Using IBM WebSphere® Message Broker in a Microsoft .NET Shop

Authors:

Jason Armstrong,
Glen Wirth,
and
Adrian Wright

Summa Technologies, Inc.
925 Liberty Avenue, 6th Floor
Pittsburgh, PA 15222, USA
(412) 258-3300

August 31, 2012
Document Version: v1.0

Table of Contents

Executive Summary

Based on Summa's experience, companies of all sizes have found it necessary to integrate more and more systems to add additional value to the business functions that IT supports. The number and complexity of the integrations being built have required architects to adopt one or more integration tools into their application development portfolios.

Since 1996, Summa has been helping companies large and small modernize and integrate systems. Many of those systems have been deployed on a variety of integration platforms including IBM WebSphere® Message Broker and Microsoft BizTalk®. In the past, the default assumptions were to use BizTalk for integrating Microsoft-based solutions and IBM WebSphere Message Broker for legacy and Java integration projects.

Message Broker V8 challenges those assumptions by providing first-class support for .NET and a close integration between Message Broker Toolkit and Microsoft Visual Studio®. The tight integration makes it easier to build solutions quicker, faster, and cheaper than developers can with just Microsoft BizTalk.

The off-the-shelve performance difference between the two products also heavily favors Message Broker due to its 6.5 times faster response times over BizTalk when processing web service requests.

Microsoft .NET shops would do well to consider Message Broker V8 for a variety of reasons including:

- First-class integration with .NET and Visual Studio
- High productivity for development and maintenance
- Performance-to-Cost ratio
- Customer support and responsiveness
- Broad protocol support
- Reliability and high availability
- Ease of management and deployment

Developers and architects familiar with Microsoft BizTalk are well aware of its limitations. Including second-class support for Visual Studio, low developer productivity, and performance and reliability issues. Combining that with Microsoft's attention shifting to Windows Azure™ there has been confusion about the role of Microsoft BizTalk, Azure AppFabric and Azure Service Bus. Now is an important time to consider alternative integration platforms that embrace core Microsoft development technologies.

For Microsoft .NET development shops, IBM WebSphere Message Broker provides a number of compelling features that will allow them to quickly integrate systems faster, easier, and more reliably than ever before. Organizations with a mixed environment can greatly benefit from the broad range of features provided by Message Broker and can be used to augment an existing BizTalk infrastructure.

Introduction

At a minimum, an integration solution or ESB must support mediation, routing, transformation, and enrichment. These four functions are essential as they allow data to efficiently and effectively be transported between various systems. In addition to these four primary functions, an integration solution in a .NET shop needs to support the following additional capabilities:

- Provide and consume web services
- Provide and consume Windows Communication Foundation (WCF) services
- Support Microsoft SQL Server®
- Support Microsoft Active Directory®
- Support Microsoft .NET languages
- Support Microsoft Server Cluster
- Support MSMQ

This paper looks at IBM WebSphere Message Broker version 8 in a Microsoft shop from the perspective of development and IT operations, with the primary audience being development managers and architects in mid-sized and large enterprises. The focus is on capabilities of Message Broker V8, which Summa believes to be relevant for .NET shops considering an Enterprise Service Bus (ESB).

In larger companies, Message Broker and BizTalk are used to satisfy very different use cases. However, in many smaller and mid-sized companies, with limited resources, management must make a choice between hand-coding integrations, using BizTalk, or implementing Message Broker. For these customers a high-level comparison between BizTalk and Message Broker is appropriate.

This paper is based on Summa's experience in implementing diverse integration solutions over the past 16 years on a variety of platforms, including recent experiences implementing Message Broker 8 in Microsoft .NET shops. The key differentiators that Summa sees between Message Broker and other integration solutions for a .NET environment are:

- The .NET Compute Node
- Built-in Microsoft Windows Communication Foundation (WCF) Channel
- Integration with Microsoft SQL Server
- Record and Replay Functionality
- Security
- Cross-Platform Support and Enterprise Information System Support
- Licensing Options

The paper assumes that readers are familiar with the Microsoft development platform as well as high-level Integration and ESB concepts such as mediation, transformation, and enrichment. Readers who would like to learn more about these various topics should visit Summa's website at www.summa-tech.com.

