Best-of-Breed ESBs

Identifying best-of-breed characteristics in Enterprise Services Buses (ESBs)

June 2003

A white paper from Steve Craggs

Vice-Chairman, EAI Industry Consortium

Sponsored by

Table of Contents

1.0 Executive Summary

As business integration has cemented itself at the top of the list of business and information technology (IT) concerns for many companies, attention has become focused on achieving this integration quickly and effectively while at the same time maintaining attractive levels of return on investment. In the past, proprietary solutions dominated the market, but a number of changes have taken place that promise to radically shift the business dynamics underlying business integration activities. Standards have become established that promise considerable market upheaval, offering a lower risk approach to integration with a more attractive price point. Integration approaches have been refined over the last ten years, resulting in the emergence of architectures that offer flexible and adaptable platforms for integration while at the same time making high levels of application component reuse possible. In short, the EAI market is undergoing a significant level of transition, as discussed in the EAI Consortium White Paper entitled ‘Raising EAI Standards’.

One of the most exciting developments to emerge throughout this period of change is that of the Enterprise Service Bus (ESB). ESBs address a number of key challenges in this market and are rapidly establishing themselves as flexible, productive and robust standards-based solutions to many integration needs. In addition adopting an ESB approach to integration can radically alter the business dynamics of the overall integration initiative. Used properly, the ESB promises to become the cornerstone of integrated operations, seamlessly linking business operations across the company and externally with partners.

But if the ESB is to become such a critical element of operations, then selection of the particular ESB offering to implement becomes a decision of major importance. Making the wrong choice is likely to seriously impact the pace and ultimate success of business integration initiatives. Therefore it is worth taking time to contrast the characteristics of ESBs and to identify the key areas that represent best-of-breed capabilities. When these attributes are clearly defined, an informed decision can be made about any ESB investment, optimizing business risk and ensuring as far as possible a successful implementation.

The key characteristics to consider are **basic ESB services**, **robustness**, **scalability/performance**, **security**, **breadth of connectivity** and **tooling**. This paper reviews these areas, producing checklists of best-of-breed functionality for each key characteristic. Note that this does not mean that ESBs satisfying every best-of-breed aspect are therefore the best – indeed, it is highly unlikely that any ESB offering could claim to do this today. Instead, this analysis provides a set of measurement points that give a prospective purchaser a way to compare products.

The extent to which an ESB offering addresses these key characteristics relies to great extent on how well the vendor understands the current market requirements. And since ESBs are substantially standards-based, some analysts anticipate that the offerings are likely to be virtually identical. But that is not the case. Standards are technical specifications typically covering only the interfaces, leaving the ESB architecture and implementation decisions to the vendors. And that is where the differentiation is most apparent in ESB products. The question becomes one of how well the vendor has defined and implemented its ESB.

Particular attention should be paid to the first three of the characteristics—**basic ESB services, robustness, and scalability/performance**—because these areas are usually linked to the fundamental product architecture and design decisions. Whereas some of the other key characteristic areas can be improved upon over time in an evolutionary fashion, areas such as scalability, performance and robustness are notoriously hard to address once the initial design decisions have been made. If poor decisions have been taken early on, it is often virtually impossible to recover from them without having to substantially rewrite the entire product implementation.

Using these checklists as part of this product selection process will help to ensure that the ESB selected will become a key element of integration strategy, and will deliver the promised benefits.

2.0 Introduction

Business integration has grown in importance over the last ten years with companies continually striving to streamline and automate business operations internally and externally across the entire value chain — while at the same time optimizing the return on existing IT investments as much as possible. The ubiquity of the Internet has made possible levels of integration that could only be imagined previously, with IT being able to support business processes extending all the way from the back office through the front office to partners and even consumers. Although the last few years have seen extreme levels of conservatism brought about by the global economic downturn, business integration has remained the number one concern of IT, seen as a way to achieve the maximum return from existing IT investments while at the same time contributing positively to the corporate bottom line.

This intense interest in IT-supported business integration has driven major advances in technology within the integration software space (often called Enterprise Application Integration (EAI)). As a result the development of technologies such as Service-Oriented Architectures (SOAs) offer ways to achieve the desired levels of business integration effectively, mapping IT implementations more closely to the overall business process flow. These technologies often focus on providing what Gartner Group calls an *enterprise nervous system*, a backbone for the entire scope of IT operations that can link together all the different IT components across the enterprise and beyond in a productive, efficient and effective manner.

