On demand business solutions
White paper

WebSphere® software

Understanding Web services in an enterprise service bus environment.

By Stephen Todd, senior technical staff member,
WebSphere MQ Strategy, IBM Hursley Park laboratory

March 2005

Contents

2 Introduction
2 What is an ESB?
4 What should an ESB deliver?
5 An ESB based on IBM products
6 What are Web services?
9 Web services and ESBs
10 Web services at work in an ESB context
12 From a Web-services point of view, what advantages does an ESB offer?
14 How can applications running over an ESB exploit Web services?
14 Possible Web services and ESB scenarios
17 Conclusion
19 For more information

Introduction

This white paper explores Web-services considerations when operating an enterprise service bus (ESB). It explains what an ESB is (in both conceptual and practical terms), what Web services are and what Web services bring to an ESB. It also includes typical Web services and ESB scenarios to help illustrate these points.

What is an ESB?

An ESB is an architectural pattern that enables you to optimize the distribution of information between different types of applications across multiple locations. The ESB pattern is founded on and unifies message-oriented, event-driven and service-oriented approaches to integration. The core characteristics of an ESB (all of which should be oriented toward a service-based infrastructure) provide:

- *Standards-based application integration*
- *Support for Web services, message-based transport and publish-and-subscribe (event-based) integration*
- *Transformation*
- *Intelligent routing*

At a practical level, the term ESB describes a collection of distributed integration servers (sometimes referred to as *hubs*) that work together seamlessly to provide a variety of complementary services designed to integrate applications that could not previously work together in flexible and easily changeable ways. A conceptual view of the ESB is shown in Figure 1.

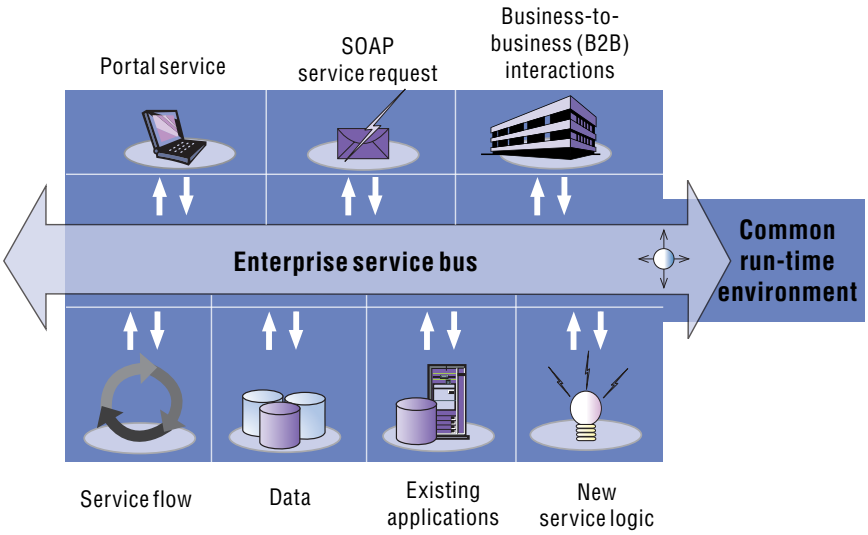

Figure 1. A conceptual view of an ESB

In this context, it is important to understand the following points about ESBs:

- *The term* enterprise *does not necessarily have to encompass the whole of an organization. One of the attractions of an ESB is that you can start small (perhaps with only two to three physical instances) and expand to fit evolving business circumstances.*
- *The term* bus *is used to convey the notion of information being carried between originators and receivers, using different communications models and data formats, to many different destinations. The bus provides a common backbone through which applications can interoperate.*
- *There is no practical or meaningful difference between a bus and a collection of hubs. The difference between a bus and a collection of hubs lies only in the way each topology is drawn. In other words, a collection of hubs can be a bus and a bus can be shown as a hub.*
- *An ESB should possess some degree of programming intelligence (for example, to determine routing or persistence, or to implement rules or content processing).*
- *The term* ESB *is conceptual. It describes what, in practice, is typically delivered through two or more instances of software. You do not buy an* ESB. *What you purchase is the software for each node of an ESB. The combination of many software instances working together is what delivers the* ESB *concept.*

An important aspect of the ESB concept is that applications (and their developers) do not need to be concerned about where services or permanent resources (like applications, databases and queues) are in an infrastructure. This capability means an ESB should make it easier (and, ideally, transparent) to connect to these services and resources – and, in so doing, to reduce the effort required to integrate heterogeneous environments. An ESB introduces new options for interoperation and helps enable information to flow to the people who need it, when they need it. In this way, an ESB can improve the responsiveness and accuracy of decision making, and materially assist in the delivery of on demand computing.

