WebSphere MQ
White paper

WebSphere® software

Increase the value of CICS applications with WebSphere MQ.

Author
Dermot Flaherty, senior technical staff member and lead architect, WebSphere MQ Product and Technologies

Contributors
- *Mark Cocker, software engineer, CICS Technical Strategy and Planning*
- *Carolyn Elkins, IT specialist, Advanced Technical Support*
- *John Tilling, senior software engineer, CICS Technical Planning and Strategy*
- *Steve Zemblowski, executive IT specialist, Advanced Technical Support*

February 2008

Contents

Introduction

This white paper is intended for IT architects and application developers who have an existing investment in the IBM CICS® platform and would like to see how they can get more out of that investment. It examines the ways in which IBM WebSphere® MQ can increase the processing capability of IBM CICS Transaction Server. WebSphere MQ achieves this by extending the reach of CICS from a wide range of computing platforms both inside and outside of your enterprise to access business data of all types (file, database and so on). The result is an increase in the business value of your investment in the CICS platform.

We will also see how the innate, loosely coupled style of programming supported by WebSphere MQ provides great flexibility in the placement of business logic, and how it naturally supports a move toward a service oriented architecture (SOA).

This paper does not provide a detailed comparison of the various connectivity mechanisms available to the CICS programmer. Instead, it will examine in general terms the differences between them and their suitability for particular application requirements.

The paper will also illustrate how IBM's enterprise service bus (ESB) products can provide additional business value.

Increased connectivity means increased business

A growing industry focus on SOA has led clients to view connectivity as a key part of the overall aim of discovery and creation of business services. Clients are aware that services can be used and reused to provide greater flexibility and increased speed of deployment of new solutions to meet changing business needs.

Enterprises today have a huge ongoing investment in the IBM System z™ platform. Mainframes contain significant core assets in business data and applications with an estimated applications-replacement cost of US$20 trillion. Furthermore, today's businesses depend on the ability of CICS to process billions of transactions a day with a total value of more than US$1 trillion a week.[1] In this context, it's worth remembering that many of us use System z applications and technology whenever we use an ATM to withdraw cash.

Given that background and the growing investment by both clients and IBM in the System z platform, there is an increasing need to see how these assets can be reused more effectively. A big part of this focus is on connectivity.

In 2006, IBM defined five entry points to help clients get started with SOA adoption. One of these is the connectivity entry point. Within this space, IBM defined the need for a messaging backbone as the foundation for connectivity and introduced the concept of an ESB to enable more-intelligent and more-dynamic routing of service requests.

Messaging backbone, SOA and ESB

One of the key principles of SOA is to decouple the physical implementation of a software component from its representation or interface. In this way, the service definition and its interface can remain stable for as long as the business requires, shielding them from lower level technology changes. For example, you might need to run a credit checking service as part of a business process. It is highly likely that the application, function or provider that executes this will change over time, but with a service oriented approach, processes that use these services are unaffected by these changes.

Enabling all applications and data sources to be able to communicate in this way requires a messaging backbone. The messaging backbone can be thought of as a pervasive communications infrastructure. It provides reliable and secure data communication between applications on all computing platforms and in all execution environments. And it provides an inherently loosely coupled way of integrating applications.

The term *loosely coupled* refers to the ability to make changes to application deployments without necessarily requiring the connected applications themselves to be changed.

As well as providing connectivity, the messaging backbone should also be able to provide guaranteed qualities-of-transport service (for example, reliable message delivery). It should allow higher-level functions, such as those embodied in an ESB, to focus on service delivery.

WebSphere MQ provides messaging-based connectivity between applications running on IBM z/OS®, Linux for IBM System z™, Linux®, Microsoft® Windows®, IBM AIX®, IBM System i™ and Sun Solaris – in fact, all the major computing platforms. WebSphere MQ is absolutely relied on by more than ten thousand IBM customers who run it on 40 or so computing platforms in more than 80 platform configurations. In addition, there are clients who use WebSphere MQ networks today to carry data worth US trillions of dollars a day.

The key point is that WebSphere MQ has proven reliability in many different client environments, and for them WebSphere MQ is their messaging backbone.

Now let's relate the messaging backbone to ESB technology. Summed up briefly, we could say that WebSphere MQ connects disparate applications reliably using a common programming model but does not understand the application data that it is carrying. Although WebSphere MQ will perform code-page conversion as data in, for example, Spanish or German flows between a Windows platform (ASCII) and a z/OS platform (EBCDIC), it has no inherent knowledge of the data format.