3.0 The Enterprise Service Bus (ESB)

One of the most promising developments in defining an enterprise nervous system is the Enterprise Service Bus (ESB). An ESB provides a whole range of functions designed to offer a manageable IT backbone that is modern and standards-based. The emergence of standards in the integration marketplace has caused a major disruption to the previous market order and promises to mark a real point of transition, radically changing the business dynamics of EAI investments. This has opened the way for new standards-based offerings, and it is this opportunity that has been seized upon by ESB vendors. As companies look to improve their levels of business integration, ESBs promise a number of key benefits such as faster implementation and deployment of new projects, a flexible platform for future expansion offering a high degree of reuse, seamless interoperability, and an improved rate of return on existing skills.

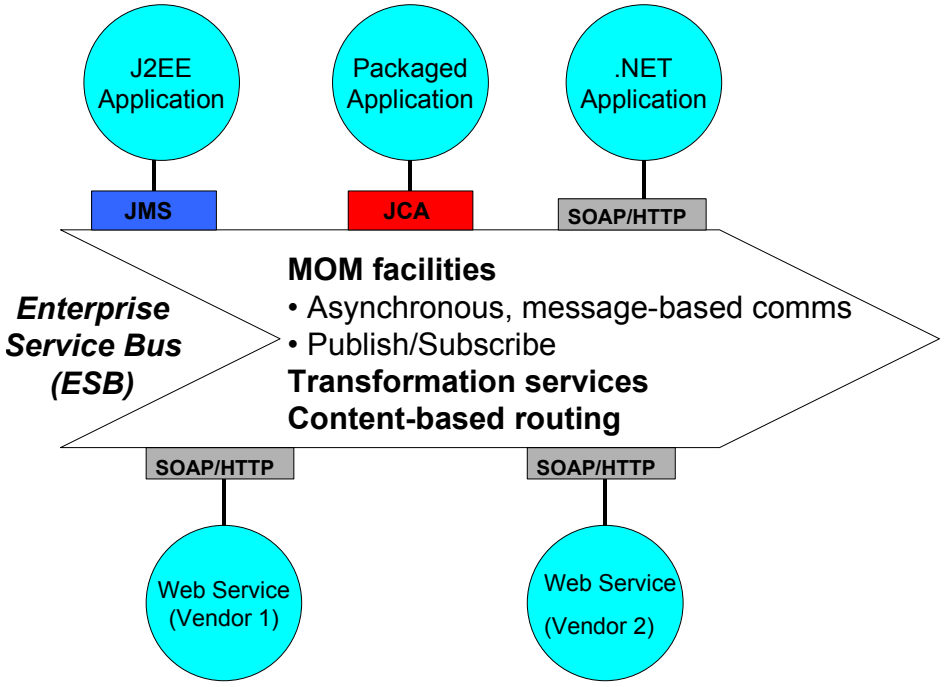
A number of vendors now deliver commercial ESB implementations, some as stand-alone offerings, others as part of offerings such as Application Server platforms. Major analyst groups see the ESB becoming a major market force over the next couple of years. But how should the prospective buyer evaluate the different offerings in the marketplace? Before looking at the specific characteristics by which to judge an ESB offering, look more closely at the concept of an ESB and why it has become such a popular technology innovation. The ESB concept evolved out of the principles of a Service-Oriented Architecture (SOA). SOAs describe a type of IT infrastructure where IT components can be accessed as services with a defined form of invocation. Typically business operations running in an SOA comprise a number of invocations of these different components, often in an event-driven or asynchronous fashion that reflects the underlying business process needs. SOAs are popular for a number of reasons – they clean up interfaces between components and enable so-called composite applications to be assembled from a mix of existing and new components. SOAs also adapt naturally to a business process approach and dramatically improve reuse. The results show a reduction in time-to-market and increase in quality of new projects.

It is possible to adopt an SOA approach to integration without using ESBs. Numerous vendors supply various different forms of EAI products, including message brokers and process integration solutions, which could also provide the foundation for an SOA. Application servers frequently offer functionality that can be used in a similar fashion. Indeed, one of the technologies receiving the most hype today, Web services, can also help significantly in delivering an SOA. That's due to a Web service's focus on standards to govern the access to and invocation of components.

