49

MQ update

July 2003

In this issue

MQ Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38126
From USA: 01144 1635 38126
Fax: 01635 38345
E-mail: info@xephon.com

North American office

Xephon/QNA
Post Office Box 350100
Westminster CO 80035-0100, USA
Telephone: (303) 410 9344
Fax: (303) 438 0290

Disclaimer

Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, scripts, and other contents of this journal before making any use of it.

Editor

Madeleine Hudson
E-mail: MadeleineH@xephon.com

Subscriptions and back-issues

A year's subscription to *MQ Update*, comprising twelve monthly issues, costs £255.00 in the UK; $380.00 in the USA and Canada; £261.00 in Europe; £267.00 in Australasia and Japan; and £265.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the July 1999 issue, are available separately to subscribers for £22.50 ($33.75) each including postage.

Contributions

When Xephon is given copyright, articles published in *MQ Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. For more information about contributing an article you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

MQ Update on-line

Code from *MQ Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at www.xephon.com/mq; you will need to supply a word from the printed issue.

Printed in England.

WebSphere MQ and the IMS Bridge

The WebSphere MQ (WMQ) IMS Bridge is an IMS Open Transaction Manager Access (OTMA) client. It is the component of WMQ for z/OS that enables implicit MQI support and facilitates access for WMQ applications to applications running under IMS systems. This allows WMQ messages to control host applications without having to rewrite, recompile, or relink them.

WHAT IS OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS V5.1 or later. It functions as an interface for host-based communications servers accessing IMS applications through the z/OS Cross Systems Coupling Facility (XCF).

OTMA is implemented in a z/OS sysplex environment. It allows clients to connect to IMS in a high-performance manner, enabling the client to support interactions with IMS for a large network or a large number of sessions.

A queue manager can connect to one or more IMS systems and more than one queue manager can connect to one IMS system. The only restrictions are that they must all belong to the same XCF group and they must all be in the same sysplex.

THE IMS BRIDGE

Messages are put by applications on a WMQ queue as usual to submit to an IMS transaction that uses the bridge. WMQ puts messages to an IMS queue but it is first queued in WMQ to enable the use of syncpoints (so that data integrity can be assured). The messages that contain IMS transaction data can have an IMS header (the MQIIH structure) or the WMQ IMS Bridge is enabled to make assumptions about the data in them. They also have the MQ Message Descriptor structure (MQMD).

The storage class of the WMQ queue determines whether or not the queue is an OTMA queue (ie a queue used to transmit messages to the WMQ IMS Bridge) and the particular IMS partner to which the message data is sent. Remote queue managers can also start IMS transactions by writing to these OTMA queues on WMQ for z/OS. Data returned from the IMS system is written directly to the WebSphere replyto queue specified in the MQ Message Descriptor structure (MQMD). This may be an XMIT queue to the queue manager specified in the ReplyToQMgr field of the MQMD. Figure 1 illustrates a typical WMQ and IMS Bridge application in z/OS.

In bridge applications there are no WMQ calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. WMQ applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they do when driven by non-programmable terminals.

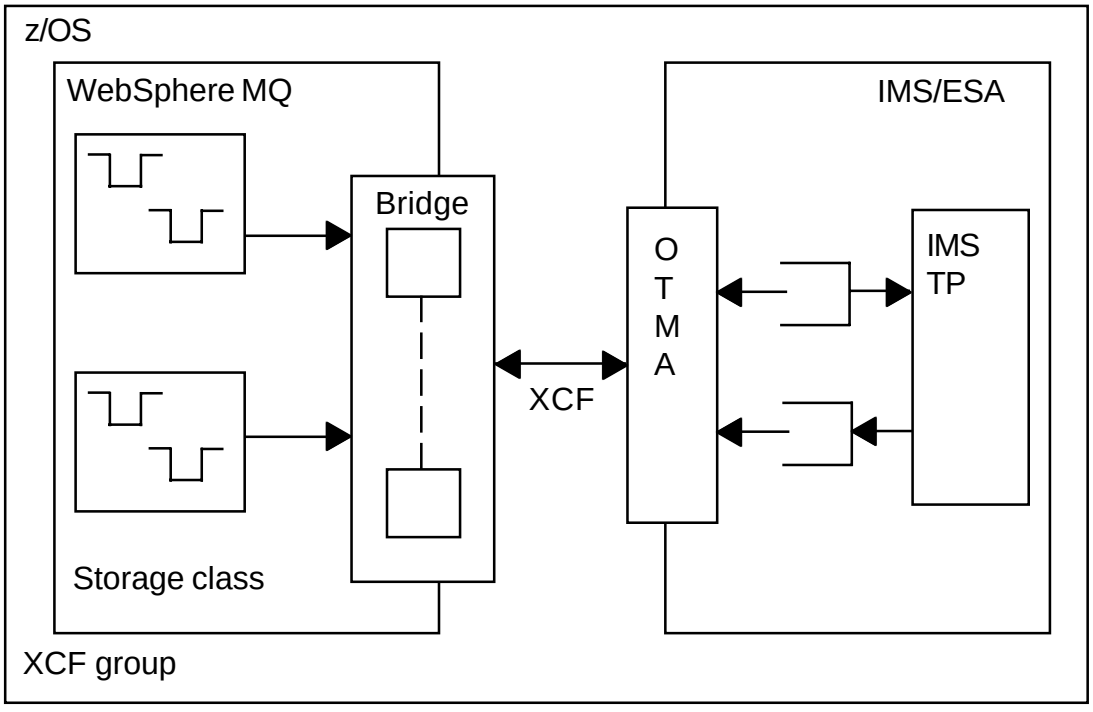

Figure 1: WMQ and the IMS Bridge in z/OS

If you are using an IMS application that processes multi-segment messages you should ensure that all segments are contained within one WMQ message.

(Note that, when IMS on the z/OS has all messages passed via the OTMA as persistent and the queues used to send messages from a client to an IMS transaction are all defined as persistent by default, this allows the application to use queue defaults when putting messages, resulting in persistent messages.)

IMS BRIDGE SET UP

To use the WMQ IMS Bridge you should set up the WMQ queue manager and the IMS system it will send messages to on the same z/OS image. Both IMS and WMQ must also belong to the same XCF group (xcfgname).

IMS CHANGES

For IMS this is achieved by adding the parameters given in Table

OTMA=Y	Optional. If coded, OTMA is started automatically at IMS start up. If *N* is used OTMA can be started by a **/START OTMA IMS** command.
GRNAME=groupname	The XCF groupname (xcfgname) to be used by IMS and WMQ. This value matches the OTMACON keyword value of the CSQ6SYSP macro in WMQ.
USERVAR=membername	The XCF membername (xcfmname) for this member of the group. This matches a value assigned to a storage class in WMQ. If there is no value specified the value of APPLID is used.

Table 1: Additional IMS parameters

1 to the IMS parameter list or in member DFSPB*xxx* in the IMS PROCLIB.

WMQ CHANGES

In WMQ various options need to be set up to enable IMS Bridge communication.

- Put WMQ and IMS in the same XCF group by updating the OTMACON parameters in CSQ6SYSP macro to match the GRNAME value in the IMS parameters.
- Define the storage classes to be used by the queue manager's local queues that pass messages to IMS. A storage class will be required for each of the IMS regions that the queue manager will pass messages to. When defining the storage class you should identify the associated IMS region by its XCF group name and member name.
- Define the local queue(s) that will be used to hold messages destined for the IMS region(s) by associating it with the appropriate storage class.

SECURITY IMPLEMENTATION

The queue manager and the corresponding IMS system accessed must be assigned an xcfmname in the same xcfgname. Then two RACF profiles must be defined in the FACILITY class for both the queue manager member name and the IMS member name, in the following format:

- IMSXCF.xcfgname.mqxcfmname
  - eg IMSXCF *ITGRP.MQPA*
- IMSXCF.xcfgname.imsxcfmname
  - eg IMSXCF.*ITGRP.IT2WA*

(ITGRP is the same XCF group name, MQPA is the queue manager XCF member name, and IT2WA is the IMS member name. Each IMS system will require a separate profile.)

The queue manager connects to IMS via its address space user-ID, normally the user-ID defined in the started-task group. This started-task (STC) user-ID must have access to the OTMA group name and this is done by granting, for example, QMGRSTC user-ID, READ access to the first RACF profile (no security check is done if /SECURE OTMA is set to NONE in IMS).

The second profile determines the level of access this user-ID (for example QMGRSTC) is allowed when the IMS Bridge connects to IMS and indicates the level of security that is required on subsequent transactions. The options are listed below.

- *None*. RACF authentication is done for each of the messages WMQ receives that is destined for the IMS Bridge. A check is made to verify the user-ID specified in the UserIdentifier field of the MQMD header and the password or PassTicket in the Authenticator field of the MQIIH structure, and to ensure that they are a valid combination. A Utoken is created with a password or PassTicket and is passed to IMS but the Utoken is not cached.

  If switch profile NO.SUBSYS.SECURITY exists in the MQADMIN class, this level of security overrides its definition.

- *Read*. The same checking is done as above when a specific user-ID is encountered for the first time, or when the user-ID has been encountered before but the cached Utoken was not created with a password or PassTicket. WMQ requests a Utoken if required and passes it to IMS.

- *Update*. As each message is received on the local queue the value in the UserIdentifier field of the MQMD header is passed to RACF to determine whether or not it has a profile. The value in the UserIdentifier will need to have RACF access to relevant IMS transactions. A Utoken is built and passed to IMS and the Utoken is cached.

- *Control/alter*. These options indicate that no security Utokens are required for any user-IDs for this IMS system. (These options would probably be used for development and test systems only.)

Note that, to amend the security level, access to the security profile must be changed and then the bridge stopped and restarted (for example, using **/STOP OTMA** and **/START OTMA** commands). The access is defined when WMQ connects to IMS and lasts for the duration of the connection.

If you change the authorities in the FACILITY class you must issue the RACF command **SETROPTS RACLIST(FACILITY) REFRESH** to activate the changes. You can use a password or a PassTicket but you must remember that the IMS Bridge does not encrypt data. Refer for more explanation to the section on using RACF PassTickets.

Some of the above setup is affected by the security settings in IMS through use of the **/SECURE OTMA** command.

Cached Utoken information is held for the duration defined by the INTERVAL and TIMEOUT parameters of the **WMQ ALTER SECURITY** command.

A WMQ message that passes through the IMS Bridge contains security information by way of a user-ID held in the UserIdentifier field of the MQMD structure, the security scope contained in the SecurityScope field of the MQIIH structure (if the MQIIH structure is present), and a Utoken (unless the *IMSXCF.xcfgname.imsxcfmname* profile has CONTROL or ALTER values set up).