An ESB has the capability to understand the data in the WebSphere MQ message. From this understanding, it can take actions based on data values, it can transform the data and it can enrich the data from external sources. This ability eases the integration to CICS transactions from multiple sources.

For example, an XML message that uses one list of units-of-measure abbreviations can be converted easily to a COBOL copybook format that uses a slightly different list of abbreviations. The ESB can also be used to supply basic information, such as the CICS program that needs to be run, so that client programs do not have to be aware of the service provider.

Or an ESB could decide, based on either data content or some other service criteria, that a particular service request should be processed through one of a number of different mechanisms depending on a service classification. As we will see, this is an important point when we examine the different interaction patterns as we compare the asynchronous queued style of communications, as embodied in WebSphere MQ, with, for example, direct forms of communication, such as connectors from IBM CICS Transaction Gateway that use Java™ 2 Platform, Enterprise Edition (J2EE) Connector architecture (JCA) .

WebSphere MQ provides a messaging backbone that is supported by all of IBM's ESB offerings – IBM WebSphere Message Broker, IBM WebSphere Enterprise Service Bus and IBM WebSphere DataPower® Integration Appliance XI50.

What is WebSphere MQ?

As a market leader in message-oriented middleware, WebSphere MQ allows clients to reliably integrate disparate applications across a wide variety of computer software and hardware. It provides a simple, yet functionally rich application programming interface, known as the Message Queue Interface (MQI), across all its supported platforms. And it fully implements the industry-standard Java Message Service (JMS) interface. These interfaces essentially support application-to-application communication through queues and allow those applications to not be concerned with the location of those queues or who processes them.

The concept of the queue represents the service interface mentioned above. It abstracts from the physical implementation and location of the application being interacted with. As a result, the developer is unaffected by changes to the destination application; the WebSphere MQ infrastructure takes care of that. Queues are hosted by queue managers, and the MQI provides the bridge from the application to a WebSphere MQ queue manager. Queues can either be local to a queue manager or hosted on another queue manager. In the latter case, WebSphere MQ provides transmission facilities to transport the message data. Applications communicate with one another using PUT and GET verbs, which allows great flexibility in creating application-interaction patterns ranging from pseudo-synchronous request-and-response styles, to fire-and-forget styles and mixtures of both.

A request-response interaction pattern is just what its name suggests. To the requester, the application appears to be a completely synchronous invocation of a server application with a response from the server after execution. But it is in fact implemented by two distinct asynchronous, one-way message exchanges. The MQI provides a simple way in which these two flows can be combined to make a *pseudo-synchronous* pattern.

Many WebSphere MQ clients use just such an interaction pattern to support customer-facing interactions and achieve satisfactory response times that meet or exceed their business requirements.

Similarly, the fire-and-forget style applies to a one-way message exchange in which a response is not required. WebSphere MQ provides reliable message delivery to help ensure that the message data is propagated to the target queue.

These styles are shown in Figure 1.

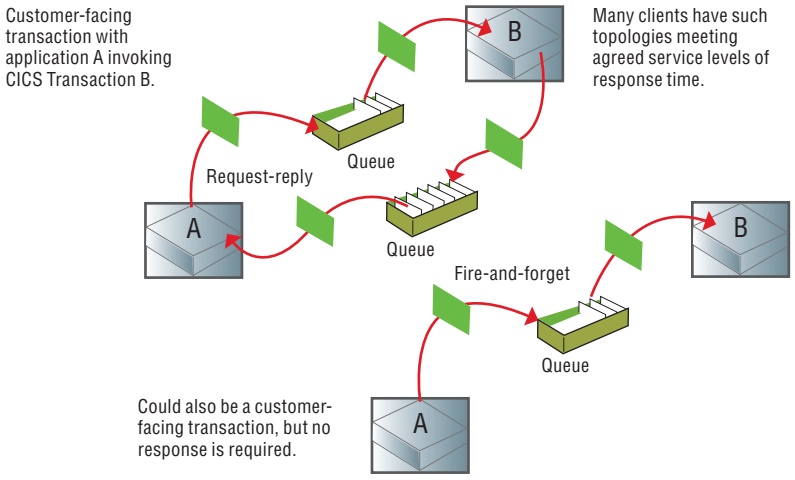

Figure 1. Pseudo-synchronous or asynchronous styles

It can be readily seen how the two styles can be combined so that a nonreturn, fire-and-forget flow can take advantage of additional processing capacity to enable parallel execution. This is shown in Figure 2, where we see that Transaction B has created a nonreturn message flow for further processing in either another CICS transaction or perhaps on another WebSphere MQ platform.