Message brokers tend to be rather heavy duty and often require significant additional skills as well as the strong likelihood of needing expensive external professional services to successfully implement the deployment. Application server vendors are so intent on providing homogeneous integration within their own world that they rarely implement the technology in such a way that it can interoperate with other vendors' application server implementations. Web services suffer from rather incomplete and competing standards definitions that might not ensure easy

interoperability across different implementations. In addition, although most of these offerings can be used to build an SOA, they do not offer a complete package but instead provide simply the functionality - the majority of work to build the SOA is left to the user.

In contrast, an ESB is a pre-packaged SOA implementation, already consisting of the necessary functional components to achieve the SOA aims. The diagram below illustrates the general concept of an ESB.



An ESB offers a backbone that can extend Messaging Oriented Middleware (MOM facilities) throughout the entire business value chain, connecting components across different spheres of business operation. It has its own communications capabilities so it can carry out this linkage by offering asynchronous, message-based communications and queued messaging as well as publish/subscribe messaging, a mechanism that allows the broadcasting of information to users that have registered an interest in a subject. As well as dealing with the 'backbone' style communications, the ESB also offers integration with the broad spectrum of components likely to be encountered through the use of various 'standard' binding approaches such as Web services, J2EE Connector Architecture, JMS, COM, and other common mechanisms. This integration is dealt with by the ESB in a standard, service-oriented way, independent of the particular binding technologies. The ESB also offers a level of transformation capabilities and XML services to address the problem of differing data format requirements in the heterogeneous components, and intelligent routing facilities to govern the flow between components.

The value of this backbone, in concept, should be quite apparent. With an ESB in place, operational IT flows can move seamlessly between different components, departments, disciplines and other companies in the value chain. As information

flows across the ESB on its specified path, value is added at each component stage until the operation is complete. The information flow typically contains all the details necessary to carry out the business service, removing the need found in traditional integration broker implementations of having to constantly return to a central hub for this knowledge. And all of this comes with the advantages of a packaged solution approach, avoiding the need to try to assemble this type of functionality from a set of different vendor offerings.

In summary, an ESB can offer:
- An easy route to enhancing, streamlining and automating business operations
- End-to-end connectivity both internally and with third parties and consumers
- A flexible base for future developments
- Significant opportunities for reuse of components
- Rapid time to market and increased productivity for new projects

But there are vast differences between the different ESBs on the market today, because even though the concept of an ESB is well-defined and interfacing to that ESB is governed in a number of cases by standards, **the way the ESB concept has been implemented by the vendor makes a huge difference.**

The rest of this paper focuses on the core characteristics that should be evaluated as part of any purchase decision.

4.0 Core Characteristics of a Best-of-Breed ESB

ESBs can provide real business value to companies looking to operate and compete more efficiently and effectively. ESB implementations take advantage of the intellectual investment made in developing standards thereby yielding numerous benefits to the end user. Standard-based solutions are generally easier to swap in and out than proprietary ones, offering a level of flexibility, a better bargaining position over price and reduced business risk due to vendor lock-in. But the secret of a best-of-breed ESB lies in the design and development decisions and effort invested in the particular product implementation. Therefore to maximize the opportunity for success in any investment in ESBs, it is important to know what metrics to use to gauge the quality and effectiveness of an ESB implementation.

In its most effective setting, the ESB is the backbone of the entire corporate value chain, supporting every area of activity and forming a critical component of the corporate IT infrastructure. Legacy investments such as in-house applications and commercially available packages are leveraged by being attached to the ESB, while new applications can be built ‘integration-ready’, able to automatically take advantage of the ESB capabilities right from the start. The backbone offers connectivity beyond the corporate IT structure into third parties and even to consumers.

Fundamental ESB functionality includes basic functions such as the messaging backbone, XML services, intelligent routing, transformation, management, the overall service-based architecture, and support for distributed deployments.

In addition, value is added when the essential components involved in critical business operations are highly resilient and robust, and are able to handle whatever loads the business might generate, now and in the foreseeable future. Components must be capable of widespread deployment and corresponding wide physical distribution. Since components will be spanning different departments and locations and might be going outside company boundaries, security must offer authentication and authorization mechanisms that adhere to the corporate policies. Another key area for an ESB is its connectivity capabilities—it must be able to interconnect component types in as many environments as possible. And in order to ensure maximum productivity, it is essential that the development toolset be powerful, comprehensive, well documented, and capable of covering the project life cycle.