What should an ESB deliver?

Another way to look at an ESB is to consider what it should deliver. Ideally, an ESB should describe (at a high, logical level) what is going on within an enterprise, or subset of an enterprise. This description should be visual or graphical so that both the available resources and the flows (from sources to destinations) can be represented and depicted. For example, these ESB capabilities have been available in IBM WebSphere® Business Integration Message Broker for some years (see Figure 2).

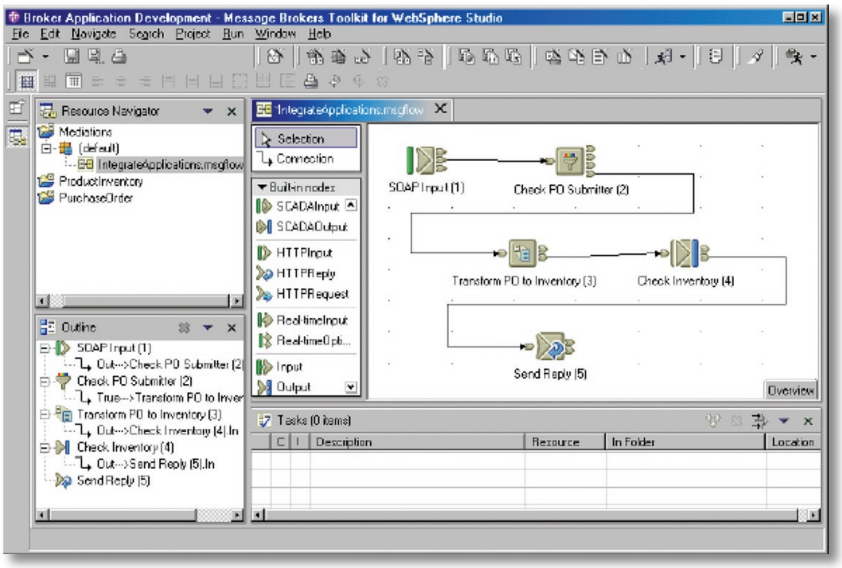

Figure 2. Graphical descriptions in an ESB in IBM WebSphere Business Integration Message Broker

This graphical concept can be extended. Besides describing the flows between the various nodes (sources, destinations, databases, local ESB processing and so on), an ESB should enable you to lay on top of these resources the particular constraints associated with how flows are actually deployed and used. Indeed, the more an ESB can describe, the more capabilities can be automated. If applicable resources are described, network bandwidths set, systems located and transaction boundaries defined, you can then determine where potential problems might occur—and how these problem areas might be avoided or resolved. There should be a clear distinction between what is to be achieved, and how it is to be achieved (from a high, logical level).

In one sense, this approach envisions applications as existing on the edge of an ESB. The first step you should take is to integrate each application with an ESB. Having done this for several applications (and you should only need to do this once for each application, instead of the many times necessary in traditional peer-to-peer connections), you can now create *new, logical combinations of applications* out of the definitions of existing ones. An ESB's flow capability effectively enables new combinations to be derived and implemented without opening up or rewriting the source or destination applications themselves. It is the intermediary logic on the ESB that enables such new combinations.

Thus, in an ideal ESB environment, you should be able to define the applications and flows that shape your organization. You should also be able to add new combinations of applications and flows—or modify existing ones—easily.

An ESB based on IBM products

An ESB can be implemented today using existing IBM products. IBM WebSphere MQ can provide the base for an ESB with point-to-point messaging, some publish-and-subscribe capability and clustering function. IBM WebSphere MQ Everyplace® software complements WebSphere MQ with reduced-footprint message handling for small devices like personal digital assistants (PDAs). Higher in the ESB function stack, IBM WebSphere Business Integration Event Broker and IBM WebSphere Business Integration Message Broker provide value-added services, such as routing, transformations and rules-definition handling, as well as fan-in and fan-out capabilities.

If you need tight links to the Java™ development and run-time environments, IBM WebSphere Application Server can provide these links, as well as advanced Web-services capabilities.

What are Web services?

Web services are standards that enable disparate systems to communicate by providing well-defined interfaces. An ESB provides an infrastructure that enables varied and distributed applications and systems to exchange information. Combining Web services standards with an ESB can potentially deliver the broadest connectivity between systems. An ESB supports Web services by enabling a flexible mix and match of the new aspects of Web services with more-established application-integration techniques—enabling the power of the new to be combined with the reach of the old (see Figure 3).