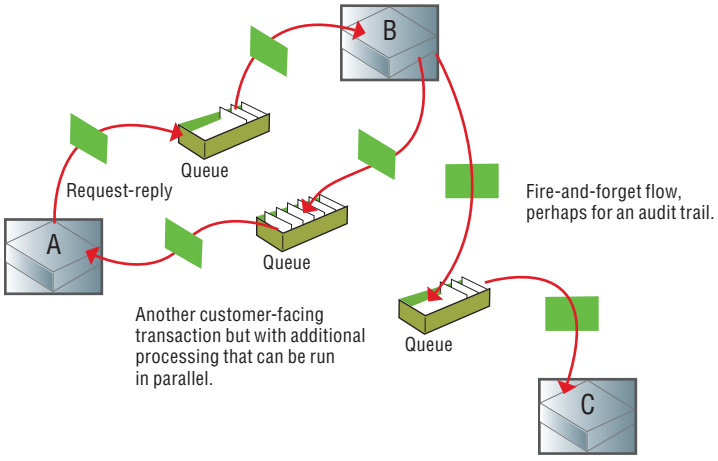

Figure 2. Pseudo-synchronous and asynchronous styles enable parallel processing

Transactional support is provided by WebSphere MQ—both as a resource manager coordinated by transaction coordinators such as CICS or Resource Recovery Services (RRS), and as an XA programming-interface coordinator on distributed platforms. This helps to ensure that messages can be delivered with full transactional integrity, providing assured delivery in support of business-critical applications.

Naturally, in a real-world scenario in which CICS is the execution environment, the server Transaction B (in the examples shown in the diagrams) will attempt to ensure that the consumption of the request message, the generation of the reply message and the generation of the fire-and-forget flow are all performed under syncpoint control. In the event of transaction failure, the updates (including the consumption of the request message) would be rolled back.

This is shown in Figure 3, which also shows an IBM DB2® update as part of the processing in Transaction B.

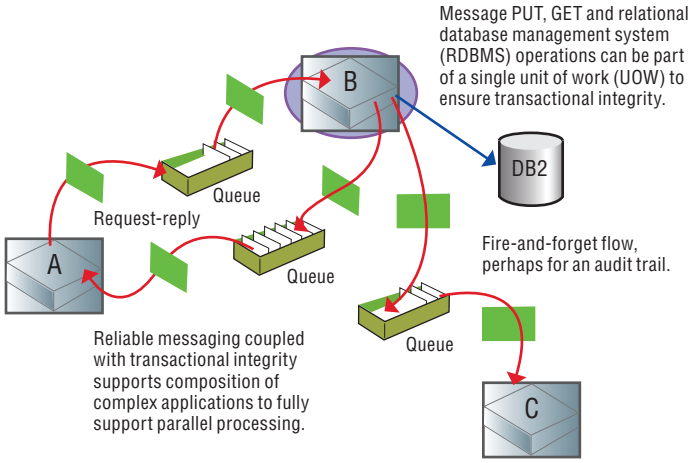

Figure 3. Transactional support ensures all or nothing

WebSphere MQ can efficiently carry data of any type and of any size (which is achieved by grouping physical messages to create logical messages of unlimited size). With Version 6, individual messages can be 100 MB in size across all platforms (including IBM Parallel Sysplex® shared message queues on z/OS).

Message retrieval can be sequential (on a first-in-first-out basis), selective (based on a message or correlation identifier that can be set by the application when the message is created) or subject to message priority. Message priority can also be set at message-creation time to provide support for expedited message delivery.

The use of a message identifier, correlation identifier or both facilitates the very common request-response message-interaction pattern in which the response carries the original message or correlation identifier, and the response message is one of a number going to the same reply queue.

WebSphere MQ and workload delivery

As we have seen, WebSphere MQ provides a simple yet powerful set of application programming interfaces (APIs) to support both reliable data transport and an extremely flexible range of application-interaction patterns across all the major computing platforms. In this way, it can be thought of as a consistent method for delivering work to processing environments, such as CICS. In the following sections, we'll look at two important ways that WebSphere MQ provides a *business data pipe* for CICS.