Security is checked by the **/SECURE OTMA** command in IMS. To enable user-ID authentication the CHECK option must be the minimal level selected. The options are:

- *None*. No transaction authentication is performed.
- *Check*. The user-ID in the UserIdentifier field is sent to IMS to authenticate transaction access and command authority. An Accessor Environment Element (ACEE) is built in the IMS control region.
- *Full*. The user-ID in the UserIdentifier field is sent to IMS to authenticate transaction access and command authority. An

ACEE is built in the IMS dependent region as well as in the IMS control region.

- *Profile*. The user-ID in the UserIdentifier field is sent to IMS to authenticate transaction access and command authority. The SecurityScope field in the MQIIH structure is used to determine whether an ACEE is built in the IMS dependent region as well as in the control region.

USER-ID AND SECURITY

These are two of the methods by which the user-ID and, if necessary, the password may be forwarded to IMS on z/OS.

- The first option involves coding the z/OS user-ID and password into the application or deriving it at runtime from a configuration file, for example. In this case the user-ID is sent in the MQMD UserIdentifier field and the password in the MQIIH IMS header, which is positioned immediately before the data in the buffer.

  However, having plain text information on the network presents a security exposure and is best avoided.

  (Note: check whether Windows 16-bit and 32-bit WMQ applications support the MQIIH IMS header.)

- The other option is to configure the channel by defining a unique (non-expiry) user-ID and set The MCA User-ID field to the appropriate user, for example, APPLTS1 on the test system and APPLPS1 on the production system. If the profiles are then set to match the security required by IMS, data integrity is assured.

  There is no need for the front-end application to know anything about the z/OS security information. All validation is performed by the z/OS and the MCA user-ID that is sent in the message descriptor UserIdentifier field. This avoids the need for the MQIIH IMS header and also the password to be coded in the application.

(Note: if the IMS SECOPT security parameter is NONE no user information will be forwarded to IMS even if it is fully specified by WMQ.)

USING RACF PASSTICKETS

You can use a PassTicket instead of a password in the IMS header (MQIIH). A PassTicket is created from a user-ID, the target application name (APPL), and a secret key.

- The APPL field consists of an application name, which should be of the form MVS*xxxx*, where *xxxx* is the SMFID of the z/OS system on which the target queue manager runs.
- The secret key is an 8-byte value containing uppercase alphabetic and numeric characters; it can be used only once and is valid for a 20 minute period.
- The PassTicket information in IMS headers is passed to RACF by WMQ.

Setting up RACF PassTickets

To start PassTickets validation you must:

- Activate the PTKTDATA class and issue the **SETROPTS CLASSACT(PTKTDATA)** command.

  The PTKTDATA class is where all profiles that hold PassTicket information are defined.
- Define a secure sign-on application key for each application in the PTKTDATA class profile.
- Issue the **SETROPTS RACLIST(PTKTDATA)** command.

(Note: you must issue the **SETROPTS RACLIST (PTKTDATA) REFRESH** command if any changes are applied to profiles in the PKTDATA class.)

The secured sign-on function

This function enables workstations and client machines to

communicate with a host system without using a RACF password. It is an alternative that removes the need for the password to be sent across the network in clear text. The user authentication of a mainframe application user-ID is moved from RACF to another authorized function executing on the host system or on a remote environment.

The PassTicket is a one-time-only password that is generated by a requesting product or function. It gives only one user access to a specific application for approximately ten minutes. For most applications, once a particular PassTicket is used the same user cannot use it again for the same application during the same ten-minute interval.

By keeping a copy of all used valid PassTickets for the duration of the ten-minute interval, during which they might possibly be used again, RACF provides a level of protection against reuse.

Use the **RDEFINE** command to define the profile as follows:

- RDEFINE PTKTDATA profile-name.
- SSIGNON(key-description).
- UACC(access-authority):
  - PTKTDATA – specifies the PassTicket class.
  - profile-name – is the name of the profile; each application must have a unique profile, so you cannot have a generic profile in the PTKTDATA class. If you define one it is ignored during PassTicket processing for the application.
  - key-description – defines the secured sign-on application key and specifies the method RACF is to use to protect it in the RACF database on the host. Encryption or masking methods can be specified. Secured sign-on keys are 64-bit Data Encryption Standard (DES) keys.
  - access-authority – is the universal access authority associated with the resource protected by this profile. The UACC is defaulted to NONE for the PTKTDATA class. After a profile has been created you can change it with the **RALTER** command as follows:

- RALTER PTKTDATA profile-name
- SSIGNON(key-description)
- UACC(access-authority).

Profile names

A PTKTDATA class profile name can consist of:

- Application name only.
- Application name appended by a RACF connect group name.
- Application name appended by a RACF user-ID.
- Application name appended by both a RACF connect group name and a RACF user-ID.

For example, if the application name is MQIMS1, the connect group is SYSA, and the user-ID is USERIMS1, the following are the profile names as per the list above:

- MQIMS1.
- MQIMS1.SYSA.
- MQIMS1.USERIMS1.
- MQIMS1.SYSA.USERIMS1.

Depending on the application, for example, IMS, the secured sign-on function uses a specific method for determining profile names in the PTKTDATA class.

(Note: check whether your installation is using RACF exit ICHRIX01 to modify the application name used for user-verification processing. If so, the application name used to determine the PTKTDATA class profile name must match the application name ICHRIX01 selects.)

To define a profile for an IMS application for example, define the profile to the PTKTDATA class with a left-most qualifier that matches the standard naming conventions you use to define these applications to the APPL class.

Saida Davies
IBM (UK)

Quality-checking exported message flows

PROBLEM: QA WALKTHROUGHS ON MESSAGE FLOWS

When moving WMQI message flows from a test environment to production it is wise first to inspect the flow for any errors that might create production problems. This is especially important because flows that ran perfectly in test may require changes in order to work in production. These changes may include ODBC DSNs, queue manager names, environmentally sensitive data, etc.

However, the point and click GUI that WMQI Control Centre uses can make it difficult to see the big picture and view the details in their proper context, since it hides the details you need to see behind the icons and multiple property pages. Add to this the fact that you should also verify that the necessary queues are set up in production and that they have the required properties. Compare this with traditional programming, where everything is packaged together in a source listing.

SOLUTION

Let's consider a basic walkthrough of a message flow. At the very least you would want to verify that all input and output queues exist on the production WMQI server and that they all have valid properties. For instance, to control looping you should verify that the input queue has a backout threshold greater than zero and that a backout queue is defined. If the output queues were remote you would want to verify that they pointed to valid queues on the remote queue managers. This requires a lot of manual work but you can automate the process without too much effort.

THEORY

Message flows are exported into XML files and you can analyse the XML with the Document Object Model (DOM) API. There are several DOM implementations available, depending on your

platform. If you use Java the most popular DOM parser is Xerces, which is available as a free download from Apache's Web site (*http://www.apache.org*). For Visual Basic, Active Server Pages, or anything else that uses a COM interface, you can use Microsoft's MSXML parser, which should already be installed on most Windows workstations (*http://msdn.microsoft.com/xml*). Finally, if you program with any .Net languages, XML DOM support is built into that framework.

To extract data from XML the DOM uses a filter format called XPath. Once you understand the structure of the message flow XML you can use XPath filters to extract information about the various nodes and properties within a message flow, and that lays our foundation for automated quality checking.

Let's assume for this discussion that we just want to list the queue names of the flow's input and output nodes and verify that those queues all exist and have valid properties. I will show you the steps necessary to accomplish this.

APPLICATION

The structure, or schema, of a message flow export file is defined in a text file called a Document Type Definition (DTD). In order to work with export XML you must have a copy of file *mqsi.dtd* in the directory as the export file. You can find this file in the *\Bin* directory for WMQI.

XML STRUCTURE

The MessageProcessingNode element defines each node in a message flow. It has an attribute called xmi.label, which defines the name of the node. To get the node type, see the title attribute of the MessageProcessingNodeTypeRef element, which is a child of the first element. There is a child element AttributeGroup for each property page for a node (ie Basic, Advanced, Request) and this element has a child element named Attribute for each non-default property you set for a node. Attribute xmi.label contains the property name, and attribute value contains the property value. If you take a good look at some export XML you should get the idea.

USEFUL XPATH EXPRESSIONS

The XPath filters can specify one particular XML element or attribute or a set of similar elements or attributes, based on how specific they are. There follow a few examples of XPath expressions. (These XPath expressions work for message flows exported from WMQI V2.1. Later product releases may alter the export structure. Each of the XPath examples should be a single line of code – they are broken here for clarity.)

- XPath expression that gets all MQInput node names:

```
//MessageProcessingNode
[MessageProcessingNodeTypeRef@title='MQInput']
/@xmi.label
```

- XPath expression that gets all MQInput queue names:

```
//MessageProcessingNode
[MessageProcessingNodeTypeRef/@title='MQInput']
//Attribute[@xmi.label='queueName']/@value
```

- XPath expression to get the MQInput node message domain:

```
//MessageProcessingNode
[MessageProcessingNodeTypeRef/@title='MQInput']
//Attribute[@xmi.label='messageDomainProperty']
/@value
```

The following expressions will extract information on the MQOutput nodes.

- XPath expression to get all MQOutput node names.

```
//MessageProcessingNode
[MessageProcessingNodeTypeRef/@title='MQOutput']
/@xmi.label
```

To get the MQOutput queue names:

- XPath expression to get all MQOutput queue names.

```
//MessageProcessingNode
[MessageProcessingNodeTypeRef/@title='MQOutput']
//Attribute[@xmi.label='queueName']/@value
```

XPATH SYNTAX

By way of a brief explanation of XPath syntax: the expression is

a list of element names joined by slashes and filters. A single slash (/) between two elements represents a direct parent/child relationship. A double slash (//) at the start of the expression or between elements represents an ancestor//descendent relationship with any number of levels in between. Thus the expression //ElementName means 'search the entire document until you find ElementName'. To specify attributes in XPath prefix them with '@'. Filter expressions are placed between square brackets '[]'.

Therefore, our first XPath example is requesting the contents of every xmi.label attribute belonging to MessageProcessingNode elements, where the child element MessageProcessingNodeTypeRef has a title attribute of 'MQInput'.

XPATH AND THE DOM

You use these XPath expressions with the XML DOM through a pair of methods: SelectNodes() and SelectSingleNode(). The first one will return a list of XML nodes that match your XPath expression while the second returns only one specific node.

ADDITIONAL INFORMATION FROM MQSERIES

Now that you have a list of input and output queues you should be able to open each queue with MQOO_INQUIRY for the rest of the information you need, namely:

- Verify the queue exists.
- Does every input queue have a valid backout count and backout queue?
- For remote queues, what queue and queue manager do they point to, and do they exist?

PROGRAM OUTLINE

Put all this together into a program that produces a single report summarizing your message flow and flagging all errors.

1 Get the location of the export file and load it into the XML parser.

2 Using the first two XPath examples build a list of MQInput node names and their associated queues.

3 Use the third XPath sample to get the message domain. If it is not valid for your company's standards flag it as an error (ie Blob in an XML shop).