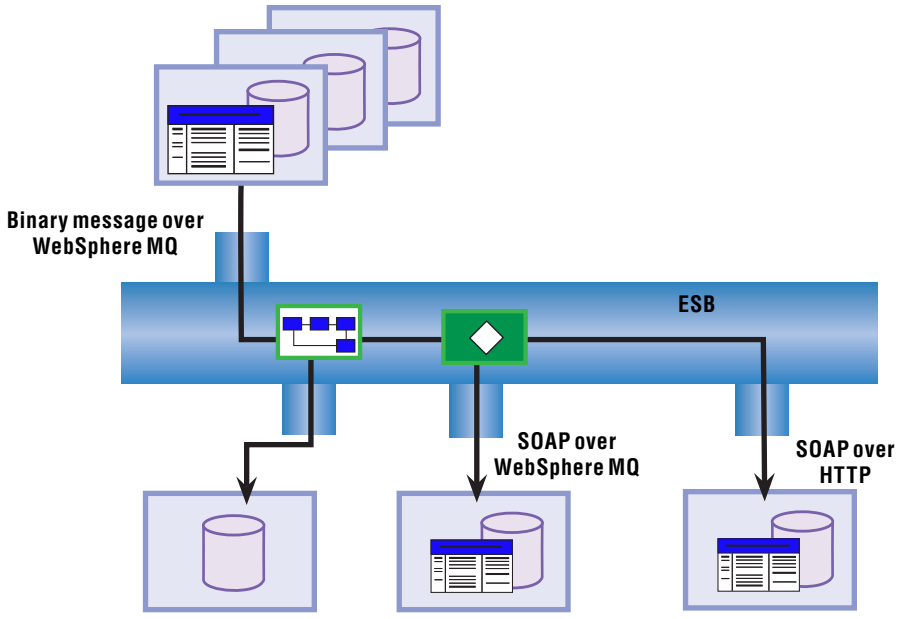

Figure 3. An ESB that supports Web services

In this context, Web services are a collection of open standards – developed and endorsed by many companies in the IT industry, including IBM, Microsoft, Oracle and others – that permit interoperation between heterogeneous infrastructures (see Figure 4). The basic standards include:

- *Simple Object Access Protocol (SOAP), an XML-based standard for one-way and request-response messaging.*
- *Web Services Description Language (WSDL), an XML-based standard for storing service definitions, including details of what format or signature a particular service provides, how that service is to be invoked and where it is located.*
- *Universal Description, Discovery and Integration (UDDI), a standard for repositories of information about services (designed to assist people and applications to find Web services).*

A number of other, complementary Web-services standards cover other aspects of IT – including transactions, security, reliability and so on – and are shown in Figure 4.

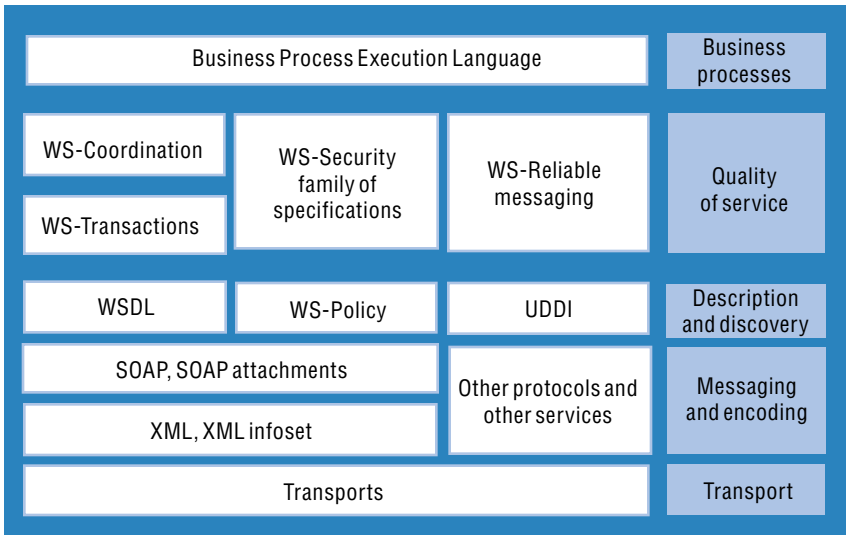

Figure 4. Selected Web-services standards

Web services use SOAP (a transport-neutral protocol) to exchange structured data. SOAP consists of three parts:

- *An envelope that defines a framework for describing what is in a message and how to process the message*
- *A set of encoding rules for expressing instances of application-defined data types*
- *A convention for representing remote procedure calls and responses*

SOAP is designed to be used with a variety of transport protocols to deliver information with metadata (descriptions of the structure and semantics in a message) between two participants. In this way, SOAP helps programs running in truly heterogeneous environments to interoperate. For example, a Microsoft® .NET application and WebSphere Application Server can consume each other's hosted Web services.

