A primer on the IBM Enterprise Service Bus for decision makers.

The term “Enterprise Service Bus (ESB)” is currently receiving a lot of attention from IT media and analysts alike; however, there is no clear, commonly accepted definition. Analyst definitions vary widely, but many are limiting the term to describe a very narrow notion of connectivity software for applications written specifically to the specific standards for Web services. Meanwhile software vendors, many of them selling nothing much more than a JMS compliant messaging system, are using the term to describe only what they have. This paper outlines IBM’s definition.

IBM Corporation

What business problem does the ESB address?

Studies show that application integration is one of the top challenges facing CIOs today[1]. Integration is not just a technology issue. It is also a business issue. Harnessing company resources to realize new opportunities and counter competitive threats is one of the most difficult problems facing a CEO. Business leaders are asking to be able to change their business, including their Information Technology -- *on demand.*

Business applications that do not communicate with each other can prevent CEOs from meeting their objectives. Typical business applications run on *different platforms*, have been written using *different programming languages*, represent data using *different data representations*, and utilize *different programming models*. It is a Tower of Babel. The result is not just that operational systems don't work with business systems, but also that a CEO cannot quickly harness his/her company resources to realize business goals.

Software vendors have, over the years, developed products to facilitate application communications. Unfortunately, different types of software have emerged for different business purposes. Point-to-point messaging software is used for application to application integration, publish/subscribe software is used for information distribution, and the new approach of Web services has introduced yet another connectivity paradigm based on commonly agreed open standards. Many companies have ended up with a mixture of technologies that are not integrated. The challenge is to find a way to get ALL software assets – old and new -- working together in a common reconfigurable connectivity framework that minimizes both cost and disruption.

The Enterprise Service Bus (ESB) is an architectural framework for addressing this problem. It provides an enterprise with a common way for all applications, existing and new, to communicate with minimal change to existing applications. *Its purpose is to help a company change its business, more or less on demand.*

An ESB solves four main classes of problem:

1) Application Communications

Programmers require a simple way to connect applications, without wasting time and money writing and maintaining complex connectivity logic. However, connecting applications is only part of their problem. The notion of connectivity does not itself address whether the information being transferred from one system to the next is actually useful. After all, applications run on different platforms, are written in different programming languages, describe and interpret data in different ways, and/or are implemented using different programming models. These applications need a common "communications" infrastructure to overcome and mediate their differences. The ESB provides the framework to overcome these differences, thus facilitating application communications.

2) Departmental or Corporate Mergers and Consolidations

Many businesses today have departmental silos that don't communicate well with each another. From an application IT perspective, each department may use different IT standards. This inability to communicate can inhibit a company from achieving its business goals. Similarly, companies do not need to integrate just their departments. They often have to integrate with other companies too. Investors in particular will eye an organization critically to see how fast it can be knit together with another business to improve shareholder value. The ESB is thus more than just a technology. It is part of a strategic business strategy to ensure that a company can quickly integrate its varied departments or its overall business with that of another.

3) Information Distribution

[1] For example, the Merrill Lynch CIO Survey Results, September 2004

Business events occur inside or outside an organization, but the people and systems that could and should respond be made aware of those event often aren't. The result: lost business opportunities. Many companies need a high-speed scalable information notification system that can deliver information *to the right place, at the right time, in the right format, potentially anywhere --and everywhere -- in the enterprise or beyond.* The business may also need a way to identify patterns of events, in real-time, and have no easy means of doing so. The ESB provides the framework that allows these events to surface and be delivered, to any application, person, or device in your organization.

4) Services-Based Integration

An organization is adopting Web services standards and a services oriented architecture and needs a mechanism for these applications to communicate. The organization also needs a way to mediate the differences between applications written to the Web services standards to those written to other internal or industry standards. The ESB provides a ready framework to mediate between Web services or to mediate between Web services and just about anything else.

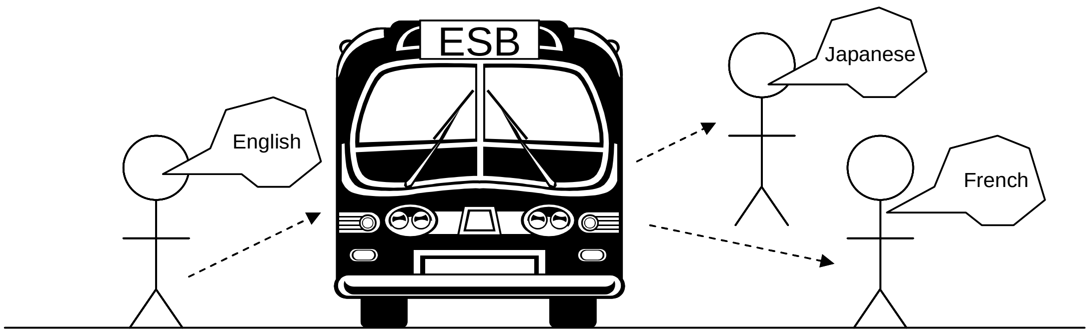
What is an Enterprise Service Bus?