4 Use the last two XPath samples to build a list of MQOutput node and queue names.

5 For each MQInput node, open the queue in MQOO_INQUIRE mode. Get the backout requeue count and backout queue name properties. If the count is zero, write an error message. If the backout queue name is blank or the queue does not exist, write an error message. If the backout queue type is remote, try to verify the existence of the destination queue on the remote queue manager. Otherwise, write the node type, node name, and queue name to your analysis report.

6 For each MQOutput node, open the output queue as MQOO_INQUIRE. If the queue does not exist, write an error message. If the queue type is remote, try to access the remote queue manager and verify that the destination queue exists. Write the MQOutput node name and queue name to the analysis report.

CONCLUSION

You may be interested in other types of information from your extracted message flows and you should be able to get it by extending the examples shown above. Some things you might want to add to this process include: validating ODBC DSNs and table names; validating the input message domain; and checking the destination mode when the output queue name is blank.

A sample Visual Basic program is included with this article that demonstrates these examples and produces a sample report.

FORM1FRM

```
VERSION 5.00
Object = "{F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0"; "COMDLG32.OCX"
Object = "{3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0"; "RICHTX32.OCX"
Begin VB.Form Form1
   Caption         =   "Message Flow Analyzer"
   ClientHeight    =   6285
   ClientLeft      =   60
   ClientTop       =   345
   ClientWidth     =   9690
   LinkTopic       =   "Form1"
   ScaleHeight     =   6285
   ScaleWidth      =   9690
   StartUpPosition =   3  'Windows Default
   Begin RichTextLib.RichTextBox rtbReport
      Height          =   5055
      Left            =   120
      TabIndex        =   5
      Top             =   600
      Width           =   9495
      _ExtentX        =   16748
      _ExtentY        =   8916
      _Version        =   393217
      ScrollBars      =   2
      TextRTF         =   $"Form1.frx":0000
   End
   Begin VB.CommandButton cmdCancel
      Cancel          =   -1  'True
      Caption         =   "Cancel"
      Height          =   495
      Left            =   1560
      TabIndex        =   4
      Top             =   5760
      Width           =   1215
   End
   Begin VB.CommandButton cmdAnalyze
      Caption         =   "Analyze"
      Default         =   -1  'True
      Height          =   495
      Left            =   120
      TabIndex        =   3
      Top             =   5760
      Width           =   1215
   End
   Begin MSComDlg.CommonDialog cdlFile
      Left            =   9120
      Top             =   5760
      _ExtentX        =   847
      _ExtentY        =   847
```

```
      _Version        =   393216
   End
   Begin VB.CommandButton cmdBrowse
      Caption         =   "Browse..."
      Height          =   375
      Left            =   852Ø
      TabIndex        =   2
      Top             =   9Ø
      Width           =   1Ø95
   End
   Begin VB.TextBox txtFile
      Height          =   285
      Left            =   48Ø
      TabIndex        =   1
      Top             =   12Ø
      Width           =   7935
   End
   Begin VB.Label Label1
      Caption         =   "File:"
      Height          =   255
      Left            =   Ø
      TabIndex        =   Ø
      Top             =   12Ø
      Width           =   375
   End
End
Attribute VB_Name = "Form1"
Attribute VB_GlobalNameSpace = False
Attribute VB_Creatable = False
Attribute VB_PredeclaredId = True
Attribute VB_Exposed = False
Option Explicit
Private xmlDoc      As MSXML.DOMDocument
Private mqSession   As MQAX2ØØ.mqSession
Private mqQMgr      As MQAX2ØØ.MQQueueManager
Private mqQueue     As MQAX2ØØ.mqQueue
Private Sub cmdAnalyze_Click()
'    On Error GoTo ERROR_HANDLER
    Dim strXPath As String, strQueue As String, strQMgr As String, _
        strNodeName As String, strBoQueue As String, i As Long, _
        strRmtQMgr As String, strRmtQueue As String
    Dim xmlNode         As MSXML.IXMLDOMNode
    Dim xmlInputNodes   As MSXML.IXMLDOMNodeList
    Dim xmlOutputNodes  As MSXML.IXMLDOMNodeList
    Dim xmlOutputNodes2 As MSXML.IXMLDOMNodeList
        Set xmlDoc = New MSXML.DOMDocument
    xmlDoc.Load txtFile.Text
    If xmlDoc.parseError <> Ø Then
        MsgBox xmlDoc.parseError.reason, vbCritical, "XML Parsing
Error"
```

```
        Exit Sub
    End If
    MousePointer = vbHourglass
    ' Get information on the input queue.
    strXPath = "//MessageProcessingNode[MessageProcessingNodeTypeRef/
@title='MQInput']/@xmi.label"
    Set xmlNode = xmlDoc.selectSingleNode(strXPath)
    If Not xmlNode Is Nothing Then
        strNodeName = xmlNode.Text
        rtbReport.Text = "Input Node: " & strNodeName & vbCrLf
    Else
        rtbReport.Text = "No Input Nodes"
    End If
    strXPath = "//MessageProcessingNode[MessageProcessingNodeTypeRef/
@title='MQInput']//Attribute[@xmi.label='queueName']/@value "
    Set xmlInputNodes = xmlDoc.selectNodes(strXPath)
    If xmlInputNodes.length = 0 Then
        rtbReport.Text = rtbReport.Text & "No Input Nodes"
    ElseIf xmlInputNodes.length = 1 Then
        strQueue = xmlInputNodes(0).Text
        strQMgr = "QMMQSIP1"
        rtbReport.Text = rtbReport.Text & "Input Queue: " & strQueue &
vbCrLf
        Set mqQueue = GetQueue(strQMgr, strQueue)
        If Not mqQueue Is Nothing Then
            strBoQueue = mqQueue.BackoutRequeueName
            rtbReport.Text = rtbReport.Text & vbTab &
"BackoutThreshold: " & _
                             mqQueue.BackoutThreshold & vbCrLf
            rtbReport.Text = rtbReport.Text & vbTab &
"BackoutRequeueName: " & _
                             strBoQueue & vbCrLf
            If strBoQueue = "" Then
                rtbReport.Text = rtbReport.Text & vbTab & _
                           "ERROR: No backout queue specified" & vbCrLf
            End If
            If mqQueue.BackoutThreshold = 0 Then
                rtbReport.Text = rtbReport.Text & vbTab & _
                           "ERROR: No backout count specified" & vbCrLf
            End If
            ' Verify that the backout queue exists.
            mqQueue.Close
            Set mqQueue = GetQueue(strQMgr, strBoQueue)
            If Not mqQueue Is Nothing Then
                If mqQueue.QueueType = MQ.MQQT_LOCAL Then
                    rtbReport.Text = rtbReport.Text & vbTab & _
                                     "WARNING: Specified backout queue
should not be local, unless it is read by another msg flow." & vbCrLf
                Else
                    strRmtQMgr = mqQueue.RemoteQueueManagerName
```

```
                    strRmtQueue = mqQueue.RemoteQueueName
                    Set mqQueue = GetQueue(strRmtQMgr, strRmtQueue)
                    If mqQueue Is Nothing _
                    And mqSession.ReasonCode = MQRC_NOT_AUTHORIZED Then
                        rtbReport.Text = rtbReport.Text & vbTab & _
                                         "WARNING: Could not verify
existence of " & strRmtQMgr & "/" & strRmtQueue & vbCrLf
                    ElseIf mqQueue Is Nothing Then
                        rtbReport.Text = rtbReport.Text & vbTab & _
                                         "ERROR: Specified backout
queue does not exist on the remote queue manager." & vbCrLf
                    Else
                        rtbReport.Text = rtbReport.Text & vbTab & _
                               "Remote QMgr: " & strRmtQMgr & vbCrLf
                        rtbReport.Text = rtbReport.Text & vbTab & _
                               "Remote Queue: " & mqQueue.Name & vbCrLf
                    End If
                End If
            Else
                rtbReport.Text = rtbReport.Text & vbTab & _
              "ERROR: Specified backout queue does not exist." & vbCrLf
            End If
        Else
            rtbReport.Text = rtbReport.Text & vbTab & "ERROR: Input
queue not found!!!" & vbCrLf
        End If
    End If
    rtbReport.Text = rtbReport.Text & vbCrLf
    rtbReport.Text = rtbReport.Text & vbCrLf
    ' Process the output nodes.
    strXPath = "//MessageProcessingNode[MessageProcessingNodeTypeRef/
@title='MQOutput']/@xmi.label"
    Set xmlOutputNodes = xmlDoc.selectNodes(strXPath)
    strXPath = "//MessageProcessingNode[MessageProcessingNodeTypeRef/
@title='MQOutput']//Attribute[@xmi.label='queueName']/@value"
    Set xmlOutputNodes2 = xmlDoc.selectNodes(strXPath)
    If xmlOutputNodes.length = Ø Then
        rtbReport.Text = "No Output Nodes"
        Exit Sub
    End If
    For i = Ø To xmlOutputNodes.length - 1
        rtbReport.Text = rtbReport.Text & "Node Name: " &
xmlOutputNodes(i).Text & vbCrLf
        strXPath = ".//Attribute[@xmi.label='queueName']/@value"
        strQueue = xmlOutputNodes2(i).Text
        rtbReport.Text = rtbReport.Text & vbTab & "Output Queue: " &
strQueue & vbCrLf
        ' Verify that the queue exists.
        Set mqQueue = GetQueue("QMMQSIP1", strQueue)
        If mqQueue Is Nothing Then
```

```
            rtbReport.Text = rtbReport.Text & vbTab & "ERROR: Output
queue does not exist!!!" & vbCrLf
        Else
            ' Verify that the queue is remote.
            If mqQueue.QueueType = MQ.MQQT_LOCAL Then
                rtbReport.Text = rtbReport.Text & vbTab & "WARNING:
Output queue should not be local unless it is read by another msg
flow." & vbCrLf
            Else
                rtbReport.Text = rtbReport.Text & vbTab & "Remote QMgr/
Queue:" & _
                                mqQueue.RemoteQueueManagerName & " / " & _
                                mqQueue.RemoteQueueName & vbCrLf
                ' Verify that the queue exists on the remote queue mgr.
               Set mqQueue = GetQueue(mqQueue.RemoteQueueManagerName, _
                                       mqQueue.RemoteQueueName)
                If mqSession.ReasonCode = MQRC_NOT_AUTHORIZED Then
                    rtbReport.Text = rtbReport.Text & vbTab & _
                                     "WARNING: Could not verify
existence of " & strRmtQMgr & "/" & strRmtQueue & vbCrLf
                ElseIf mqQueue Is Nothing Then
                    rtbReport.Text = rtbReport.Text & vbTab & "ERROR:
Output queue was not found on remote QMgr." & vbCrLf
                End If
            End If
        End If
        DoEvents
    Next
    MousePointer = vbNormal
    Exit Sub
ERROR_HANDLER:
End Sub
Private Sub cmdBrowse_Click()
    On Error GoTo CANCEL_EXIT
    cdlFile.DialogTitle = "Open Exported Message Flow"
    cdlFile.InitDir = App.Path
    cdlFile.CancelError = True
    cdlFile.Filter = "XML Files (*.xml)|*.xml|All Files (*.*)|*.*"
    cdlFile.ShowOpen
    txtFile.Text = cdlFile.FileName
CANCEL_EXIT:
End Sub
Private Sub cmdCancel_Click()
    Unload Me
    End
End Sub
Private Function GetQueue(ByVal strQMgr As String, _
                          ByVal strQueue As String) As MQAX200.mqQueue
    Dim mqQueue As MQAX200.mqQueue
    Set mqQMgr = mqSession.AccessQueueManager(strQMgr)
```

```
    If mqSession.CompletionCode = 2 Then
        Set mqQueue = Nothing
        Exit Function
    End If
    Set mqQueue = mqQMgr.AccessQueue(strQueue, MQ.MQOO_INQUIRE)
    If mqSession.CompletionCode = 2 Then
        Set mqQueue = Nothing
        Exit Function
    End If
    Set GetQueue = mqQueue
End Function
Private Sub Form_Load()
    Set mqSession = New MQAX200.mqSession
    mqSession.ExceptionThreshold = 3
End Sub
Private Sub Form_Resize()
    On Error GoTo BAIL_OUT
    Const MARGIN As Long = 60
    cmdBrowse.Left = ScaleWidth _
                   - cmdBrowse.Width _
                   - MARGIN
    txtFile.Width = cmdBrowse.Left _
                  - txtFile.Left _
                  - MARGIN
    rtbReport.Left = MARGIN
    rtbReport.Width = ScaleWidth - (2 * MARGIN)
    rtbReport.Height = ScaleHeight _
                     - rtbReport.Top _
                     - cmdAnalyze.Height _
                     - (2 * MARGIN)
    cmdAnalyze.Top = rtbReport.Top _
                   + rtbReport.Height _
                   + MARGIN
    cmdCancel.Top = cmdAnalyze.Top
BAIL_OUT:
End Sub
```