The WebSphere MQ programming model uses queues as an indirection mechanism for application communication. The following example shows how request-response interactions can easily be constructed using this model:

1. *Application A issues a PUT command to send a message to the request queue (specifying a reply queue for the response message).*
2. *Application A issues the appropriate COMMIT verb. (This has the effect of releasing the request message for subsequent retrieval by Transaction B.)*
3. *Application A issues a GET command (with the Wait option) to receive a message from the reply queue.*
4. *CICS Transaction B issues a GET command to receive a message from the request queue.*
5. *CICS Transaction B processes the message (perhaps creating a DB2 update and a further message for subsequent processing).*
6. *CICS Transaction B generates the response message.*
7. *CICS Transaction B issues an EXEC CICS SYNCPOINT call to commit the resources (including the WebSphere MQ messages) processed in the transaction. (This has the effect of releasing the response message for delivery to the waiting Application A.)*
8. *Upon receipt of the response message, the GET command issued by Application A is satisfied, and the application then completes processing, commits the message update and ends.*

We can readily see that this is a classic client-server application pair with CICS Transaction B taking the part of the server application. In messaging terms, Transaction B is a *message-driven application*, in the sense that its work is provided by the message it retrieves and its processing is typically bounded by the receipt of the message and the final PUT issued for the reply.

If we look at this in more detail, we typically find the following pattern for CICS Transaction B:

1. *Transaction B starts.*
2. *Transaction B issues a GET command to receive a message from the request queue.*
3. *While there are no more request messages, Transaction B takes these actions:*
   a. Processes the message
   b. Issues a PUT command to send a message to the reply queue.
   c. Issues an EXEC CICS SYNCPOINT call.
   d. Issues a GET command to receive a message from the request queue. (It waits if no message is available.)
4. *Transaction B ends.*

So we can see that after the transaction starts, it will process the queue until there are no more messages left to process. WebSphere MQ provides facilities for allowing a GET message to wait for a specified interval to allow for *late* messages to arrive (to prevent premature ending of the transaction).

This style of CICS application is not only very efficient but also scalable, because multiple transactions can safely process messages in parallel.

As shown in the pattern, CICS Transaction B will retrieve the request message, process it and generate the response message. If the processing requires a DB2 update (for example), then depending on the business requirements the update can be performed within the single unit of work that is processing the request message (as illustrated in Figure 3), or it can be processed in parallel by the simple mechanism of generating a fire-and-forget message. The DB2 update could actually be performed by another CICS transaction similar to Transaction C shown in Figure 3.

Another important WebSphere MQ facility in this scenario is support for *triggering*, which allows an application to start when messages first arrive on a queue. So CICS Transaction B would not have to be a long-running transaction waiting for messages to arrive, but would be started when (for example) messages first arrive on an empty request queue.

The key point here is that through the use of message queues, WebSphere MQ provides a flexible *pull* model for processing data, in which the processing programs do not need to wait for input but instead can be triggered when there is work to do. This is an extremely efficient way to use CICS resources, especially when combined with the features for threadsafe transactions in CICS Transaction Server, Version 3.2

Also notice that this model is well-suited to the notion of an application queue, in which messages of a similar type — for example, those that require processing by the same application — are placed on the same queue. This is a common pattern.

Now let's look at a case in which there is no requirement for message ordering, in which each message would represent a piece of work, such as an order-entry update, that is independent of its position in the message queue. In such a case, the pull model easily supports multiple instances of server transactions (B or C) processing a particular queue to maximize throughput. Each message carries the information required to process it, including the message- and correlation-identifier information as well as the reply-queue name. Therefore, it can be processed by any instance of the CICS server transaction, provided there are no other affinities.

It can be seen that the pull model and triggering facilities provided by WebSphere MQ provide a powerful and flexible alternative to more-traditional *push* models. As a result, we can avoid the typical issues we get with push models in which the target is not available.

WebSphere MQ naturally supports work to be processed at rates that the CICS system can sustain. To better understand this support, we will now look at related aspects of workload delivery.

WebSphere MQ clustering

If we look at the pull-model and triggering delivery style of WebSphere MQ from an availability perspective, it can be seen that if there is any problem with the queue itself, CICS Transaction B or even the application-owning region (AOR) in which Transaction B runs, then we have a problem with the delivery of the message data into CICS. WebSphere MQ provides two mechanisms to address this. One is a platform-wide facility called *WebSphere MQ clustering*, which we will look at now, and the other is support of *sysplex shared queues*, which we will examine later.

WebSphere MQ clustering provides two powerful facilities that can be used to reduce administration and provide workload distribution.

In the preceding example, we've seen how Application A, running perhaps on a distributed platform, can interact with CICS Transaction B, running on z/OS. (Application A could also be another CICS transaction or another program running on z/OS.) The applications communicate as indicated, and all they need to know is the names of the queues they are processing.

But clearly some administration is required to define the queues to WebSphere MQ and to set up the paths to help ensure successful transmission of message data between, for example, a WebSphere MQ queue manager running on Windows and one running on z/OS. WebSphere MQ provides a number of artifacts, particularly channels and transmission queues, that are required to establish intercommunication between two instances of WebSphere MQ. And it provides commands to define these artifacts.