Features and Functionality

IBM WebSphere Message Broker 8 was released in December 2011. One of the primary goals of this major release was to target the Microsoft .NET developer community. Prior to this release, many architects would not have considered IBM WebSphere Message Broker as a candidate integration platform for their .NET shop (though it did already include cross-platform support for WCF web services and WCF Channel).

Message Broker enables developers to write integration logic in a variety of languages including Java and PHP. Additional methods for transforming data are available via graphical mapping and XSLT. With the version 8 release, developers are now free to author transformations or custom logic using any of the .NET languages including C# and Visual Basic.

Message Broker uses the concept of message flows to develop integrations. Messages enter from one or more sources, are transformed and enriched, and ultimately sent to one or more output destinations. A wide variety of protocols, message sources, and destinations are supported including raw HTTP, web services, IBM WebSphere MQ, Java Messaging Services (JMS), FTP, files, CICS, TCP/IP, databases, and enterprise information systems (EIS) like SAP®. A host of data structures are supported including Binary, COBOL, XML, CSV, JSON, SAP IDOCs, various other user-defined structures, and industry standard formats such as EDI and HL7.

With this broad range of support it is possible for organizations of all sizes to meet their integration needs. Whether the system they are connected to is running on Windows, Linux, AS/400, the mainframe, or on a mobile device developers are able to connect diverse types of systems together; it does not matter if those systems are written in .NET, Java, or COBOL.

Message flows are developed using the IBM WebSphere Message Broker Toolkit. The toolkit is an IDE that allows developers to graphically develop message flows by dropping nodes onto a canvas. Each node exposes various properties that can be set to change the behavior of the node. Nodes are connected together to create a progression of data through the flow. Some nodes are used to designate input, apply content-based routing rules, transform or manipulate data, or write custom code.

The nodes provided out of the box are very robust and allow developers to implement a great deal of functionality with little to no coding. For example, using a Route node, a developer can easily route a message to a particular destination based on message content. Meanwhile, the Timer nodes can be used to trigger a flow at specific intervals.

In many cases, all a developer needs is basic familiarity with XPath notation in order to implement many message flows. However, when it is necessary to write code developers have several options available. Prior to version 8, the choices were ESQL, Java, or PHP. With the arrival of version 8, developers now have the ability to write code in any of the .NET languages.

At runtime, Message Broker message flows are deployed to one or more execution groups that listen for messages to arrive. Once a message arrives it is processed and sent to the destination system.

.NET Compute Node

The .NET compute node, new to version 8, is a node that can be dropped onto a flow's canvas at design time. When a developer double-clicks the node, they are taken to Visual Studio where they can implement custom transformation logic in one of the .NET languages (C#, VB.NET, etc.). The developer has access to any classes available in the .NET framework.

For example, a custom algorithm can be implemented, elements can be added or deleted from a message, custom security checks can be performed. Because the full power of the .NET framework is available (just don't use GUI elements), you can easily integrate with a variety of existing Microsoft solutions or .NET applications like Microsoft Dynamics, Microsoft Sharepoint, Microsoft SQL Server, MongoDB, or Log4net.

The following is a snipped of code that can be implemented in a .NET compute code using the familiar .NET array syntax:

```
NBElement amount = root["XMLNSC"]["Money"]["Amount"];
```

Or, for developers who prefer to use LINQ:

```
NBElement x = InputMessage.RootElement["XMLNSC"]["Top"];
var list = x.Where(t => t.Name == "Money" &&
     t["Currency"].ToString() == "USD");
foreach (NBElement element in list)
{
     // Process each element in turn
}
```

Developers have access to all of the features of Visual Studio. This includes the debugger. When debugging a flow written in WebSphere Message Broker 8 flow, which contains a .NET compute node, the flow will automatically launch Visual Studio in debugging mode. The developer can then step through the code line-by-line using the Visual Studio debugger; HotSwap functionality is also supported.

At runtime, the .NET common-language runtime (CLR) is hosted inside of the WebSphere Message Broker execution group. Each CLR has one or more dedicated application domains and these can be used to isolate or to share data between different .NET compute nodes. Classes are references by name and namespace from the Message Broker flows. Custom DLLs are stored in a working directory and do not need to be registered in the GAC (Global Assembly Cache).