FORM1FRX

```
~
{\rtf1\ansi\ansicpg1252\deff0\deflang1033{\fonttbl{\f0\fnil\fcharset0
MS Sans Serif;}}
\viewkind4\uc1\pard\f0\fs17
\par }
```

PROJECT1VBP

```
Type=Exe
Reference=*\G{00020430-0000-0000-C000-
```

```
000000000046}#2.0#0#..\..\..\WINNT\System32\stdole2.tlb#OLE Automation
Reference=*\G{D63E0CE2-A0A2-11D0-9C02-
00C04FC99C8E}#2.0#0#..\..\..\WINNT\System32\msxml.dll#Microsoft XML,
version 2.0
Reference=*\G{B3927DD1-B888-11CF-A5F7-
444553540000}#5.1#0#..\..\..\program files\mqseries\bin\mqax200.dll#IBM
MQSeries Automation Classes for ActiveX
Object={F9043C88-F6F2-101A-A3C9-08002B2F49FB}#1.2#0; COMDLG32.OCX
Object={3B7C8863-D78F-101B-B9B5-04021C009402}#1.2#0; RICHTX32.OCX
Form=Form1.frm
Startup="Form1"
ExeName32="Project1.exe"
Command32=""
Name="Project1"
HelpContextID="0"
CompatibleMode="0"
MajorVer=1
MinorVer=0
RevisionVer=0
AutoIncrementVer=0
ServerSupportFiles=0
VersionCompanyName="WPSC"
CompilationType=0
OptimizationType=0
FavorPentiumPro(tm)=0
CodeViewDebugInfo=0
NoAliasing=0
BoundsCheck=0
OverflowCheck=0
FlPointCheck=0
FDIVCheck=0
UnroundedFP=0
StartMode=0
Unattended=0
Retained=0
ThreadPerObject=0
MaxNumberOfThreads=1
[MS Transaction Server]
AutoRefresh=1
```

PROJECT1VBW

```
Form1 = 4, 44, 701, 542, Z, 22, 22, 742, 537, C
```

Mills Perry
IT Consultant/Instructor, ZyQuest (USA)

Contributing to *MQ Update*

In addition to *MQ Update*, the Xephon family of *Update* publications now includes *CICS Update*, *MVS Update*, *TCP/SNA Update*, *DB2 Update*, *AIX Update*, *and RACF Update.*

Although the articles published are of a very high standard, the vast majority are not written by professional writers, and we rely heavily on our readers themselves taking the time and trouble to share their experiences with others.

Many have discovered that writing an article is not the daunting task that it might appear to be at first glance.

They have found that the effort needed to pass on valuable information to others is more than offset by our generous terms and conditions and the recognition they gain from their fellow professionals. Often, just a few hundred words are sufficient to describe a problem and the steps taken to solve it.

If you have ever experienced any difficulties with MQ, or made an interesting discovery, you could receive a cash payment, a free subscription to any of our *Updates*, or a credit against any of Xephon's wide range of products and services, simply by telling us all about it.

For a copy of our *Notes for Contributors*, which explains the terms and conditions under which we publish articles, please point your browser at www.xephon.com/nfc.

WebSphere MQ high availability options

INTRODUCTION

With the recent explosion of the Internet and e-business today's organizations are increasingly finding themselves in the position of having to operate on a round-the-clock basis. Because any unplanned system outages could have a great financial impact on the business, there is much more emphasis now on high availability (HA) design and planning than ever before.

Enterprise architects and designers must now endeavour to design systems so that business-critical applications and their associated infrastructure are kept running 24 hours a day.

The concept of 'high availability' encompasses many things so I want to establish that in this article I am talking only about the HA options that are specifically available to WebSphere MQ (WMQ) applications.

CONSIDERATIONS AND PLANNING

HA means different things to different people so it is important to define up front exactly what it means to your organization. True 24x7 operation implies zero downtime. In reality this is a very difficult thing to achieve. It is more common and more realistic for the organization to decide that 99.9% availability, for example, will meet its HA requirements. There will always be a need for at least a small amount of downtime in order to perform planned maintenance, system and application software upgrades, and other routine tasks.

Another important question that must be addressed is what data needs to be highly available and how it is to be accessed. For example, if your applications always need to GET messages from the target queues in your system, some type of hardware cluster will become an essential requirement. If, however, you are only concerned with having your applications PUT messages onto

queues, a WMQ Queue Manager cluster may be all that you require in order to achieve your HA requirements.

WMQ is only one piece of the HA puzzle. When designing your system you need to take into consideration all factors that, when combined, make up the application. This could include such items as databases, HTTP servers, application servers, DASD devices, and so on. Often people will implement a WMQ HA solution of some type but neglect to account for all the other complementary pieces of the overall system.

WHAT ARE THE OPTIONS?

When we talk about WMQ there are basically three options available to us. With those three options we can achieve varying degrees of HA.

- The first option is to build a queue manager cluster. We know that this functionality has been in the product for some time now and has become a very popular way to address not only HA but workload balancing as well, reducing the systems administration burden.
- Taking it one step further, we can implement a hardware cluster. Most vendors today have their own 'brand' of hardware clustering. For example, Microsoft has Microsoft Cluster Server, AIX has HACMP, HP-UX has Serviceguard, and so on. Even though each vendor puts its own spin on products, the principles of what makes up a hardware cluster and how it operates are the same, regardless of the implementation.
- We can also set up an environment where we use option one, the queue manager cluster, in conjunction with option two, the hardware cluster. This type of hybrid setup affords us the best of both worlds by giving us all of the advantages associated with each option.

Now that we know what options are available to us I think it bears repeating that an abundance of planning in advance is critical. Don't do any more than required in order to meet your specific requirements because all that will do is add unnecessary cost and

complexity to your environment. In the simplest of terms, use the right tools for the job at hand, nothing more, nothing less.

QUEUE MANAGER CLUSTERS

The way that we achieve high availability within a WMQ cluster is to have multiple instances of the same queue hosted on more than one queue manager in the cluster.

If the destination queue manager fails and the message is still on the transmission queue, it will be rerouted to another instance of the target queue elsewhere in the cluster. If, however, the message is already on the target queue when the queue manager fails but has not been processed, it will not be available until after the failing queue manager has been restarted.

Figure 1 illustrates two clustered queue managers, QM1 and QM2. Notice that Qb is hosted on both of these queue managers so that if, for example, QM1 fails, the message traffic destined for Qb can be rerouted to QM2.

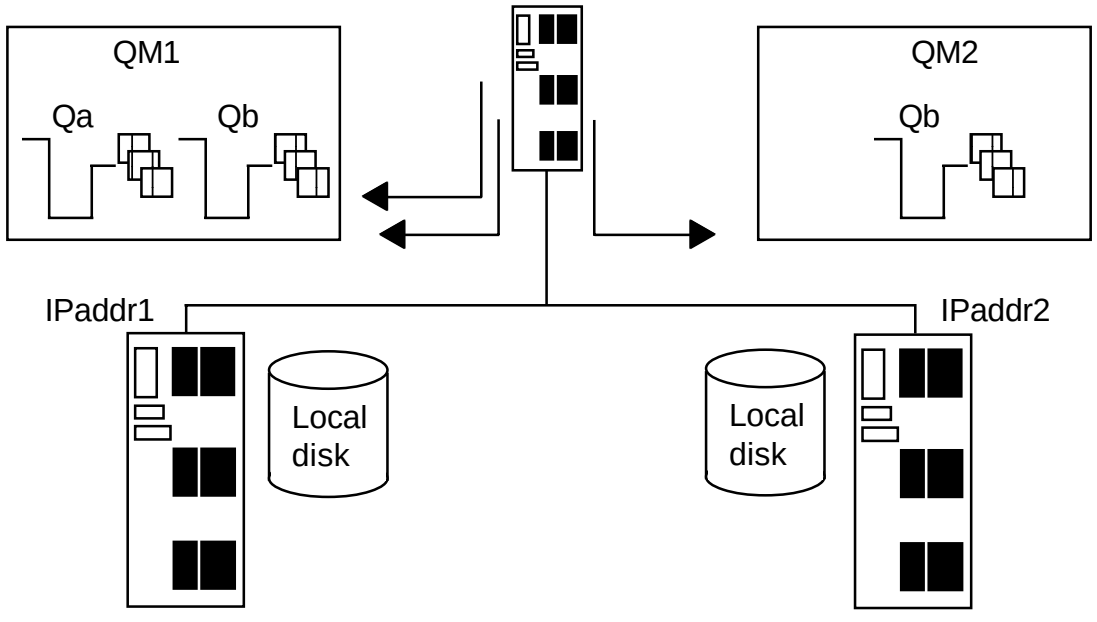

Figure 1: Two clustered queue managers

In a hardware-clustered environment the cluster software monitors the resources on the node. In the event of some type of failure the cluster software will initiate a failover process, whereby resources from the failed node are moved to and started on another node in the cluster. When the failing node is repaired, a failback will be initiated to move the resources back to their original node. Most cluster software gives you the ability to either manually or automatically initiate the failback process. The two most common types of hardware cluster configuration are active/passive and active/active.