WebSphere MQ clustering reduces this administration by allowing WebSphere MQ queue managers to participate in a *cluster*. The key feature of a WebSphere MQ cluster is a repository, accessible throughout the cluster, which contains information about the location of *cluster queues*.

A cluster queue is defined through normal WebSphere MQ commands but known throughout the WebSphere MQ cluster and accessible from anywhere in the cluster without the need for additional definitions. This reduces the WebSphere MQ administration required and removes potential sources of definition errors.

In addition, multiple instances of a cluster queue can be defined (one per queue manager). WebSphere MQ provides support for workload balancing among the instances for applications that perform a series of PUT operations to a cluster queue. Figure 4 illustrates this concept.

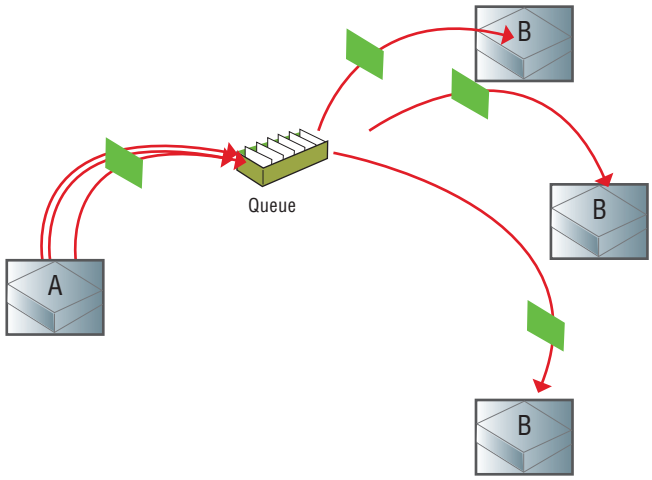

Figure 4. Application A issues a PUT command to write three messages to a cluster queue.

Figure 4 shows a variation on the previous examples. Here we have Application A sending three messages to what it considers a single queue. But the queue is defined to be a cluster queue. The result is that WebSphere MQ distributes the messages to the three actual instances of the queue. In this case, the instances are assumed to be hosted by three queue managers, each of which is being accessed by three CICS AORs, and each of which, in turn, is running an instance of Transaction B.

The result is that we not only achieve improved overall availability by spreading the workload across a number of processing programs, but we achieve a high degree of parallel processing, because each instance of Transaction B can overlap its execution with other instances. Note that this support is completely transparent to both Application A and Transaction B. WebSphere MQ will spread the workload among the cluster queue instances according to the workload algorithm selected, and in this way, the message data can be spread across multiple AORs to ameliorate the effects of a particular AOR (or WebSphere MQ queue manager) not being available. In WebSphere MQ, Version 6, there is support for simple round-robin distribution as well as more-sophisticated spreading. (For example, you can select a particular route only if others are not available.)

WebSphere MQ clustering is a widely used technique for spreading work across the AORs in a CICS environment to increase the overall availability.

If a particular cluster queue is unavailable (for example, because the WebSphere MQ queue manager that hosts that instance is down), new messages can still reach other cluster instances, but any messages on the failed instance remain unavailable until the availability is restored. (This is true of all queues that are *owned* by individual queue managers.) To address this issue, WebSphere MQ provides sysplex shared queues to provide maximum message availability for z/OS.

WebSphere MQ support of sysplex shared queues

To provide the maximum availability of message data, WebSphere MQ on z/OS provides unique technology that makes use of the coupling facility to hold a *shared* queue.

In this case, the shared queue is not owned by any individual queue manager and, as a result, is isolated from an outage of an individual queue manager or logical partition (LPAR).

As with WebSphere MQ clustering, the use of shared queues is transparent to the application, which can use the normal WebSphere MQ facilities – message priority, persistence, sequential or selective retrieval and triggering, for example – to process them. With WebSphere MQ for z/OS, Version 6, shared messages can now be up 100 MB in size, which is the same limit as that for nonshared messages.

The key feature of shared queues is their availability across the sysplex. This enables CICS transactions to run in multiple AORs and access the same queue. This is illustrated in Figure 5.

