coding considerably, especially, when large message tree with deep hierarchy has been defined.
- They make ESQL more efficient when used appropriately
- Reference variables can be used to store locations in any type of message tree (InputRoot, OutputRoot, Local Environment, Environment, etc)

Backout Processing Considerations

- When an exception is thrown within a message flow and is not caught by inclusion of a TryCatch Node, The Input Node for the message flow catches it. If the catch terminal of theMQInput Node is connected, the message is propagated to this terminal and is processed according to the message flow logic
- If the Catch terminal of the MQInput Node is not connected, the transaction is rolled back. If the message was read under a syncpoint the original remains in the queue but WebSphere MQ increments the MQMD Backout count
- The MQInput Node then reads the message once again
- The MQMD Backout count is examined before the message is processed again. If it is not zero then it implies that the message received by the message flow is a backed out message and the broker then performs backout processing
- If the MQMD backout count is less than the Backout Requeue Threshold attribute specified in the queue definition,the message is propagated through the output terminal of the MQInput Node for normal processing once again
- If the MQMD backout count is not less than the Backout Requeue Threshold attribute specified in the queue definition,the message is propagated through the Failure terminal of the MQInput Noden
- If the Failure terminal is wired then it follows that path but, if it is not wired the Backout Requeue Name attribute is looked for and if a queue name found in this attribute, the message is put into that queue
- If no name is specified in the Backout Requeue Name attribute, the message is written to the default Dead Letter Queue defined for the queue manager

- If the message could not be written to the Dead Letter Queue then it remains in the Input Queue

IIB Message Flow Standard Unit Test Conditions

- The following are the standard Unit Test Conditions that should be considered by a developer while performing Unit Test on an interface:
- When a valid Test Data is passed through the message flow one or more output message(s) are created on the output queue(s)
- When an invalid Test Data is passed through the message flow the message is propagated to the error handling queue for error processing
- All fields of the output message are of correct length and have been correctly formatted according to the requirements
- Where the message has repeating fields or structure, the interface works correctly for both single and multiple instances of the field or structure
- Where the message has an optional field or structure, the interface works correctly for both with and without the field or structure in the message

Important Structures

MQMD – Message Descriptor

- The MQMD structure contains the control information that accompanies the application data when a message travels from one queue to another The following table summarizes the fields in the structure:

Field	Description
StrucId	Structure Identifier
Version	Structure Version Number
Report	Option for report messages
MsgType	Message Type
Expiry	Message Lifetime

Field	Description
Feedback	Feedback or Reason Code
Encoding	Numeric Encoding of message Data
CodedCharSetId	Character set identifier of message data
Format	Format name of message data
Priority	Priority of message data
Persistence	Message Persistence
MsgId	Message Identifier
CorrelId	Correlation Identifier
BackoutCount	Backout Counter
ReplyToQ	Name of Reply Queue
ReplyToQMgr	Name of Reply Queue Manager
User Identifier	User Identifier
AccountingTo Ken	Accounting Token
ApplIdentity Data	Application data relating to identity
PutApplType	Type of application that put the message
PutApplName	Name of application that put the message
PutDate	Date when message was put

Put Time	Time when message was put
ApplOriginData	Application Data relating to origin
GroupId	Group Identifier

Field	Description
MsgSeqNumber	Sequence number of logical message within group
Offset	Offset of data in physical message from the start of logical message
MsgFlags	message MsgFlags
OriginalLength	Length of original message

MQRFH2 – Rules and Formatting Header

The MQRFH2 structure defines the layout of the rules and formatting header. This header can be used to send string data that has been encoded using an XML-like syntax.It allows Unicode string to be transported without translations, and it can carry numeric data-types.

The following table summarizes the fields in the structure:

Field	Description
StrucId	Structure Identifier
Version	Structure Version Number
StrucLength	The total length of MQRFH2
Encoding	Numeric encoding of data
CodedCharSetId	Character set identifier of data
Format	Format Name of data
Flags	Flags

NameValueCCSID	Character set identifier of Name Value data
NameValueLength	Structure Version Number
NameValueData	Option for report messages

Properties – Message Flow

The Properties tree is the first element of the message tree and holds information about the characteristics of the message.

Field	Description
MessageSet	The Message Set Identifier
MessageType	The Message Name
Encoding	Numeric encoding of data
CodedCharSetId	Character set identifier of data
Transactional	Boolean flag to indicate Transactionality
Persistence	Boolean flag to indicate Persistency
CreationTime	The put time of the message
ExpirationTime	The expiration time of the message
Priority	The priority of the message
ReplyIdentifier	The reply identifier
ReplyProtocol	The reply protocol

Debug Tracing

In the event of an error in a message flow, a debug level trace should be set for the flow. This is done with the help of a set of commands to generate the debug trace file for diagnostics

- Open a command prompt window
- Execute the command mqsichangetrace <Broker Name> -u-e < Execution Group Name > -l debug -f < Message Flow Name > -r

- Execute the message flow with the message that you want to trace
- Execute the command mqsireadlog <Broker Name> -u -e <Execution Group Name> -l debug -f <Message Flow Name> -o <Temporary File Name>
- Execute the command mqsiformatlog –i <Temporary File Name> -o <Trace File Name>
- Open the generated Trace files in a text editor and perform diagnosis