Active/passive

In an active/passive setup one of the two nodes is nominated to be the primary or active node. Under normal operating conditions the application runs on the active node. The passive node simply sits and waits for a failure to occur, at which point in time a failover occurs and it becomes the active node, running the application and its resources.

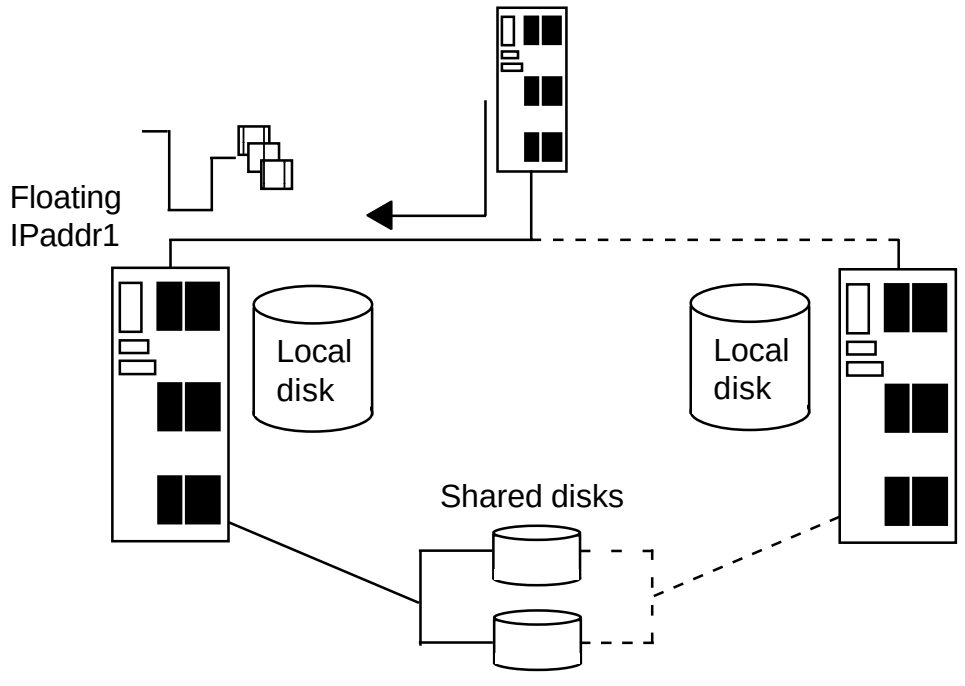

Figure 2: WMQ running in an active/passive hardware cluster

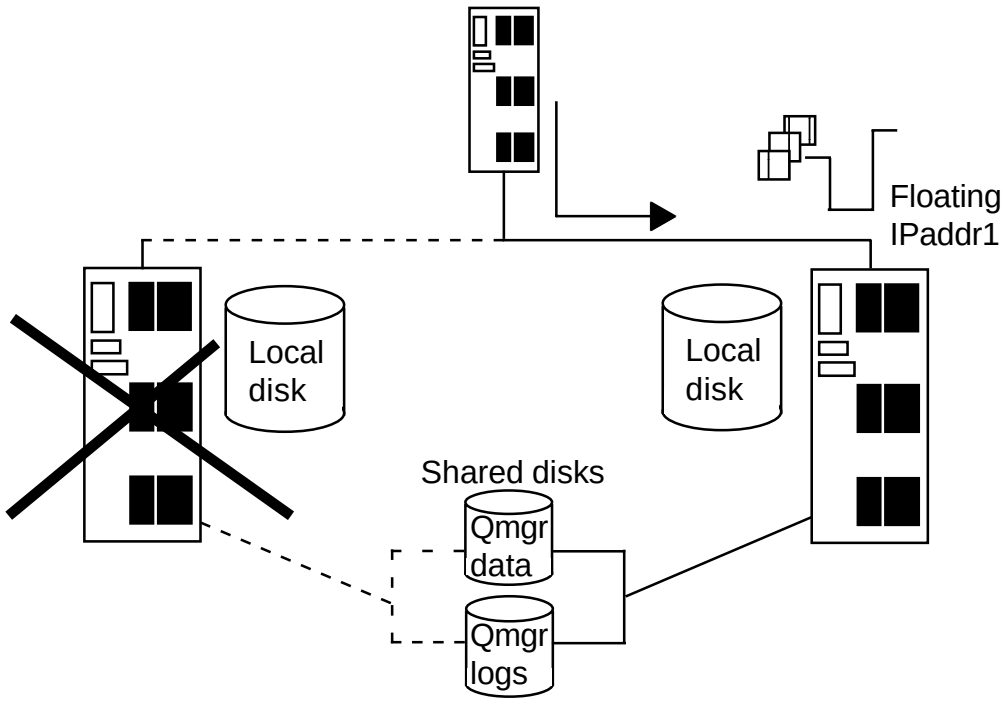

Figure 3: The passive node becomes active after a failover

The two nodes both use a shared disk device but it is important to understand that only one node is ever connected to the disk at any time. There is also what is known as a virtual, or cluster, IP address. This is a floating IP address that stays the same no matter what side of the cluster happens to be active at any given time. This is the IP address that applications outside of the cluster use to communicate with applications inside the cluster.

In WMQ terms, the queue manager's data and log files reside on the shared disk and in the event of a failure on the active node the queue manager processes are moved across to the passive node and started.

Figures 2 and 3 illustrate WMQ running in an active/passive hardware cluster.

Active/active

The second way to configure an HA hardware cluster is in an active/active configuration. In this type of setup both machines

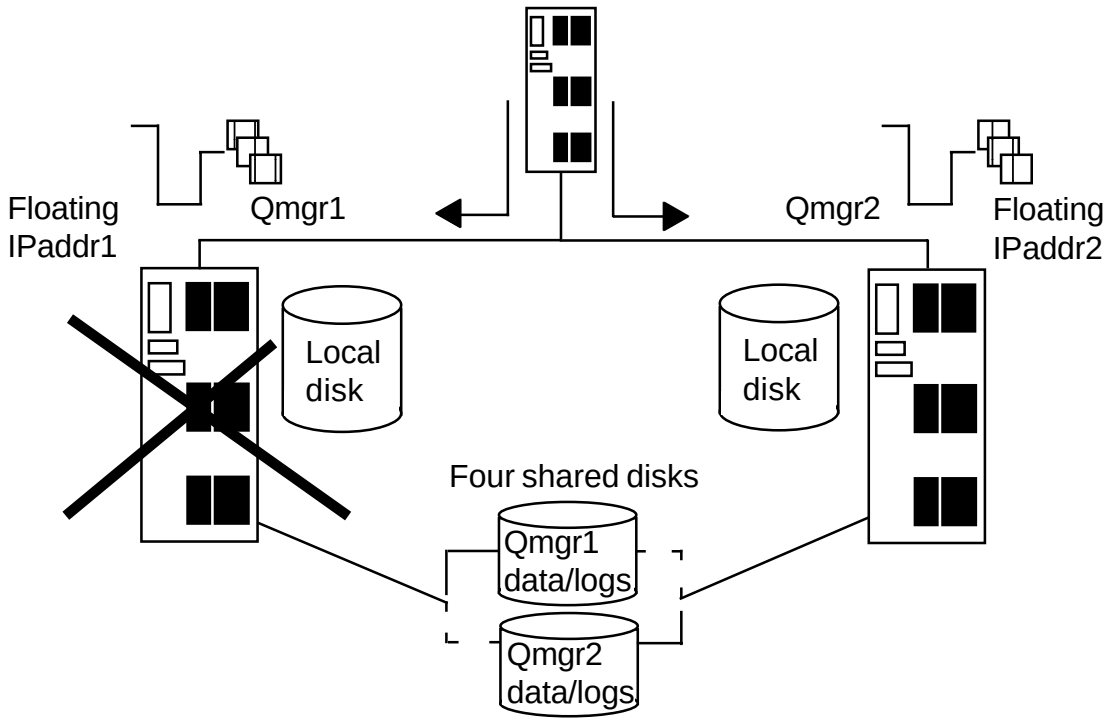

Figure 4: WMQ running in an active/active hardware cluster

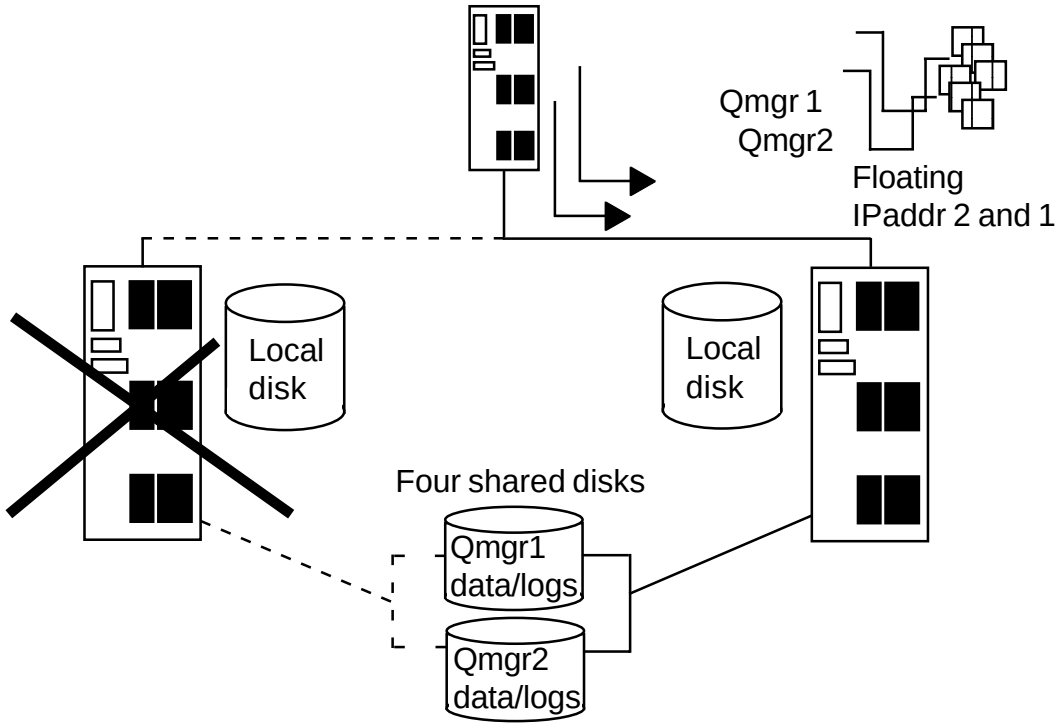

Figure 5: Queue manager migrates to surviving machine after a failover

are each concurrently running their own queue manager. In the event of a failure on one machine its queue manager is migrated to the surviving machine, which will now have two queue managers running on it. To an outside application it will appear as though nothing has changed.