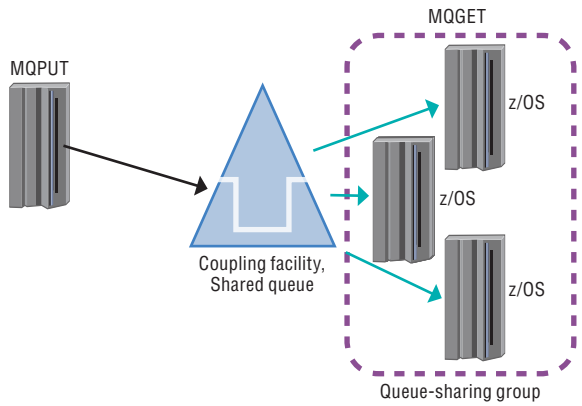

Figure 5. WebSphere MQ for z/OS shared queues

Figure 5 shows multiple processing applications all able to access the same shared queue in a queue-sharing group within the sysplex.

In the examples that we've been using with CICS Transaction B, we can see how multiple instances of Transaction B can be running in AORs across the sysplex, all processing messages from the same shared-request queue to maximize throughput.

Furthermore, although applications running outside the sysplex cannot retrieve messages directly from the shared queues (because GET operations can access only those queues that are local to the connected queue manager), they can use PUT operations to write messages to a shared queue.

WebSphere MQ supports dynamic, virtual IP addressing so that sysplex distributor can be used to route connections around the sysplex under the control of z/OS Workload Manager. Alternatively, WebSphere MQ clustering can also be used for PUT operations to write messages to a shared queue.

Additionally, WebSphere MQ maintains knowledge, through the queue-sharing group, of the transactional state of all the shared-queue resources. In this manner, it provides support for dynamic backout of shared-queue messages within a unit of work in case a connected queue manager fails. This is illustrated in Figure 6.

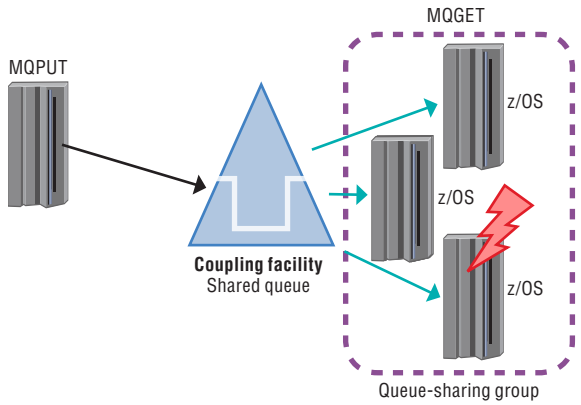

Figure 6. Tolerance of queue-manager and server outages

Figure 6 shows an outage of one of the queue managers in the queue-sharing group (QSG). If this occurs, the surviving members of the QSG roll back any incomplete units of work that were in flight at the time of the outage.

So, if we had an instance of Transaction B in flight at the time of the outage (for example, if it had retrieved a message from the shared-request queue and had started its processing but had not reached CICS syncpoint), the surviving members of the queue-sharing group roll back the request message from the shared queue (and any other shared messages created or retrieved within the unit of work).

In this way, WebSphere MQ will act to ensure maximum availability of shared messages for planned and unplanned queue-manager outages.

WebSphere MQ support of the CICS bridge

The previous sections have shown how the use of WebSphere MQ can provide an extremely consistent and powerful way to access data from a wide variety of sources and make them accessible to CICS transactions in a highly available manner.

The examples have all shown CICS transactions applications using the MQI within their processing programs to gain access to WebSphere MQ queues and message data. But WebSphere MQ also provides support for *implicit* access to WebSphere MQ by use of the CICS bridge.

This means you can reuse your existing CICS transactions that were controlled by 3270-connected terminals to be controlled by WebSphere MQ messages, without having to rewrite, recompile or relink those transactions.

The CICS bridge enables an application that is not running in a CICS environment to run a *program* or *transaction* on CICS and get a response back. The application can be run from any environment that has access to a WebSphere MQ network that encompasses WebSphere MQ for z/OS.

A *program* is a CICS program that can be invoked using the EXEC CICS LINK command. It must conform to the distributed program link (DPL) subset of the CICS API. That is, it must not use CICS terminal or syncpoint facilities.

A *transaction* is a CICS transaction designed to run on a 3270 terminal. This transaction can use basic message service (BMS) or terminal control (TC) commands. It can be conversational or part of a pseudoconversation. Issuing syncpoints is permitted.

The CICS bridge allows an application to run a single CICS program or set of CICS programs. Where multiple CICS programs are being invoked, the bridge supports scenarios in which a response has to be received before the next program can be run (synchronous processing) as well as those scenarios in which this is not the case (asynchronous processing).