By providing the ability to write code in a familiar language, IBM has shortened the learning curve to adopting WebSphere Message Broker.

WCF

Microsoft Windows Communication Foundation (WCF) provides a unified model for building service-oriented applications. It provides a technology-agnostic approach for integrating systems using a variety of protocols, including REST-based web services, SOAP-based web services, and asynchronous messaging systems.

IBM Message Broker supports web services via HTTP and HTTPS. It also includes a WebSphere MQ (WMQ) channel client for WCF for the .NET developer to exploit Message Broker's built-in MQ for very high performance messaging. Using the MQ WCF channel, developers can leverage the WCF programming model in their client applications. This support is critical because it allows developers to be insulated from working directly with MQ APIs. Instead, developers write and develop their client code in a normal manner, and then use configuration files to wire their WCF consumer to the Message Broker-based service.

Previously, developers would have to communicate with WMQ using a proprietary API or some form of bridging technology. The WCF channel supports .NET frameworks v3.0 and above. In order to use this functionality, a developer simply has to make sure the WMQ WCF channel's DLL is available to the project and build process. The DLL can either be deployed via the

application or registered in the GAC. Configuration only requires a connection string to be entered in the WCF configuration section of the `web.config` file.

The ability to support WCF via WMQ makes cross-platform support simple because WMQ clients exist for all major platforms including, Windows, Linux, and iOS. The WCF channel can be used in any custom .NET application. This includes standalone applications, windows services, and self-hosted applications (process activations is not currently supported). This flexibility also means that .NET applications can benefit from the reliability, throughput, and extended features of WMQ and Message Broker, such as SSL, the WMQ infrastructure (poison message handling, dead letter queues, pub/sub functionality), and security.

Security

Security is a critical component of any system. By default, WebSphere Message Broker will delegate to the security mechanisms used by the underlying transport layer. For example, if using IBM WebSphere MQ, WMQ security policies will be used. Message Broker provides the ability to override this behavior and use its own security manager. This security manager can secure the transport layer, but also individual messages based on identity.

The broker security manager can be used to authenticate the identity of the message, use trust assertions, and verify authorization at the message flow level. Identify mapping is also supported. The security manager can delegate to a third-party security provider such as LDAP (including Active Directory) or WS-Trust v1.3 compliant servers. This includes Tivoli Federated Identity Management, and Microsoft Active Directory.

In addition to invoking security at the flow level via one of the input nodes, a SecurityPEP (Security Policy Enforcement Point) node can be used to verify security within the flow itself.

In addition to authentication and authorization, Message Broker provides facilities for ensuring message integrity and confidentiality. All of this can be managed through the security manager's profiles and policy sets.

Lastly, broker also provides support for auditing via monitoring and record-and-reply functionality. Monitoring can be enabled at the message flow level to support business-level transaction auditing. When monitoring is enabled, message flows can be configured to emit events in a variety of circumstances. This information can be used for auditing transactions, or depending on the amount of information captured, replay transactions. This information can also be integrated into IBM WebSphere Business Monitor to provided enhanced monitoring capabilities.

Cross Platform and Enterprise Applications

Many .NET shops are in fact hybrid-shops. They have solutions running on a variety of platforms besides the Windows platform. For example, they may have an ERP deployed on Linux, a warehouse management system running on the AS/400 platform, and a third-party application running on a JEE application server. Message Broker has the ability to connect to a variety of platforms via its built-in connectors and application adapters.

Out-of-the-box, there are EIS (Enterprise Information System) connectors built-in for a variety of products including PeopleSoft, SAP, Siebel, JD Edwards, and others. As of v8 fix pack 1 (aka v8.0.0.1) these EIS adapters are built-in and free for use in Message Broker Standard and Advanced.[1]

[1] http://www-01.ibm.com/common/ssi/cgi-bin/ssialias?infotype=AN&subtype=CA&htmlfid=897/ENUS212-233

Developers also have the ability to create custom nodes to support other systems and create reusable libraries. Additional adapters are available to Message Broker customers through SupportPacs and include many specialized features that enhance the native capabilities and can be submitted to the MQ/Broker community at large.[2]

Licensing Options

Prior to version 7 of Message Broker, the only option available was an all-inclusive advanced package targeted towards enterprise customers. While a very capable offering, it was heavyweight and expensive. In version 7, a Starter Edition at a price point more suitable to the mid-market in addition to the traditional "all-in" edition. The Starter Edition provided the same functionality as the full edition, but with a limitation on number of execution groups and number of flows that could be deployed within an execution group.