Industry definitions of an Enterprise Service Bus vary. Some analysts and software vendors restrict its definition to describe only a connectivity infrastructure for connecting Web services. IBM's view is broader.

The IBM definition of an Enterprise Service Bus is that of an application communications infrastructure that allows any application to communicate with any other as a reusable asset or service. This notion of services is not new; IBM has supported the concept for a long time with message oriented middleware. What has changed has been the introduction of a new set of standards, *Web services*, AND an infrastructure specifically designed to connect them. Supporting these standards is important, and IBM does so within its ESB framework, but support for only these standards is not sufficient. Most applications in production today do not in fact support the Web services standards, and will not for some time. So what companies really need is an infrastructure that allows applications to be treated as "reusable services" whether or not they currently support the Web services standards. Companies actually need a communications infrastructure that can mediate between *all kinds of applications.*

An ESB is therefore best described as a communications infrastructure that enables applications - running on *different platforms*, written in *different programming languages*, which use *different data representations*, or which using *different programming models* - to communicate with each other, without changing the applications themselves.

A fundamental notion of IBM's view of an ESB is the notion of *"customized communications"*. Just as a speaker may need to customize a message for different audiences, applications need to customize communications when interacting with other applications. It is this notion of *mediating the differences* that lies at the heart of IBM's definition.



Step on the bus speaking one language. Step off the bus speaking another.

One can think of an Enterprise Services Bus as a passenger bus network spanning all the countries of the world. Each country communicates in its own way. The ESB mediates the differences. I step on the bus speaking one language and step off, potentially in multiple locations, speaking a different language, or potentially different languages (since communications may in fact be delivered to many targets each which communicate in their own way).

This notion of customized communications extends beyond the idea of translation. While on the bus, I may also want to carry out some other operation such as rerouting of communications, logging the interaction, augmenting the communications, comparing the communications with other communications and driving a specific decision as a result. Customized communications can mean <u>any</u> mediation that can occur to information as it moves from one application to another.

What is IBM's Enterprise Service Bus?

An IBM Enterprise Service Bus is an applications communications architecture that can be built today using current IBM products. The products support transport, transformation, intelligent routing, logging, mediating between programming models such as Web services, publish/subscribe, and point-to-point, open standards such as JMS and SOAP, and much more. If you have made an investment in IBM's proven, and popular WebSphere MQ product, you are already well on the way to having an ESB.

The elements of today's Enterprise Service Bus by IBM are as follows:

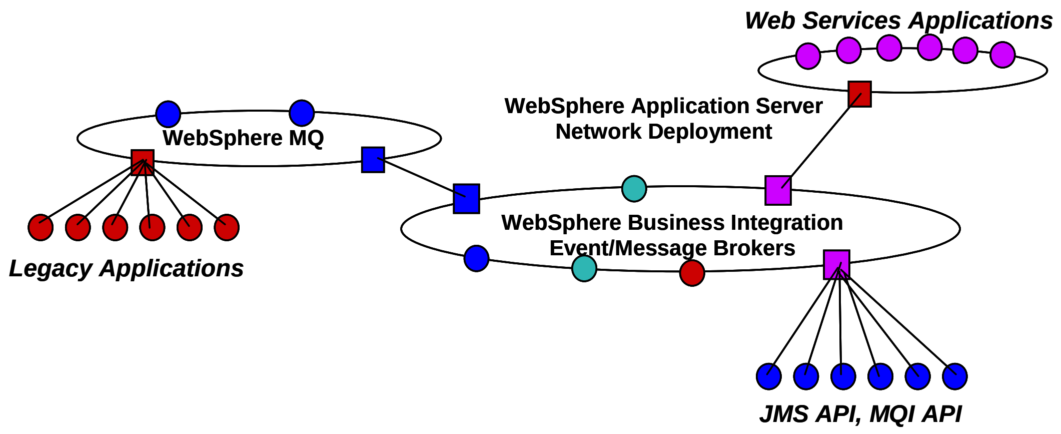
- **WebSphere MQ** is at the core of the IBM ESB framework. WebSphere MQ has long provided the reliable transport and common API for connecting applications written in nearly all operating platforms, programming languages, and message formats. In recent years, it has also been augmented to provide full support for JMS, SOAP and other open standards. Today it is the only commercially available messaging system that in one product combines JMS (the messaging application programming interface (API) used for the Java-based J2EE environment) with MQI (the messaging API in most common use for non-Java environments).