The CICS bridge also allows an application to run a 3270-based CICS transaction, without knowledge of the 3270 data stream. It uses standard CICS and WebSphere MQ security features. You can configure the bridge to authenticate, trust or ignore the requestor's user ID. Given this flexibility, there are many instances in which you can use the CICS bridge:

- *Write a new WebSphere MQ application that needs access to logic or data, or both, that resides on your CICS server.*
- *Run CICS programs from an IBM Lotus® Notes® application.*
- *Access your CICS applications from any distributed platform where WebSphere MQ is available.*

With WebSphere MQ, Version 6, it is now possible to run multiple instances of the CICS bridge-monitor task, accessing the same WebSphere MQ bridge queue. Coupled with the ability to use CICS transaction routing with CICS Transaction Server, Version 2.2, this feature allows a wide range of flexibility in setting up the bridge environment. Figure 7 illustrates what is possible with WebSphere MQ, Version 6.

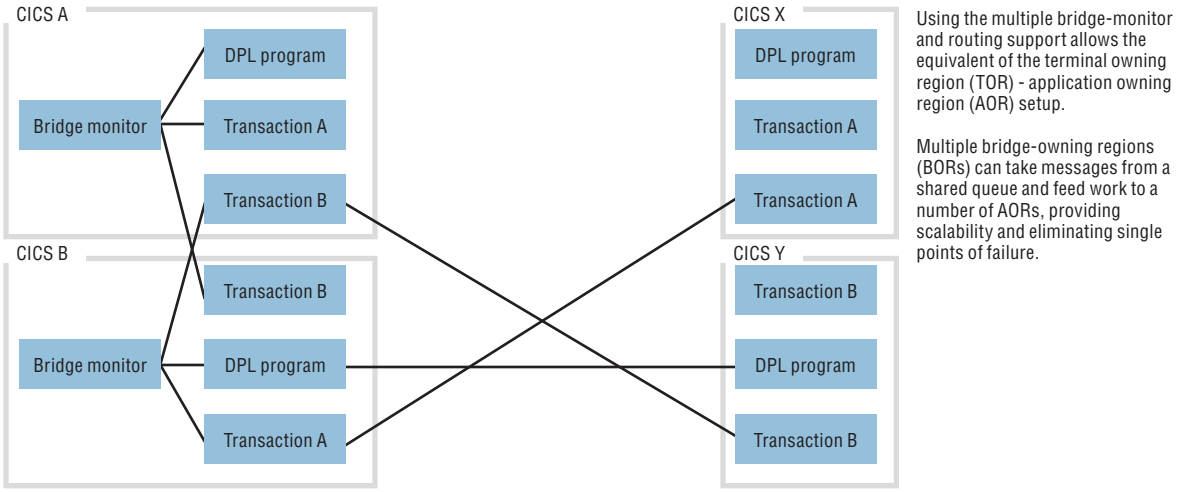

Figure 7. Bridge-monitor tasks running in multiple AORs

Figure 7 shows bridge-monitor tasks running in multiple AORs, accessing the same shared queue and being able to drive work across the CICS AORs.

Recent enhancements to CICS and WebSphere MQ connectivity

The components to connect CICS and WebSphere MQ for z/OS have been enhanced and are now integrated with CICS Transaction Server, Version 3.2.

These components include the CICS-MQ adapter, the CICS-MQ trigger monitor and the CICS-MQ bridge. The enhancements include:

- *Exploitation of the CICS open transaction environment. The components have been made threadsafe and are enabled to use CICS open trusted computing bases (TCBs). Exploitation of the CICS open transaction environment will benefit threadsafe applications using WebSphere MQ. For multiple WebSphere MQ requests, TCB switching can be avoided, resulting in a saving of CPU and an increase in overall throughput, because applications can now run on multiple open TCBs and avoid bottlenecking.*
- *Improved diagnostics, including use of CICS facilities for system trace, dump formatting and messages.*
- *Improved statistical information, including the use of connections between CICS and WebSphere MQ, and the type and success of calls made. This information has also been made available within the IBM CICSPlex® System Manager Web user interface.*

WebSphere MQ and the Web

The previous sections have shown how WebSphere MQ and the wider WebSphere MQ network can provide a number of ways to deliver work to CICS. We've also seen how CICS transactions can *explicitly* access WebSphere MQ message data through the use of the MQI, or be driven *implicitly* by using the CICS bridge.

The growing interest in SOA and Web Services has led to the ability to define CICS transactions as *services* that can be invoked by SOAP requests over HTTP transport.

WebSphere MQ, Version 6 provides the capability to invoke those same CICS services over a a WebSphere MQ network. This enables service invocation over the reliable WebSphere MQ network, which can be used for much wider data access.