To some developers, SOAP is not attractive because it can seem like a heavyweight format. Its need for parsers and serializers can be cumbersome, and it dramatically increases object sizes compared to binary data. Many users need a binary-equivalent standard to SOAP. Yet to others, SOAP is attractive because it is actually a lightweight format and protocol for exchanging information in a decentralized, distributed environment. Its attractions are enhanced by its use of XML.

The default transport mechanism for SOAP messages is HTTP. However, SOAP can be used in combination with a variety of transport protocols, such as Simple Mail Transfer Protocol (SMTP), File Transfer Protocol (FTP) and so on. HTTP has its weaknesses, including its lack of guaranteed and reliable (once-and-once-only) delivery, as well as the need for the back-end service to be available online for any request to be received. By substituting HTTP with a different message-delivery option, such as SOAP over WebSphere MQ (see Figure 5), you can provide a transport for SOAP messages that possesses excellent security, reliability and asynchronicity (store-and-forward) delivery.

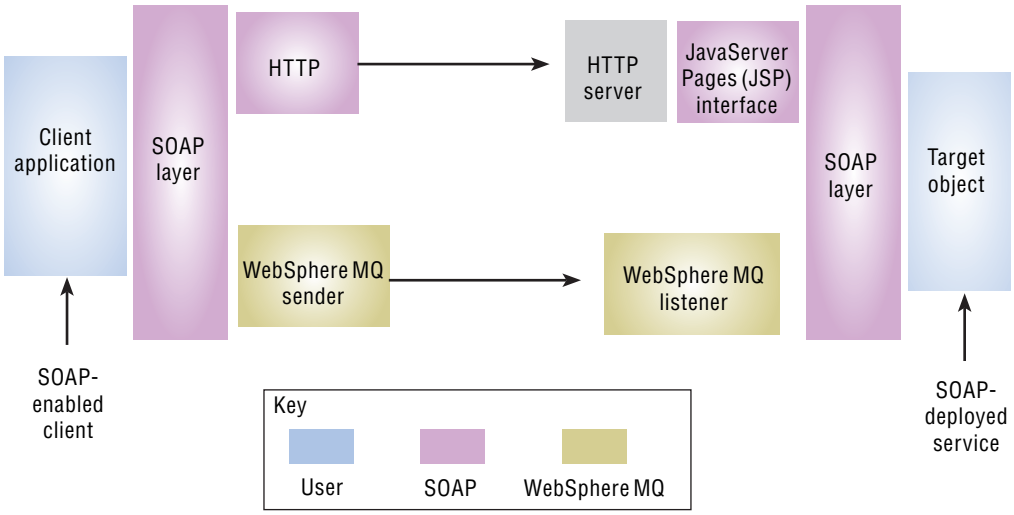

Figure 5. SOAP over WebSphere MQ and SOAP over HTTP

Web services and ESBs

One way to understand where Web services fit into an ESB environment is to consider WebSphere MQ. At its simplest, WebSphere MQ knows about moving data from one place to another. But it does not know anything about the format of the data that is carried in its messages. Fundamentally, early Web-services standards, especially SOAP, were principally concerned with the format of what was being carried (and less about the mechanism of carriage). Web services were essentially remote procedure calls (RPCs) or one-way call formats. These services did not indicate how the data would be carried to the other end. You could assume that it would be by HTTP, but there was no discussion about different kinds of quality of service or other attributes that might be applicable.

The presumption was that any connection would be synchronous and that any asynchronous properties would have to be built, as required, on top of the synchronous connection by using a work-around. Therefore, you can see an obvious marriage between:

- *WebSphere MQ—to provide reliable, once-only assured transport.*
- *Web services—to look after data content and formats.*

This situation fostered the genesis of the SOAP over WebSphere MQ implementation. Through this implementation, you can benefit from improved ease of use at the development stage (through the extensive Web-services development tools). And you can have confidence in middleware delivery because of the proven capabilities of WebSphere MQ. The result is that you can leverage your entire infrastructure—from Eclipse to Microsoft .NET and existing WebSphere MQ networks—far more effectively.