- **IBM WebSphere Business Integration (BI) Message Broker** is an application built upon WebSphere MQ that inherits many of WebSphere MQ's capabilities such as assured message delivery and end-to-end transactionality. WebSphere BI Message Broker adds the following:

  - additional specialized messaging protocols
  - a highly scalable, Internet capable, publish/subscribe service
  - the concept of "customizable communications" -- message routing, transformation, enrichment, warehousing, mediations between Web services;
  - interoperability of Web Services with non Web Services.

- **IBM WebSphere Application Server Network Deployment** extends the IBM ESB further by adding a Web Services UDDI registry and a Web Services gateway for secure mapping between Web Services requests and other formats defined using the WSDL standard, as well as JMS over MQ, and JMS over HTTP. All of this is made transparent to the original requestor of the information.

- **Most WebSphere products** have been or are being upgraded to easily plug into an Enterprise Service Bus. WebSphere Application Server V6 delivers a new Java messaging engine that simplifies administration, improves Web Services support and works effectively with WebSphere MQ and the WebSphere BI Message Broker.



An IBM Enterprise Service Bus is implemented in a topology that is right for your particular business and incrementally as your needs warrant. Multiple IBM Enterprise Service Bus components can be connected together over time in any number of units to produce a powerful, scalable distributed enterprise bus that connects anything to anything. An IBM Enterprise Service Bus is your bridge to both legacy application programs and new applications written to new standards.

An IBM Enterprise Service Bus enables you to do all of the following:

- Connect virtually anything to anything – operational systems, business systems, physical devices (including RF Tags and telemetry devices), Web browsers – all without incurring the cost and time associated with totally rewriting your applications.
- Hide the complexities of platform, software architecture, programming models, and network protocols. This enables your programmers to more productively spend their time working on business logic and getting the job done faster.
- Assure delivery of information, even when systems and networks go off-line. This means that you don't incur huge downtime costs trying to find out what happened when something fails.
- Distribute information to virtually anywhere in your organization, and beyond, in manner customized to the needs of each receiving application.

IBM is investing heavily in Enterprise Service Bus offerings and capabilities, and is continually enhancing its ESB portfolio. This means you can be assured that, by building upon the powerful WebSphere MQ and WebSphere BI Message Broker foundation already used by thousands of companies, you are making a sound investment.

What differentiates an IBM Enterprise Service Bus from other vendors' offerings?

Many vendors use the term "Enterprise Service Bus" to refer to a very narrow set of capabilities. Some offer nothing more than a JMS-compliant messaging engine. Some will even claim that because their JMS messaging system is written in Java, it is more of an Enterprise Service Bus than a messaging system written in another language like C++, or even COBOL. Others expand their ESB definition further to incorporate XML transformations and perhaps even orchestration of services.

IBM fully supports these capabilities too, but believes that an ESB, to be truly usable, needs to be much more. While one day all applications may be written to J2EE and/or comply to the Web services standards, that day is not yet here. So what companies need today is a bridge strategy that incorporates these standards – and everything else. The issue is that what companies really need is a common framework for all their enterprise applications so that they can achieve *on demand* flexibility. An Enterprise Service Bus must be able to connect *everything*, including applications written in programming languages, like C++ and COBOL, and programming models and data representations.

An IBM Enterprise Service Bus is the cornerstone of a Service Oriented Architecture. It is a vision that is central both to IBM's WebSphere product strategy and IBM's overall *on demand* initiative. It is this focus – to build an on demand world – that distinguishes IBM from virtually everyone else. With one common framework, an IBM Enterprise Service Bus supports different platforms, different programming languages, different data representations, and different programming models, and customizes the communications for each and all.

Here are six ways an IBM ESB solution is different and why these differences matter:

1) Customized communications and central management.

As already stated, the purpose of an ESB is to provide a common communications framework that does not require applications to be rewritten. It is about achieving *on demand* flexibility for a business. An IBM ESB customizes communications: information gets on the bus speaking one "language" and gets off the bus speaking another.

Some ESB vendors rely on their customers implementing a "canonical" model throughout the customer's entire organization and force them to use "fat" adapters to transform the data into and out of this canonical model wherever they get on the bus. This approach can be effective within the domain of an individual integration project or process interaction, but as an enterprise strategy, as requirements and technologies change, or as departments and organizations are linked, it is not typically manageable or affordable. Changes to a canonical form, for example following a merger, could require every "fat" adapter in an enterprise to be changed.