It is important to remember, when provisioning the hardware for an active/active cluster, that during the time of a failure the surviving machine will have to handle basically double its workload (its own plus that of the failed machine). With this in mind you need to ensure that both machines are adequately resourced.

In an active/active setup we also have the concept of virtual or floating IP address as well as shared disk devices. However, in this case, each machine will have its own set of shared disks.

Figures 4 and 5 illustrate WMQ running in an active/active hardware cluster.

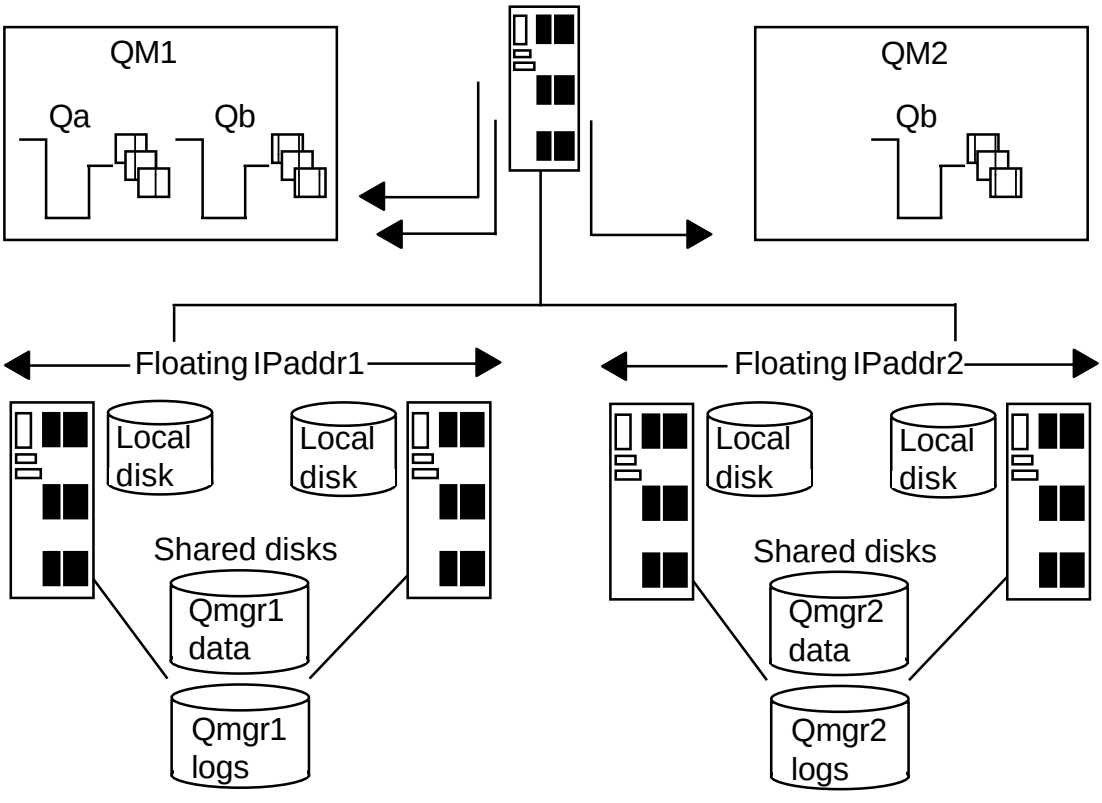

Figure 6: A hybrid WMQ cluster/hardware cluster

QUEUE MANAGER CLUSTER PLUS HARDWARE CLUSTER

As I mentioned earlier, you can use a combination of a WMQ cluster in conjunction with a hardware cluster. This will give you all of the benefits of the MQ cluster, such as workload balancing and reduced administration, along with the principal benefit of a hardware cluster – failover support.

The use of an HA passive node to partner each node in the WMQ cluster provides automatic recovery of any queue managers that were running on the node should it fail. This minimizes the time that messages are unavailable as a result of being stuck in a destination queue hosted by the failed queue manager.

With an HA cluster that supports active/active operation all nodes can be running queue managers that are participants in a WMQ cluster and each node can be a standby for one of the other nodes.

Figure 6 illustrates a hybrid WMQ cluster/hardware cluster.

APPLICATION CONSIDERATIONS

Making your WMQ environment highly available is of little use if the applications that make use of MQ services are not highly available. So you will need to put in some thought and most likely some effort to make your applications 'cluster aware'.

In WMQ terms the unit of failover is the queue manager. You will need to decide what constitutes a unit of failover for your application. Most likely (but not necessarily) it would be an instance of the running application. There follows a list of some other factors you will need to consider when making your application programs highly available.

- When the application fails over are there any configuration files that need to move as well?
- Is any configuration data machine-specific?
- Is any synchronization between machines required?
- Are you going to monitor actively the health of your own application? If so what method will you use?

- Are there any application/message affinities that you need to consider?
- How will you test?

These are just a few questions to think about to help you get started. The main point I want to make is that there will be factors that need consideration at the application level.

HELPFUL SUPPORTPACS

As is usual in 'MQ land' IBM has provided some very helpful SupportPacs to help us design and implement HA WMQ systems. The following is a list of relevant SupportPacs that can be found on the WMQ Web site.

- *IC61 – Configuring WebSphere MQ Integrator for AIX with HACMP.*
- *IC62 – Configuring WebSphere MQ Integrator for Sun Solaris with Sun Cluster 2.x.*
- *IC63 – Configuring WebSphere MQ Integrator for Sun Solaris with VERITAS Cluster Server.*
- *IC71 – Configuring WebSphere MQ Integrator for Windows NT/2000 with Microsoft Cluster Server.*
- *MC63 – MQSeries for AIX, implementing with HACMP.*
- *MC68 – Configuring MQSeries with Compaq TruCluster for high availability.*
- *MC69 – Configuring MQSeries with Sun Cluster 2.x.*
- *MC6A – Configuring MQSeries for Sun Solaris with VERITAS Cluster Server.*
- *MC6B – WebSphere MQ for HP-UX, implementing with Multicomputer/Serviceguard.*
- *MC74 – MQSeries for Windows NT/2000 Microsoft Cluster Server support.*

- *MD05 – MQSeries design considerations for large clusters.*

SUMMARY

We all know that highly available systems are more important now than ever before. Outages cost businesses money. It is, therefore, important to make HA an important piece of your system architecture and design.

This article has outlined the ways in which you can incorporate WebSphere MQ into an HA environment. It's not difficult; however, it does take a good deal of planning. I would urge you to take a look at the SupportPacs that apply to your situation, do some research and some planning, and make your next WMQ project highly available.

Dale Eckert
Middleware Architect (USA)

E-mail alerts

Our e-mail alert service will notify you when new issues of *MQ Update* have been placed on our Web site. If you'd like to sign up, go to http://www.xephon.com/mq and click the 'Receive an e-mail alert' link.

WMQI Broker plug-in node: performance analysis

This article describes the performance analysis and optimization for a WebSphere MQ Integrator (WMQI) Broker plug-in node. This plug-in is part of the WebSphere Business Integration for Financial Networks (WBI for FN) product.

INTRODUCTION

WBI for FN is structured into a base product and extensions. The base product provides capabilities to deliver products on top of WMQI Broker, including customization and common services, such as auditing and timing.

The extensions provide real business value to customers; they can be provided by IBM, ISVs, or customers themselves. An IBM extension that was made available in December 2002 is the Extension for SWIFTNet (ESN). This extension allows financial applications to access the new Secure IP Network provided by SWIFT.

At the end of 2002 a performance evaluation was performed on z/OS for the FIN processing part of the SWIFTNet extension. One outcome of the performance evaluation was that a plug-in node provided as part of the WBI for FN product takes most of the processing time of the Extension for SWIFTNet message flows. This article describes the performance analysis for this node and the optimizations carried out in order to reduce the processing time it requires.

THE CONFIGURATION-DATA PROVIDER NODE

The node that was identified as using most of the processing time was the configuration-data provider node (CPN). This node can be used in message flows to get configuration information.

WMQI Broker allows you to develop message flows. A message flow is composed of message processing nodes that perform

actions on a message. These nodes can be customized with resource properties, such as a WMQ queue name or data source name to access a database. These properties are statically defined for the nodes.

WBI for FN provides a configuration service that enables the provision of dynamic configuration information into a message flow. This information is structured into organizational units, configuration object types, and configuration objects.

Organizational units, eg different departments in an enterprise, are used to separate the resources and security settings in a message flow. A configuration object type describes possible attributes of object types used in the message flow processing of, for example, a printer. The configuration objects are instances of the configuration object types in an organizational unit. They hold the actual values for the attributes, as Figure 1 illustrates.

The organizational units, configuration object types, and configuration object information can be defined in WBI for FN. This

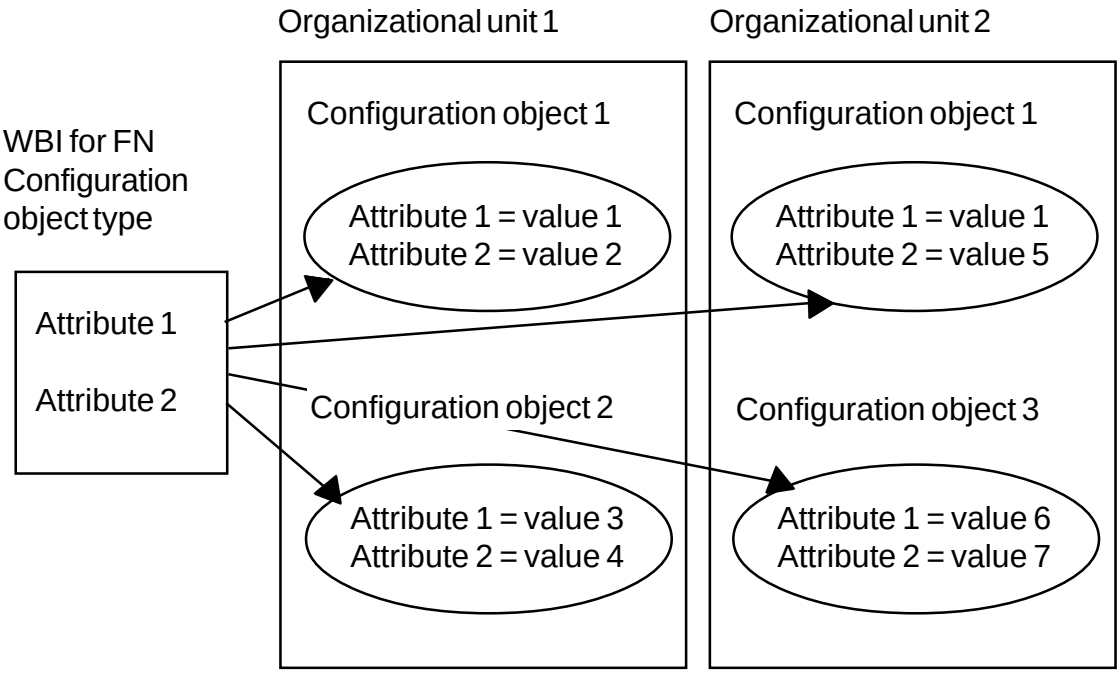

Figure 1: WBI for FN configuration model

information is stored in a database table and can be used to augment a message while it is processed in WMQI Broker. Based on this information the message flow can determine whether a specific operation, eg auditing, needs to be performed or which resources to use, for example, the name of a WMQ queue to output a message.