What was, and continues to be, even more attractive about the marriage of WebSphere MQ and Web services is that Web-services experts need to know little about the WebSphere MQ infrastructure, and WebSphere MQ owners need to know equally as little about what is being transported in the SOAP message that is carried over WebSphere MQ. An added attraction comes with WebSphere MQ clustering capabilities. Using WebSphere MQ clustering offers an excellent way to autoconfigure WebSphere MQ—and to provide self-sustaining, self-recovering workload balancing and even autonomic properties on a significant scale.

Web services at work in an ESB context

As earlier described, a Web service is something very limited. It is a well-defined set of formats and interfaces. The basic infrastructure for these formats and interfaces is defined by SOAP and WSDL. The details for a particular service are defined by the person who writes that Web service, often just by writing lines of code that embrace XML running over SOAP, so the developer can say, "This is a Web service and it is now available."

Web services become interesting at several levels—particularly when Web services start to interact with each other. However, you must remember that not every application is written as a Web service. Valuable legacy applications will almost certainly be in use. Product extensions like IBM CICS® Web Services Bridge can help in these situations by enabling Web services and legacy applications to connect. These situations are where the ESB concept can come into focus. An ESB offers a means by which modern

Web-services applications and valuable legacy applications can make the most of each other—and your developers don't have to create new applications to carry out the functions of the familiar ones that couldn't be connected. In essence, an ESB facilitates the combining of the old and the new.

How does this work? In the case described in this white paper, the first action is to decide to use CICS over Web services. The existing CICS applications have known interfaces that are already defined—the inputs and outputs that the CICS application requires. From this data, the necessary paths can be generated—in particular the WSDL code that tells people in the outside world what the application needs. CICS Web Services Bridge (see Figure 6) generates the entry and exit doors into and out of the CICS application.