The IBM approach to an ESB focuses on manageability and speed of change. IBM blends the concepts of messaging, customized communications, and management into one. As information passes through the bus, it is transformed in flight to whatever the target requires. This might involve putting data into a canonical format or not. A key point is that an IBM ESB is a centrally managed network of interconnected message processing engines or 'brokers'. Adapters can still be used, but they can be thinner and more manageable. And IBM does not force you into a <u>single</u> canonical model, which means that you don't need to police your programmers at the enterprise level. And if you need to merge or consolidate departments, divisions, or companies, it can be done quickly and easily – *on demand.*

2) Rich any-to-any transformations, not just transformations that involve XML.

XML is an open standard used for describing data. It is a good idea, but it does not represent all of the data that companies typically use. Every industry has its own standards, and most companies have a library of legacy data representations – example COBOL copybooks, that are not in XML formats. Furthermore, XML messages tend to be rather large so if you need to shift a lot of messages around your organization and you need to do it at very high speed, XML formats may not fit your strategy.

Furthermore, many vendors don't offer "any" in-flight transformation services and others have only recently introduced offerings to do this. Those that do offer transformations, typically only handle only XML-to-XML (XSLT). Unfortunately, XSLT takes care of only a fraction of most companies' needs.

An IBM ESB can transform anything to anything, including XML-to-XML or XML-to-anything else. The IBM solution includes a logical message model that enables messages to be transformed from any format to any other. You can even implement a canonical model…where it's appropriate. Overall, this gives the IBM ESB a lot of power to help a company transform whatever they need to transform.

3) Secure, reliable connectivity of resources that communicate using different programming models.

Most vendors presenting an ESB story have very limited connectivity capabilities. They may only support JMS, the API used by applications written to the J2EE programming model (such as applications written to the IBM WebSphere Application Server). Interestingly, even though Java itself should run on many systems, many of these vendors only support a limited number of Java platforms. Furthermore, few vendors support the interconnectivity of telemetry and mobile devices, except through specialized unwieldy adapters. And virtually none of them support the MQI API, which is widely and heavily used throughout the industry.

IBM offers the widest support for your infrastructure -- 35 plus platforms, JMS, MQI, publish/subscribe, Web Services, and more. This is not just a portion of your requirements. It is your entire enterprise. The result is that you can rest assured that no applications will be left out.

4) **Scalability and performance with centralized administration to minimize the cost of ownership.**

Many vendors offer systems that only work as single-broker configurations; they don't connect brokers together to form a distributed bus. Even some who claim to offer an integrated broker network actually require you to build and administer all of the links between each and every broker at each and every broker. As a network grows, this can be a daunting administrative job, one that creates a high cost of maintenance and administration, particularly as your network grows.

An IBM ESB can include a network of brokers, implemented as a single administrative unit. Multiple Message Brokers can be implemented as a single broker domain. In publish/subscribe scenarios, specific subsets of an overall broker network can be identified as a publish/subscribe "collective" that can harnesses the power of hundreds of brokers to deliver information to tens of thousands of users – even over the Internet. The result is performance and scalability with minimal administration.

5) **Support for the JMS API without proprietary extensions.**

As already stated, JMS is a standards-based API for the Java programming language. If you write to it, your applications *should* be able to use any other vendors messaging system that supports JMS. Unfortunately, some vendors don't quite see it that way and have opted to add proprietary extensions to their JMS implementations. This means that if you write to their JMS extensions you will have locked yourself into their proprietary JMS model. So much for the promise of open systems.

IBM's JMS implementation supports the JMS specification without any extensions. This means that if you are writing in Java and you want to use JMS as your API, we don't lock you in.

6) **Support for the de facto industry-standard MQI API as well as JMS.**

WebSphere MQ, used by more than 80% of those who buy messaging systems, is *the* de facto industry standard in messaging. It grew to that position partially by providing the MQI API across more than 35 operating systems platforms and programming languages when other vendors were offering very limited support. As such, MQI is now pervasive in the marketplace, used by tens of thousands of applications. If you want to connect to those applications, MQI is the logical API to continue to use. It is also the only effective API to use across most programming environments other than Java, where you are more likely to use JMS.

The IBM solution supports both MQI and JMS, in one offering. This means a C++ or COBOL application written to the MQI API can talk with a Java application written to JMS. The result is seamless integration of JMS applications with whatever other applications you have today.

Where can I find out more about the IBM ESB?

Directly from the IBM Website:

- The IBM ESB web pages: http://www.ibm.com/software/integration/esb
- IBM makes the ESB real – Read the Charles Schwab story
- Understand Enterprise Service Bus scenarios and solutions in SOA
    - Part 1: The role of the Enterprise Service Bus
    - Part 2: ESB scenarios and issues driving the architecture

Through the IBM intranet from an IBM sales Specialist:

- ESB analyst papers: http://w3-3.ibm.com/software/analyst/
    - ➞ Forrester, Bruce Silver
- ESB Frequently Asked Questions: click here

Talk to your IBM Client Rep or IBM Business Partner.