These factors establish the fundamental characteristics and the key value-add characteristics that should be addressed by a best-of-breed ESB:

- Fundamental ESB characteristics:
  - XML, messaging, transformation, intelligent routing services
  - Basic connectivity (Web Services, J2EE Connectors, JMS)
  - Service-oriented architecture
  - Support for highly distributed deployments
  - Manageability
- Key, value-add characteristics:
  - Robustness
  - Scalability and Performance
  - Security
  - Breadth of connectivity
  - Development / Deployment toolset

Each of these characteristics is discussed at length to help develop checklists for prospective ESB buyers as they evaluate products. It is important to note that these checklists do not define a product specification. Beyond basic ESB functionality, the other categories suggest key areas that might be more or less important in a specific implementation. As most ESB vendors do not satisfy every characteristic completely, the checklist provides a set of questions to present to a vendor.

Consider though that some functionality can be added in later product releases, but base functionality, robustness, and scalability result from fundamental design decisions, and trying to improve these aspects of a product is usually a very difficult task.

4.1 *FUNDAMENTAL ESB CHARACTERISTICS*

ESB characteristics are essentially requirements rather than best-of-breed characteristics. In other words, if an offering does not satisfy these basic service needs then it cannot claim to be an ESB. These services cover the overall architectural approach of the integration offering, the basic services available over the ESB, the on-ramp and off-ramp connectivity services, support for a high level of distribution, and the manageability of the overall implementation. When properly assembled, the fundamental ESB characteristics are not just a collection of parts, but rather a packaged, out-of-the-box solution that has been thoroughly tested and

documented. The Service Oriented Architecture (SOA)—offering its approach to allow IT components to be invoked as steps within the overall business service delivery— is an essential attribute of any offering claiming to be an ESB. The workspace for development and deployment of the basic ESB functionality must enhance ease of use and accelerate project delivery.

The fundamental ESB consists of:

- Bus-related engines that provide transformation, XML and intelligent routing services and the communications bus itself
- Definition tools and repository services
- Administration and management services
- Support for the standard forms of connectivity such as Web services
- Adherence to industry standards for enterprise application integration

Basic bus services

The bus services for transformation, XML services, routing, and communication are broadly interpreted by vendors so look a bit closer to evaluate the effectiveness and completeness of an ESB's bus services.

Transformation services are essential in an ESB. The various components hooked into the ESB have their own expectations of data formats, and these might differ from other components. One of the most frequently noted examples is the variation in date formats between the US and Europe. A major source of value in an ESB is that it shields any individual component from any knowledge of the implementation details of any other component. The transformation services make it possible to ensure that data received by any component is in the format it expects, thereby removing the need to make changes. Using the date formatting example, transformation rules could automatically switch the date format when passing from a European-based to a US-based component.

The areas of differentiation in an implementation of transformation services are likely to be the extent to which the implementation uses a standards-based approach to transformation rather than a proprietary engine, and the types of tools provided to make the mapping definitions. Most companies find that a GUI-based tool that allows record mappings to be manipulated through the use of cut-and-paste or drag-and-drop provides the best productivity. In addition, there are standards emerging in this area that might also be beneficial, such as XSLT and XQuery.

XML services are generally accepted and are unlikely to differ significantly between vendors. The power of XML in making data structures 'self-defining' provides an important cornerstone of ESB functionality, since the very nature of an ESB is to pass information or data between different IT components as part of an overall business service. Transient XML support will certainly be required, and persistent XML support might also be of interest although the performance gains of the latter approach tend to be outweighed by the corresponding increase in complexity. However there are other aspects of XML document handling that could also be of interest but that are not considered to be part of the standard XML services package, such as handling very large XML documents and even providing

some sort of document management capabilities. The XML services offered within any ESB should therefore be reviewed carefully.

Intelligent **routing** services provide another part of core ESB functionality. In order to be able to define the business services that are made up of the various components on the ESB, it is necessary to be able to specify the required flow from component to component. But the path used to physically get from one component to another has to be understood by the ESB, and indeed the ESB may well want to dynamically alter this path, such as a reaction to a failure in part of the network.

A particular service might want to take decisions on the component flow based on the results of a particular step. For example, a financial payments service might want to use a different component flow for handling large value items than it would for other items. In this case, the ESB must provide a routing service that can intelligently consider the content of the information being passed from one step to another and choose the appropriate next business step based on that information. This sort of intelligent routing is fundamental if the integration solution intends to handle the company's business rules effectively.