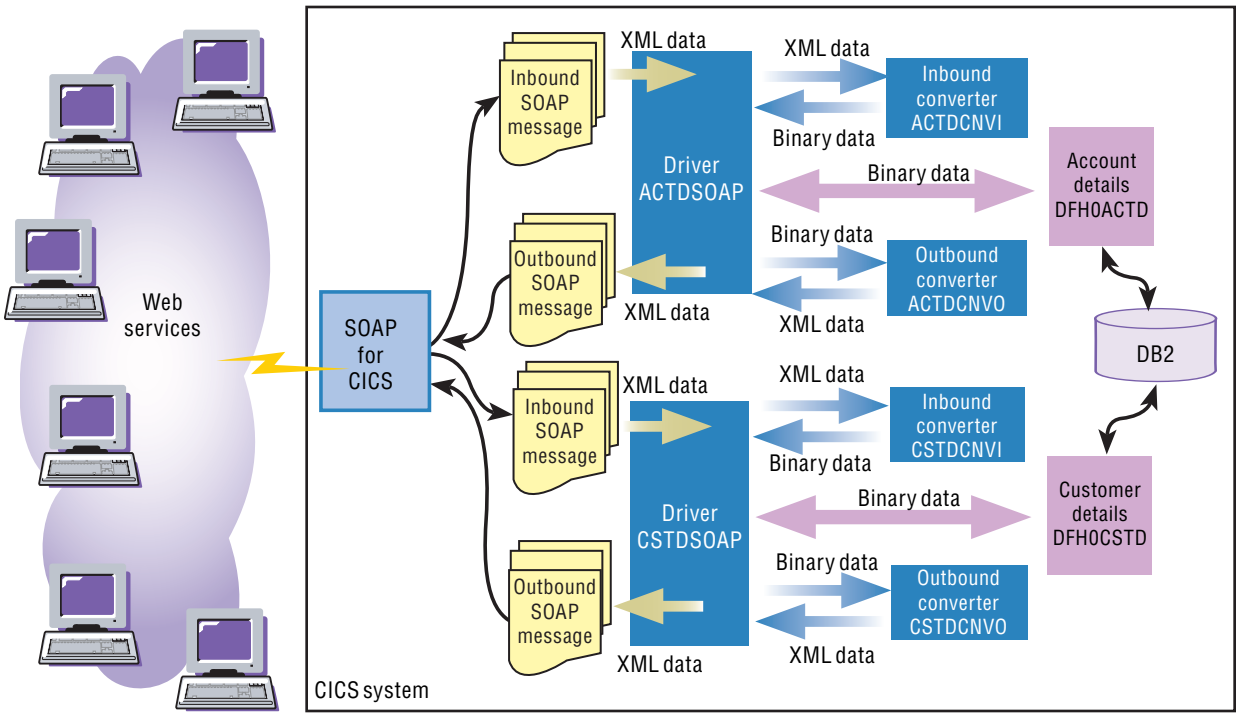

Figure 6. CICS Web Services Bridge

The second set of actions involves the client on the other end (that you want to connect) and the development tools you use to support Web-services application creation, whether with .NET, Eclipse or some other development tooling. You specify to the tooling function what Web service should be targeted (the CICS application) and where the relevant WSDL definitions can be found (probably, but not necessarily, using UDDI). The code at the client places the relevant call. The connection between a new-style application and a legacy-style CICS application has been implemented. Old and new applications have been integrated—using Web services. The client itself can be a user interface (UI)—like one provided through a Web browser—or it can be an application with no overt UI. The choice depends on the circumstances, but either is possible.

What an ESB adds is a common intermediary point (or points) where the rigidities of individual point-to-point connections can be avoided. In this instance, an ESB is effectively a switch for connecting multiple Web-services applications and legacy applications—or even other resources, like databases or application servers.

From a Web-services point of view, what advantages does an ESB offer?
As already explained, the default transport for Web services is HTTP. Although HTTP provides a specific transport mechanism that is relatively easy to set up and use, it does not provide industrial-strength reliability and management. An ESB adds the strengths of WebSphere MQ and WebSphere Business Integration Message Broker, for example, to a Web-services environment, providing assured message delivery and a managed infrastructure.

In a typical scenario, Web services-based applications can be deployed to exploit an ESB with little or no modification to the application. The consumer and provider of the Web service need to access only the ESB infrastructure. The changes required to enable the ESB to provide the Web-services transport are made at deployment setup. In this context, an ESB implementation based on WebSphere Business Integration Message Broker provides a Web-services

proxy for interoperation with existing (legacy) applications—because the ESB can act as an intermediary (see Figure 7). It decouples consumers from providers in ways that SOAP alone cannot (because SOAP is based on a tightly coupled model). By decoupling consumers and providers, consumers can take advantage of new services (or they can be shielded from changes to existing services) by exploiting the routing and other capabilities available in the ESB.

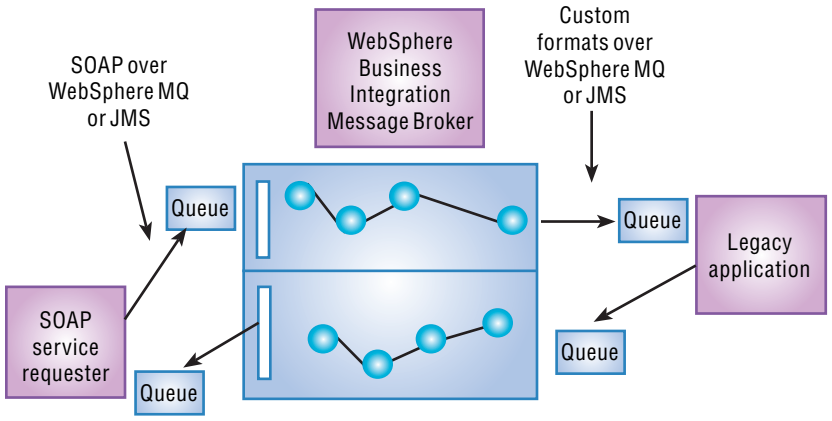

Figure 7. WebSphere Business Integration Message Broker as a Web-services proxy

A benefit of Web services in an ESB context is that they enable you to take an architectural approach to decomposing IT solutions into pieces that can be reused. This helps increase efficiency because it makes the delivery of agile information systems that are responsive to changing business demands easier. Similarly, loosely coupled, federated reuse of existing assets is made possible by combining the standardization inherent in Web services with the flexibility inherent in an ESB. The ESB offers the capability to support a wide range of devices, from PDAs and cell phones, to telemetry devices, to mainframes. The ESB can also support applications, from new Web services-based applications to legacy applications—and enables them to work together as peers in a distributed environment.

How can applications running over an ESB exploit Web services?

A developer can write new applications for an ESB using the Web-services development infrastructure provided by products like IBM WebSphere Studio Application Developer or Microsoft Visual Studio .Net. These tools:

- *Provide the means to define the SOAP message formats.*
- *Help you easily enable disparate consumers and providers of services to interact properly.*
- *Enable the application programmer to see the service as a standard method definition, and then call that service using a standard remote method call (the benefit being that the programmer does not need to be involved in details of format or transport).*

Where existing service applications are running on an ESB, the message-broker component can be configured to wrap these applications as Web services. In this situation, the ESB generates WSDL for the specific Web service and provides the run-time environment that hosts the wrapping. Alternatively, you can use an ESB to permit programs that do not understand Web services (most often, legacy applications) to call Web services. The standard features of an ESB, such as transformation, routing, warehousing and auditing, can be applied to all forms of messages, including Web-services messages.