With version 8 of Message Broker, the Starter Edition has been replaced with the Standard Edition, and the flow limit has been removed. A new Express Edition has been added which is a great entry-level offering and is suitable for many mid-market or departmental applications. The current line-up of Message Broker now consists of the following editions:

- WebSphere Message Broker Express
- WebSphere Message Broker Standard
- WebSphere Message Broker Advanced

The addition of an Express offering and the licensing changes for Standard are welcome changes, and will be attractive to smaller companies, departmental solutions and ISVs. The Standard edition will be the edition of choice for many mid-market customers while the Advanced edition will be the edition of choice for corporate shared IT infrastructure environments.

Express and Standard support a single runtime of the integration engine process or "execution group" per operating system instance (i.e. per machine or VM). With hot failover support and and optimized support for 64-bit Windows, this will provide solid performance and reliability at an economical price point. The Advanced edition enables an arbitrary number of "execution groups" per operating system instance.

Since version 6 of Message Broker, shared development environments are provided free of charge in addition to no-charge workstation installations. This means that only system testing, performance testing, and production environments must be licensed. Each edition of Message Broker supports virtualized and subcapcity license models; this means that organizations can right-size their environment from the start and incrementally add capacity as load increases. Additionally, IBM WebSphere MQ is bundled into the license cost for Message Broker.

[2] http://www-01.ibm.com/support/docview.wss?uid=swg27007205

Comparisons to Microsoft BizTalk

As mentioned earlier, in larger companies, Message Broker and BizTalk are used to satisfy very different use cases. However, in many smaller and mid-sized companies, with limited resources, management must make a choice between hand-coding integrations, using BizTalk, or implementing Message Broker. For these customers a high-level comparison between BizTalk and Message Broker is appropriate.

In many ways, Message Broker 8 is easier to use than Microsoft BizTalk beginning with the installation process, continuing through development, and all the way through deployment. Lastly, we talk briefly about high-level performance differences.

Developer Installation

Installation is a very different experience with both products.

Steps for installing Message Broker 8:

1. Install Microsoft Visual Studio
2. Install IBM WebSphere Message Broker (which also installs IBM MQ)
3. Configure IBM WebSphere Message Broker

Steps for installing BizTalk 2010 include:[3]

1. Install Microsoft Excel
2. Install Microsoft Visual Studio
3. Install Microsoft SQL Server 2008 R2
4. Configure SQL Server 2008 R2
5. Install Microsoft BizTalk 2010
6. Configure BizTalk 2010

From start-to-finish the time required to be up and running with a basic operating system installation is 1 hour for Message Broker 8 and over 5 hours for BizTalk 2010.[4]

Development Process

Message Broker provides a rich set of functionality out of the box that allows many integrations to be developed with a minimal amount of code. This includes the support for enterprise information systems such as Siebel®, PeopleSoft®, and SAP. At a lower level, it also provides support for numerous integration patterns and mechanisms including publish/subscribe support with no extra coding.

As mentioned previously, the debugging capabilities built into Message Broker 8 allow developers to author .NET compute nodes in Visual Studio 2010. The integration is seamless and focused on supporting a development process that leverages the best features of IBM WebSphere Message Broker Toolkit and Microsoft Visual Studio 2010.

For developers familiar with Microsoft BizTalk they will be familiar with the disjointed experience of using BizTalk and Visual Studio. When using BizTalk you cannot code directly in Visual Basic or C#, but must instead wring in XLANG/s; a subset of C#. The editor presented to developers is a little more than a rudimentary text editor without support for modern productivity aides such as code highlighting and formatting. Intellisense capabilities are greatly limited.