In addition to what might be viewed as standard Web services using SOAP, there is a growing interest in the world of rich Internet applications and Web 2.0. Many clients see AJAX and RESTful Web services as a way of rapidly creating new dynamic applications with appropriate qualities of service to meet business needs.

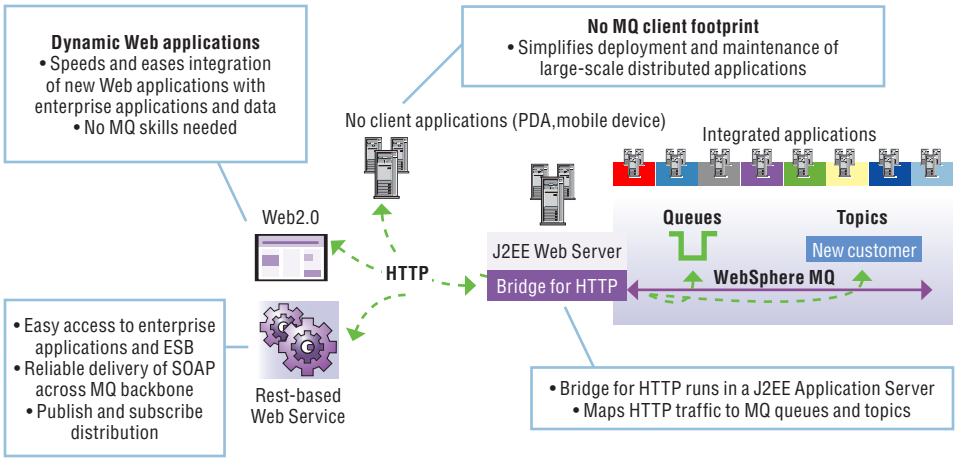

Figure 8. The WebSphere MQ Bridge for HTTP enables Web 2.0 applications to access business information securely and reliably.

WebSphere MQ has recently released SupportPac MA0Y – IBM WebSphere MQ Bridge for HTTP – which enables Web 2.0 developers to use WebSphere MQ itself as a service. This allows Web 2.0 developers to connect their applications into the messaging backbone for SOA which, in turn, connects them to the wider range of enterprise information.

WebSphere MQ will also provide a standardized way of describing WebSphere MQ applications as services, which in turn will allow them to be cataloged in registries. This capability will help them be reused as services in composite SOA applications.

This focus on services in general and Web 2.0 support in particular provides an increasing range of ways to access data for processing in CICS.

WebSphere MQ for z/OS helps ensure secure and auditable access to business data

Security is a key client concern in a number of areas. The primary areas of concern are connections to the queue managers and queue-manager networks, the data passing into the queue managers, operational integrity and data protection.

Connection security is usually implemented by using Secure Sockets Layer (SSL) support for TCP/IP channels, provided natively by WebSphere MQ. In addition, security exit points can be used to prevent unauthorized connections.

To address the security of operations and access to WebSphere MQ objects on z/OS, WebSphere MQ uses a System Authorization Facility (SAF) interface to control all command and administration access. Through the SAF interface, you can use IBM Resource Access Control Facility (IBM RACF®), ACF/2, Top Secret and other applications as your security manager on z/OS.

End-to-end security is a key concern in cases in which an application that is located in a WebSphere MQ network (but not necessarily on z/OS) wants to send encrypted-message data to a recipient on z/OS. To address this concern, IBM provides an additional feature through WebSphere MQ Extended Security Edition.

Summary

WebSphere MQ provides a rich set of features that allow a wide range of data to be accessed from and processed by your CICS applications. CICS applications can use the functionally rich interface of WebSphere MQ (MQI) explicitly, or they can have implicit access using the CICS bridge. The CICS bridge enables you to reuse your existing CICS transactions to work with WebSphere MQ without the need to rewrite, recompile or relink your transactions. CICS services can be driven by SOAP requests over WebSphere MQ. This enables service invocation over the reliable WebSphere MQ network, which can be used for much wider data access. WebSphere MQ has recently extended its support for the Web through SupportPac MA0Y. And it provides a number of security features to protect your business data. All of these features can help provide greater usage and value for your existing CICS installation.

For more information

To learn more about WebSphere MQ for z/OS, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/integration/wmq/zos/

To learn more about CICS Transaction Server for z/OS, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/htp/cics/

IBM Corporation
Software Group
Route 100
Somers, NY 10589
U.S.A.

Produced in the United States of America
02-08

[1] Teubner, Russ. 2003. Integrating CICS applications as Web services. SOAWorld Magazine. http://www.webservices.sys-con.com/read/39850.htm (accessed September 10, 2007).

WSW14006-USEN-01