Possible Web services and ESB scenarios

Combining Web services and the ESB concept is a powerful pairing. Many different possible scenarios exist. The following section briefly describes some of the more-common possibilities.

One of the simplest possibilities is to enable a .NET application to use a CICS service. The CICS workload is prepared as a Web service that is deployed on the ESB. The WSDL that defines this Web service generates a proxy on the ESB. The .NET application is then written to exploit the proxy on the ESB. Thus, the .NET application sees the CICS service as a simple call that is delivered by the ESB.

In a more-complex scenario, an IBM IMS™ transaction can be wrapped as a Web service using the message-broker component of an ESB (such as WebSphere Business Integration Message Broker). The broker capability converts incoming message requests, which are in SOAP format, into the COBOL Copybook format. These messages are then forwarded to the relevant IMS transaction. On the reverse route, the COBOL Copybook format response is converted back to SOAP format by the broker function on the ESB, and the results are returned to the calling application. The calling application is written in the same way as it was in the first CICS scenario. The broker can also be configured to identify high-value transactions (content processing). These transactions can simultaneously be converted to a different SOAP format and forwarded to a different application that is collecting details about high-value transactions for marketing purposes.

For Web-services development for an ESB, application developers can use development techniques in the same way as they would for HTTP Web services. The deployment process identifies the services that are to run over the ESB. This action generates the appropriate ESB-related infrastructure (for example, queue and channel definitions), as well as the associated Web-services elements (WSDL and any associated proxies).

Using the broker capabilities in an ESB in more-advanced scenarios requires setting up the appropriate message flows and defining format transformations. Much of the work of defining format transformation is provided automatically by the message repository manager (MRM) of WebSphere Business Integration Message Broker. The MRM can use both the WSDL definitions of the SOAP formats used in Web services and other formats that are used by more traditional (or legacy) applications (copybooks, electronic data interchange [EDI], WebSphere MQ messages and so on). The attraction of an ESB is that it can automate the processing of a COBOL Copybook definition and then generate a WSDL definition for a SOAP-equivalent format, along with the transformation needed to convert a SOAP message to a COBOL Copybook structure.

Another example of the use of Web services in an ESB environment is where an external Web service is invoked as part of the processing of a broker flow. This flow is illustrated in Figure 8.

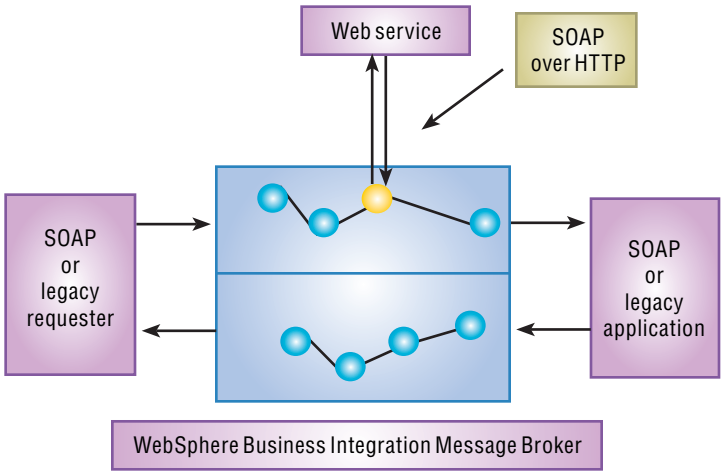

Figure 8. Accessing an external Web service as part of a broker flow

The scenario depicted in Figure 9 pulls more capabilities together to reflect a more-advanced ESB scenario. A basic service is made available to different kinds of Web-services clients. The various clients have different quality-of-service needs. Simple clients work well with SOAP over HTTP, whereas more reliability is provided for SOAP over WebSphere MQ clients. In either case, the input is a SOAP message that is logged in the warehouse for auditing or data mining. The requests are then filtered. Low-value and query requests are routed by SOAP over HTTP to services implemented on low-value hardware using basic Web-services implementations. High-value requests are decoded from SOAP to its internal format and routed over WebSphere MQ to a high-quality back-end service, such as CICS or IMS.