---

[3] This is a summarized list of the full installation details found at http://www.microsoft.com/en-us/download/details.aspx?id=11503.

[4] This includes several cycles through Microsoft Windows Update.

BizTalk provides primitive debugging support at the block level, but not at the line-level. Message Broker has real interactive debugging; the kind that all developers have come to expect from modern IDEs. For instance, breakpoints at the diagram and code level (including C# or VB). It also allows single stepping through code and variable inspection. With Message Broker, saved test cases can be easily re-executed.

With BizTalk you have to deploy to the server. Then you have to manually generate a test message. There is no interactive debugger, but merely a trace viewer that can breakpoint at major steps of the flow. There is no way to even look at the code execution, let alone debug it. So debugging a BizTalk solution will take developer much longer to debug a problem.

Packaging and Deployment

Packaging and deployment support in BizTalk requires several manual steps to be performed. For instance, a developer must go through a deployment, binding, and enlisting step. Deployments also require a number of moving parts to be properly configured before an orchestration can be started. When using Message Broker 8, the deployment process is configured by creating a BAR file (broker archive file) and deploying directly from the Toolkit to the runtime server. Custom properties can also be defined to indicate machine names, queue names, or URLs.

Performance Differences

As part of the analysis, three types of performance tests were executed:

- Batch file processing
- Queue to queue communications
- Web service request/response

The goal of the testing was not a stress, or failure, test. Instead it was focused on identifying the out-of-the-box differences between the two products. The rationale for this type of performance testing is simple – IT teams are busy, and many smaller organizations don't have the specialized skills nor time for exhaustive performance tuning.

In the test scenarios, the tests were run on a Microsoft Windows Server 2008 R2 virtual machine configured on an isolated hypervisor virtual machine (i.e. no other virtual machines were running during the test). The servers were configured with two virtual CPUs and 8GB of memory. The two key metrics of interest were throughput and response times. CPU utilization and I/O wait times were also monitored during the tests. More detailed information regarding the test configuration and systems can be found at the end of this paper.

The following table shows the results of the various tests executed:

Test	Response Time		Throughput	
	Broker	BizTalk	Broker	BizTalk
Web Service	20 ms	131 ms	450 tps	7 tps
Queuing	< 20 ms	5981 ms	450 tps	7.14 tps
Batch File	45 seconds	305 seconds	1,770 rps[5]	261 rps

[5] RPS stands for records per second.

Average CPU utilization for Message Broker was under 50% while BizTalk was seeing CPU utilization under 80%. IO wait times were very low overall.

The file-based test was using a pipe-delimited 7.5MB file containing 79,651 records. Both the queue-based tests and the web service tests were designed to extract a handful of fields from a 100KB purchase order in an XML format. The web service tests were used by direct invocation of services hosted on Message Broker and BizTalk.

For the queue-based tests, MQ Series was used for the Message Broker testing, and MSMQ was used for the BizTalk testing. It is interesting to note, that the results for Message Broker and BizTalk are achieving similar results. Real world experience has often shown far better performance with queuing over web services. The load generator is a likely suspect in this case.

The only adjustment made to the Message Broker environment was changing the number of flow instances in the execution group from the default of 1 to 10..

As the table above shows, out-of-the-box performance for Message Broker on Windows in a .NET environment is quite impressive and required no tuning to achieve this level of performance.

Conclusion

Version 8 of IBM WebSphere Message Broker has a compelling set of capabilities and user experience that should interest architects and developers in any Microsoft .NET shop who need a robust, scalable, and high-performance integration solution. With a variety of licensing options and editions available, smaller organizations can leverage enterprise-class capabilities without the enterprise price tag. Meanwhile, enterprise customers can augment existing capabilities with secure and scalable messaging without having to abandon familiar development tools. With the tight integration to Microsoft Visual Studio and high performance integration with built-in .NET support, Microsoft developers can now deploy robust high performance integrations using tools that are comfortable and familiar.