Most message flows use many configuration objects, which can be grouped into a Configuration Object Set (COS). Each COS has a name, as does each configuration object.

The name of a COS can be used as a property for a WBI for FN message processing node. This node is the configuration-data

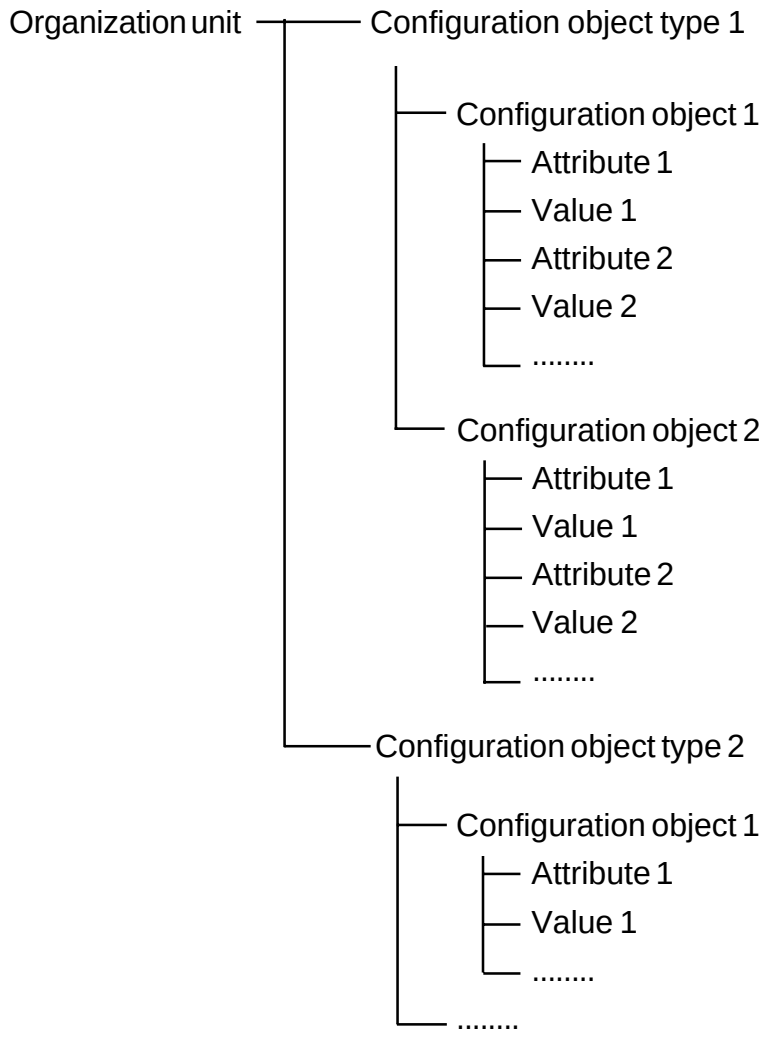

Figure 2: WBI for FN configuration information structure

provider node. It checks the message being processed and loads all objects defined in the COS that belong to the organizational unit referenced in the message. This data is then inserted at a defined place into the message. A typical structure is shown in Figure 2.

Reading the information from the database table is very time-consuming, which is why it's not retrieved every time. Instead, the CPN caches the information and regularly checks to see whether the information has changed in the meantime. Checking for changes is much cheaper (in time) than retrieving the configuration information. This is the reason why a second message being processed by the CPN needs much less time than the first.

PERFORMANCE MEASUREMENT METHOD

Based on the overall time spent by the CPN node our first assumption was that the plug-in API functions needed too much time, so we started an investigation into which plug-in API functions were time-intensive and how often these functions were called.

WMQI Broker doesn't provide any information on plug-in API calls in its traces that can be used for performance analysis. To get meaningful results our code could not be modified because this would negate the results.

To monitor CPN processing time a Performance Library has been developed as an interface library between the plug-in node and the WMQI Broker plug-in node API library.

This Performance Library allows each WMQI plug-in API call to be traced and the elapsed time of each API call to be measured. Additionally, a summary statistic of the API calls is created at defined API calls. The behaviour of the Performance Library can be controlled by a profile.

The Performance Library can be used on distributed platforms simply by renaming the Performance Library and the WMQI Broker library. On z/OS the plug-in nodes must be relinked with the Performance Library and the original WMQI Broker API is dynamically loaded by the Performance Library. This structure is shown in Figure 3. The main advantage of this structure is that the

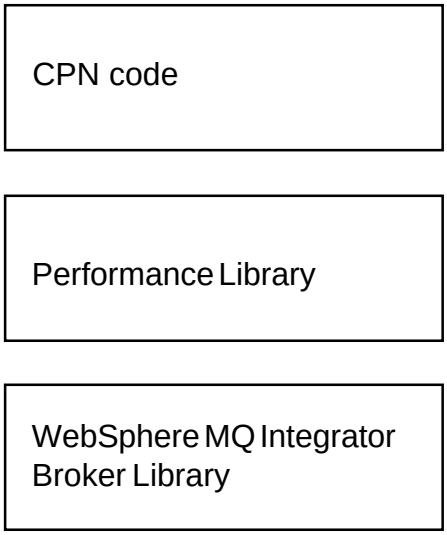

Figure 3: How use of the Performance Library is structured

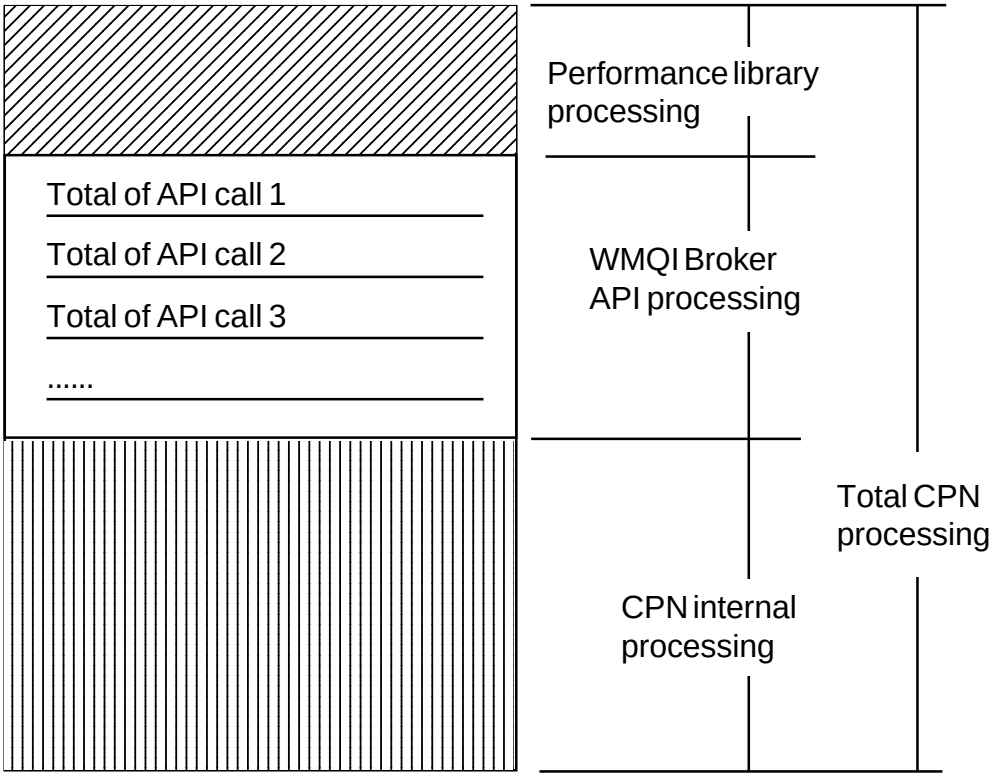

Figure 4: Time for CPN processing

source code of the node does not need to be modified.

To determine the overall processing time of the configuration provider plug-in node, from calling the plug-in node to propagating the message to the out terminal, additional code has been added that writes the elapsed time for the configuration provider processing (including the Performance Library processing) into a trace file. Figure 4 shows the CPN processing time. With the performance statistics of the Performance Library trace and the values retrieved from this additional trace the following CPN values were determined:

- Overall processing time of the CPN plug-in node.
- Overall processing time needed for WMQI Broker API calls in total and as the sum of each call.
- The total number of each WMQI Broker API function called.
- Overall processing time needed for processing in the plug-in node other than for the WMQI Broker API calls.

A test message flow was used to measure the processing time of the CPN. This flow consisted of an MQInput node, the CPN, and a WMQI Broker-provided trace node.

```
- MQMD
- MQRFH2 header with two NameValueData folders:
      <mcd>
         <Msd>xml<Msd>
      </mcd>
      <ComibmDni>
         <OU>SYSOU</OU>
      </ComibmDni>
-XML Payload:
      <Level1>
        <Level2>
          <Level3>Ay Data 1</Level3>
          <Level3>Any Data 2</Level3>
          <Level3p>Any Data 1p</Level3p>
        </Level2>
        <Level2p attr="Hello">Now the value</Level2p>
      </Level1>
```

Figure 5: Elements of the test message

The CPN was configured to use a COS consisting of six simple configuration objects. The configuration objects had between one and three attributes. The CPN does not only provide the configuration information itself, it also provides some other environment information that can be retrieved from the broker environment but is not available, for example, in ESQL. Such information might be the name of the broker, for example. That's why 105 elements are inserted into the message.

A non-persistent WMQ message, as shown in Figure 5, was used as a test message. A first message was put on the input queue after starting the broker. The processing of this first message also

WMQI Broker API call	**Total execution time (µs)**	**Total number of calls**
cniRootElement	674	118
cniCreateElementAsLastChild	1,080	105
cniSetElementType	726	209
cniSetElementName	635	122
cniSetElementcharacterValue	991	73
cniElementCharacterValue	3,320	448
cniCopyElementTree	348	1
cniSearchNextSibling	2,170	464
cniSearchFirstChild	4,992	963
cniFinalize	101	1
cniSqlCreateStatement	7,927	4
cniSqlSelect	402,509	4
cniSqlDeleteStatement	275	4
cciMbsToUcs	3,697	155
Total time	1,015,532	
Performance library time consumption	957,981	
WMQI Broker plug-in APIs	432,754	
Other CPN functions	57,551	

Table 1: Result of processing the first message

included the configuration data loading of the configuration provider plug-in node.

The performance measurement was repeated with a COS consisting of 35 configuration objects. The configuration objects are a mix of different types, with one, two, or three attributes. Along with the standard information the CPN had to insert 463 elements.