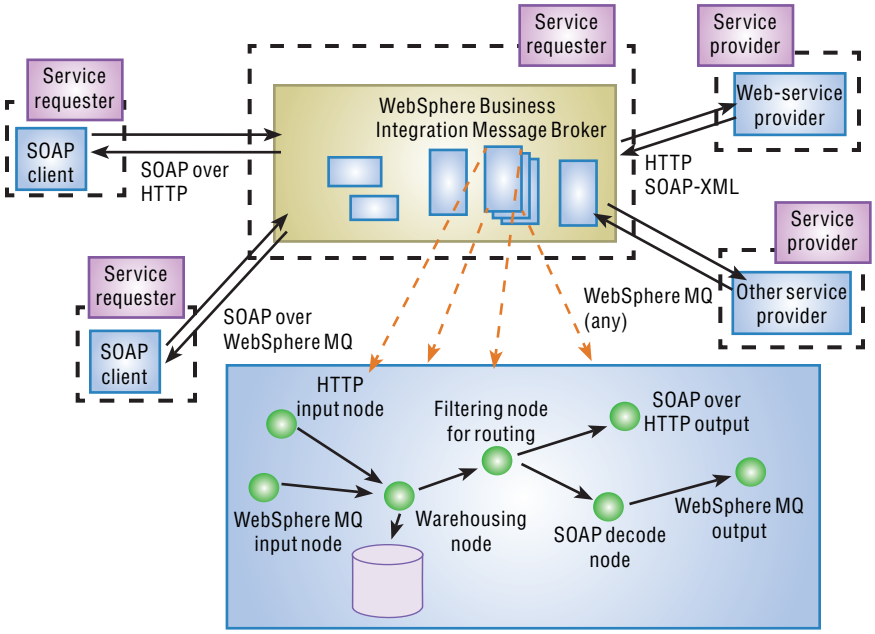

Figure 9. Web services in a more-advanced ESB scenario

Conclusion

The adoption of Web-services standards is growing. Over time, it is likely that more and more applications will be written to exploit Web-services standards. As a consequence of this trend, you might assume that new developments will have less and less need for the conversion and wrapping capabilities of an ESB. This conclusion is mistaken for two reasons. The first relates to the realities of maintaining investments in business systems. And the second relates to the need for tools to manage Web-services interactions, even when those services are implemented to Web-services standards.

The key word here is *new*. One lesson learned, time and time again over the past three decades (or more), is that old applications do not disappear quickly. They persist because they work. If this continues (and there is little reason to suppose otherwise), the need for the flexibility of an ESB in performing the transformations, routing and interconnection between legacy and new (Web services-based) applications is unlikely to disappear. Indeed, the relevance of an ESB should become ever more important as business demands become more and more dynamic. Even with pure Web services, discrepancies can occur between clients and services. These can readily be resolved using the transformation capabilities of a broker within an ESB.

What will change is that there will be closer integration of Web services with ESBs, as well as closer integration with application servers. This tight integration has positive benefits for application server-based applications. You can use an ESB for Web-services delivery (whether as service consumer, provider or intermediate), while also improving implementation efficiency.

For these reasons, adopting Web services in conjunction with an ESB can help preserve your software (and hardware) investments. New applications can be written more easily using Web services, because of modern development tooling and its improved usability. At the same time, these new capabilities can be combined, or integrated, with existing IT infrastructure and applications. As a result, an ESB can enable you to continue exploiting past investments in infrastructure and applications. Similarly, new investment in Web services-based applications is preserved as your organization's infrastructure evolves.

As such, ESBs can become critical if you need to change the way existing applications connect to each other, or you need to introduce new applications. The challenge of connecting applications in a flexible, maintainable way will continue well into the future. Making this integration ever easier, and cheaper, is of paramount importance to most companies. While the challenge of enabling application *X* to talk to application *Y* is achievable, the bigger gains come when *X* (or *Y*) can talk to *N* other applications as well. Making this function even more straightforward to define and then make operational is key to building a successful ESB.

What matters in an ESB is that it possesses the flexibility to enable software components to work together. For your organization, what matters is middleware transparency. With an ESB, you don't have to worry about the detailed middleware implementations that knit the parts together. Definitions can be described at a high level and then implemented at a low level.

Relational databases were successful in the early days by separating logical and physical models. ESBs can be successful for business-process integration—helping organizations to separate high-level tasks and low-level tasks. The business value comes through the reduction in the amount of costly, skilled manual intervention required.

For more information

To learn more about the enterprise service bus, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/integration/esb

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

Produced in the United States of America
03-05

G224-9192-00