About this Study

The study was commissioned by IBM and performed by Summa Technologies. All findings of this study are fully backed by the solid reputation of Summa. Since 1996, Summa has been architecting and implementing commercial-grade applications for organizations of all sizes, including Fortune 100 companies. Summa's IT consulting services help companies evaluate and implement modernization strategies for multi-tier distributed systems. Summa helps their customers make the right technology investments by first assessing the current environment, and then recommending and implementing the technology solutions that improve customer satisfaction, employee productivity and partner relationships. Authors of this report each have more than seven years of experience building and supporting large-scale systems.

About the Authors

Jason Armstrong is Practice Director for Business Transformation Solutions at Summa. Jason's experience with Microsoft Technologies dates back to the Visual Basic 4.0 days. He has also architected and authored solutions on both the Microsoft .NET and Java platforms. Jason started using IBM WebSphere Message Broker when it was still known as MQSI. In his current role, he works with customers to develop and implement their integration, BPM, and decision management strategies.

Glen Wirth is a Senior Integration Specialist at Summa. He has worked with numerous integration products including Microsoft BizTalk 2010, IBM Cast Iron, IBM WebSphere Message Broker, and IBM WebSphere Enterprise Service Bus. He has experience implementing integrations in a variety of industries including manufacturing, financial services, and healthcare.

Adrian Wright is a Senior Technical Consultant at Summa. He has worked primarily as a Microsoft .NET developer and solutions architect, and has experience in a variety of industries including retail, healthcare,and defense. In the integrations space, he has worked with IBM WebSphere Message Broker, Websphere MQ and MSMQ/WCF.

References

The following is a list of references used in the writing of this study:

- Enterprise Service Bus - http://en.wikipedia.org/wiki/Enterprise_service_bus
- Microsoft Developer Network (MSDN) - http://msdn.microsoft.com/en-US/
- IBM WebSphere Message Broker - http://www.ibm.com/software/integration/messagebrokerproductline/
- IBM WebSphere Message Broker 8 Info Center - http://publib.boulder.ibm.com/infocenter/wmbhelp/8r0m0/index.jsp
- Summa Integration Solutions - http://www.summa-tech.com/soaAndIntegration.php
- Hohpe, Gregor and Bobby Woolf, **Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions**. ISBN-13: 978-0321200686. Addison-Wesley Professional. 2003. http://www.amazon.com/Enterprise-Integration-Patterns-Designing-Deploying/dp/0321200683

Summa's Message Broker 8 Wish List

Experienced craftspeople of various trades will often complain about the tools they use; Summa's integration specialists are no different. The following is a wish list of enhancements that we would like to see incorporated into Message Broker (we are happy to report that with fix pack 1 of Message Broker v8, several of these wishes have been granted):[6]

- We would like to see support for SSL and XA transactions via MQ's WCF channel client using managed connections. Currently, the only option for handling SSL and XA transactions is via unmanaged connections. In our experience, the unmanaged connections are fairly reliable, but we do prefer to leverage managed code instead of unmanaged code whenever possible.
- Strongly-typed object models for .NET and Java compute nodes. We have mimicked this functionality using tools like JAXB, but it would be nice to have this as a native feature. This would make traversing message structures more elegant. –***As of v8.0.0.1 this is now supported.***
- Another native feature we would like to see is a "less code" approach to caching. Today this is simulated using static methods in either .NET or Java which wrap hash maps. While functional, it does require some additional coding to properly manage the cache. ***– As of fix pack 1 this is now supported for the Java Node with plans to add to .NET node.***

[6] The full list of fixes in v8.0.0.1 can be found at http://www.ibm.com/support/docview.wss?rs=849&uid=swg24033244

Performance Test Environment

The test environment used for performance testing consisted of a server and a load generator. The load generator was a developer-grade laptop. The server environment was created using two Microsoft Windows Server 2008 R2 virtual machines. The operating system installation, configuration, and patch levels were identical.

The WebSphere Message Broker v8 infrastructure was configured on one virtual machine while BizTalk 2010 was configured on another. Microsoft SQL Server 2008 was installed on a separate instance.

The message flows and orchestrations were developed to perform the same tasks for each of the three tests (batch file, queuing, and web services) on the two platforms.

All extraneous virtual machines were shut down on the host server during the testing. For example, the BizTalk 2010 server was not running while the Message Broker tests were executing, and vice versa.

Legal and Trademarks