INITIAL RESULTS

The results for running the first message with the CPN are listed in Table 1. This table (and also all following tables) contains only the information on calls that used a significant amount of time. All measurements were done on z/OS. Similar information could be measured on other platforms but did not form part of this exercise. These results show a ratio of 1:7 between the CPN and WMQI Broker API l .

As Table 1 illustrates, the Performance Library utilizes a considerable amount of time. This time includes that required for WMQI Broker plug-in APIs; since it has already been measured it can, therefore, be omitted when comparing the results.

The results show that the time in the CPN is mainly due to the SQL calls retrieving information from the database tables, which takes a long time. This amounts to nearly 95% of the total WMQI Broker plug-in API functions. As this was expected we ran the second message.

The configuration information was already available for processing this second message, so this part of the processing is omitted. The processing results are shown in Table 2. These results show a ratio of 8:3 between the CPN and WMQI Broker API l.

Running the same message flow without loading the data but having the COS with 35 configuration objects shows results as listed in Table 3. Many more elements are inserted into the message with this scenario. These results show a ratio of 8:3 between the CPN and WMQI Broker API l .

WMQI Broker API call	Total execution time (µs)	Total number of calls
cniRootElement	608	110
cniCreateElementAsLastChild	893	105
cniSetElementType	766	209
cniSetElementName	661	122
cniSetElementcharacterValue	957	73
cniCopyElementTree	193	1
cniSearchNextSibling	2136	464
cniSearchFirstChild	4803	926
cniFinalize	100	1
cciMbsToUcs	3514	143
Total time	357,070	
Performance library time consumption	334,846	
WMQI Broker plug-in APIs	14,956	
Other CPN functions	40,224	

Table 2: Result of processing the second message

ANALYSIS AND OPTIMIZATION

Looking at the results of the three tests it was noted that a lot of time was spent retrieving the configuration data. As this process is only carried out for the first message it was not deemed to be a problem.

The measurements with a smaller and a larger number of messages highlighted a couple of issues. The first point to note is that the relationship between CPN processing and WMQI Broker is nearly stable. The WMQI API calls indicate that the most time-consuming calls and functions were called by the CPN in order to insert the configuration information into the message. Most time is spent

WMQI Broker API call	Total execution time (µs)	Total number of calls
cniRootElement	2,875	528
cniCreateElementAsLastChild	4,291	463
cniSetElementType	3,807	985
cniSetElementName	3,439	598
cniSetElementcharacterValue	3,970	315
cniCopyElementTree	204	1
cniSearchNextSibling	51,764	10,560
cniSearchFirstChild	25,471	4,673
cniFinalize	100	1
cciMbsToUcs	17,828	737
Total time	3,179,895	
Performance library time consumption	2,898,989	
WMQI Broker plug-in APIs	115,406	
Other CPN functions	280,906	

Table 3: Results of processing the second message and inserting more elements

with cniSearchNextSibling, cniSearchFirstChild, and cniMbsToUCS.

Our analysis showed that this is because of the way the elements are inserted. The CPN used a method that can be compared with a SET ESQL statement:

```
SET complete path 1 = value 1;
SET complete path 2 = value 2;
```

This means that the complete path must always be navigated before a single value is inserted. This was done to ensure that all elements of the path exist. As the tables show, the time consumed by navigating the message tree is considerable.

To reduce this navigation expenditure we changed the coding in the CPN to use relative addressing. Relative addressing can be compared with reference variables that are navigated back and forth as needed, eg:

```
DECLARE Cursor REFERENCE to OutputRoot.MQRFH2;
...
MOVE Cursor FIRSTCHILD;
MOVE Cursor NEXTSIBLING;
MOVE Cursor PARENT;
MOVE Cursor PREVIOUSSIBLING;
```

WMQI Broker API call	Total execution time (µs)	Total number of calls
cniRootElement	67	1
cniCreateElementAsLastChild	929	105
cniSetElementType	676	209
cniSetElementName	592	122
cniSetElementcharacterValue	896	73
cniElementCharacterValue	3,166	448
cniCopyElementTree	352	1
cniSearchNextSibling	184	36
cniSearchFirstChild	1,186	236
cniFinalize	106	1
cniSqlCreateStatement	7,685	4
cniSqlSelect	366,747	4
cniSqlDeleteStatement	233	4
cciMbsToUcs	3,535	155
Total time	766,609	
Performance Library time consumption	729,792	
WMQ I Broker plug-in APIs	389,402	
Other CPN functions	36,817	

Table 4: Result of processing the first message with optimized CPN

From the comparison of CPN processing in relation to the WMQI processing for the node it was also obvious that the processing in the CPN also needed optimization.

Our analysis showed that much of the time is utilized by WBI for FN tracing calls. Tracing was not activated so the calls themselves are not doing very much, it was just the overhead of the call itself. A single call was not a problem but for the third measurement the tracing function was called more than 75,000 times. In total these calls used half of the total CPN processing time so the tracing has been reworked and reduced. Having made these changes the performance measurements were repeated for the CPN.

FINAL RESULTS

All three measurements as done for the non-optimized CPN have

WMQI Broker API call	**Total execution time (µs)**	**Total number of calls**
cniRootElement	25	1
cniCreateElementAsLastChild	725	105
cniSetElementType	691	209
cniSetElementName	593	122
cniSetElementcharacterValue	815	73
cniCopyElementTree	193	1
cniSearchNextSibling	176	36
cniSearchFirstChild	1,076	215
cniFinalize	104	1
cciMbsToUcs	3,191	143
Total time	160,458	
Performance library time consumption	141,806	
WMQI Broker plug-in APIs	7,886	
Other CPN functions	18,652	

Table 5: Results gained from processing the second message with optimized CPN

been repeated. The results for the first test, where the configuration information is loaded, are shown in Table 4.

These results show a ratio of 1:10 between the CPN and WMQI Broker API l .

Looking at the results it can be seen that loading the configuration data still took most of the processing time. Nevertheless the optimized CPN was now nearly 40% faster than the original CPN. This is due to the fact that the number of calls to cniRootElement, cniSearchNextSibling, and cniSearchFirstChild reduced dramatically.

The effect of reducing the number of trace calls can be seen from

WMQI Broker API call	**Total execution time (µs)**	**Total number of calls**
cniRootElement	23	1
cniCreateElementAsLastChild	3,191	463
cniSetElementType	3,483	985
cniSetElementName	2,981	598
cniSetElementcharacterValue	3,177	315
cniCopyElementTree	194	1
cniSearchNextSibling	6,142	1,278
cniSearchFirstChild	5,227	991
cniFinalize	102	1
cciMbsToUcs	166	8
Total time	787,523	
Performance library time consumption	689,868	
WMQI Broker plug-in APIs	26,090	
Other CPN functions	97,655	

Table 6: Results gained from processing the second message with optimized CPN and inserting more elements

the fact that now much more time is spent in the WMQI Broker API function in relationship to the time in the CPN.

To understand whether the results for reducing the number of plug-in API calls or the reduction of WBI for FN calls was more significant the second test was repeated with the same message. Using the available configuration information the results for this processing are shown in Table 5.

This results in a ratio of 2.2:1 between the CPN and WMQI Broker API l.

This is nearly the same as for the original CPN. This means that both parts have been reduced by almost the same factor. Nevertheless, the overall processing time for the CPN was reduced by nearly 58%.

The last measurement for the CPN was made running the same message with the configuration data preloaded but using the COS that inserts 35 configuration objects. The results of this test are shown in Table 6.

These results show a ratio of 10.1:3 between the CPN and WMQI Broker API l.

From the results in Table 6 it can be seen that before this measurement another change was implemented. An analysis of the calls showed that most of the calls to cciMbsToUcs are done for constants. The conversion of these constants was moved to the initialization of the node. Therefore the time and the number of calls were drastically reduced.

For this scenario the result of the optimization of the configuration provider is an overall processing time reduction of nearly two-thirds. The number of WBI for FN trace calls has been reduced by nearly 80% to 15,800. The rest of the CPN processing time has been reduced by 50%. The WMQI Broker processing time for the API calls has been reduced by approximately 77%. This results in an increase of 25% in the CPN:WMQI Broker API ratio. That is the result of shifting a part of the message tree navigation task from WMQI Broker to the CPN.

CONCLUSION

Changing the way elements are inserted into the message tree has significantly reduced the amount of time needed to perform the task. Nevertheless, to understand whether this result is good or not we coded a new message flow. This was a copy of the original message flow but replaced the CPN node with a compute node.

In this compute node the operation of the CPN was simulated with hard-coded ESQL SET ??? statements to insert all data elements as was done for the third test for the 35 configuration objects. The elapsed time for the compute node was determined using a WMQI Broker user trace. This test showed the processing time of the compute node to be 16.064 µs. This is much less than the time for the WMQI Broker plug-in APIs in the CPN alone and indicates that it would be very difficult to improve upon the compute node. The results of our tests clearly show that the performance of WMQI-delivered nodes is more efficient than that of plug-in nodes.

Michael Groetzner
IBM (Germany)

MQ news

IBM has recently announced that it is to introduce new versions of its WebSphere Business Integration Event Broker and WebSphere Business Integration Message Broker solutions, formerly known as WebSphere MQ Event Broker and WebSphere MQ Integrator Broker.

WebSphere Business Integration Brokers are designed to deliver real-time, highly personalized information across a network, the Internet, telemetry, and pervasive devices. The software is claimed to secure delivery of critical industry-specific information, such as stock quotes or pressure readings from gas pipelines.

For more information contact your local IBM representative.

* * *

MQSoftware has recently announced support for WebSphere MQ Everyplace (WMQE) with Q Pasa! V3. WMQE is designed to support secure wireless and pervasive device solutions that extend e-business communications across all customer, employee, and partners.

MQSoftware claims that its support will give users more powerful monitoring capabilities across more WebSphere products that utilize mobile devices.

For more information contact:
MQSoftware,1660 South Highway 100, Suite 400, Minneapolis, Minnesota 55416, USA.
Tel: +1 952 345 8720.
Fax: +1 952 345 8721.
Web: http://www.mqsoftware.com

MQSoftware, Surrey Technology Centre, 40 Occam Road, Surrey Research Park, Guildford, Surrey, GU2 7YG, UK.
Tel: +44 1483 295400.
Fax: +44 1483 573704.

* * *

Fiorano Software recently announced the release of Tifosi 2002 ESB (TESB), Enterprise Service Bus, a standards-based solution for event-based distributed computing.

The company claims that by using an ESB organizations can avoid a ‘spaghetti’ of application interconnections because existing applications, services, and other data sources need only to plug into the bus to communicate.

Although it is implemented on top of FioranoMQ, the company states that Tifosi can leverage existing middleware investments such as WebSphere MQ or any of the other JMS servers on the market.

For more information contact:
Fiorano Software, 718 University Avenue, Suite 212, Los Gatos, CA 95032, USA.
Tel: +1 408 354 3210.
Fax: +1 408 354 0846.
Web: http://www.fiorano.com

* * *