The ESB must provide some form of **communications** pipe that can act as the bus over which all processing can happen. This pipe must support asynchronous messaging, a key to being able to map IT services more closely to business operations. Business operations often consist of steps that can be carried out in parallel, or that have widely different expectations of time to completion. An asynchronous mechanism is essential to be able to implement this type of model.

Communications can add robust features such as publish/subscribe messaging and store-and-forward messaging. Under publish/subscribe, information is published to any subscriber authorized to receive on a topic where a publisher is sending messages. Filters can be added to let subscribers further refine the information that matches their registered interest, a highly efficient way to operate in a many-many environment. Store-and-forward holds messages in situations where variable levels of availability are likely, such as in operations that cross many time zones. The information can be stored until the next step in the service is open for business.

Although not absolutely essential for a solution to qualify as an ESB, JMS-based messaging implementations are by far the most likely choice for this communications bus. The benefit of choosing a standards-based approach has already been discussed, and JMS functionality matches well to the ESB communications requirements.

Basic connectivity

An ESB must quickly and easily enable new components to be attached to the bus for use by any authorized services. That's basic connectivity. Later, in section 4.5, the breadth of connectivity is discussed but this section looks at only the basic level of connectivity required for a product to be considered as a true ESB.

A commonly stated benefit of using an ESB as an integration tool is that it resolves a problem created by the various application server vendors with their Web

services implementations. Web services covers a set of standards that theoretically make it easy to call different application components in a standard fashion to perform a well-defined service – very similar to part of an ESB's functionality. However, the application server vendors have tended to carry out their Web services implementations with a distinctly inward focus. As a result, it is likely that even though such a Web service is wrapped in standards, the different implementations across the application server vendor community might not interoperate readily—an expectation in the value proposition of Web services.

When an ESB is used, the differences can be handled and rendered effectively transparent. Therefore, support for Web services as a form of connecting to components is an essential requirement in any ESB, but even more so it is important that ESBs can interoperate with any of the more common Web services implementations.

Anyone that intends to build an integration solution will likely have a wide breadth of connectivity needs to span all of the applications and environments that are in use. Some of the earlier integration offerings such as message brokers have invested large amounts of money in trying to offer as wide a spread of connectivity options as possible. However an ESB is inherently positioned as a lighter-weight and more affordable offering. An ESB would likely find it impractical to try to offer connectivity to every possible application, package, database and environment. Instead, given the general theme of standards utilized by ESB implementations, it is much more important that an ESB should support whatever standard approaches exist to connectivity. This leads to the need to offer J2EE Connector services as part of any credible ESB offering.

The J2EE Connector Architecture provides a standardized way to integrate with a particular application component. The reason why this is so important is that many of the larger application package providers have realized that their customers need to integrate their applications with others, and that currently they have to struggle to do this spending considerable sums of money on complex integration products and extensive (and expensive) external professional services. As a result, these package providers are keen to take a more 'open' position in the market and provide some form of standard connectivity to avoid this time-consuming and expensive work. As a result, many of the key package vendors are implementing J2EE Connector-based interfaces as well as web services interfaces to make their packages more accessible in an integration scenario. So support for J2EE Connector services is another key requirement of any ESB.

The question is where to draw the line between connectivity services that absolutely have to be made available in order for an offering to qualify as an ESB, and connectivity services which are valuable to have but which more fairly fit into the category of best-of-breed differentiators rather than essential base functions. Perhaps the only other connectivity requirement on the 'must have' list should be the ability to interoperate with other messaging products, since it would hardly make sense to introduce an ESB backbone that cannot interoperate across other communications facilities. The objective has to be to allow information to be exchanged seamlessly across the entire enterprise.

Support for highly distributed deployments

An ESB has to support widely distribute deployments. This is where the service-based SOA approach comes into its own—in a widely distributed solution, it is imperative that services can be accessed in a standard way without the need to understand the underlying technologies or global location. Intelligent routing is also vital, since trying to continually refer back to a central hub holding the information about the next step in the operation would bog down even a modest deployment. Indeed, when there are a large number of nodes in an ESB deployment there may be numerous possible paths to use in routing the information from one step to another, with unavoidable performance implications if less efficient routes are chosen. The combination of intelligent routing with the global location transparency is an essential requirement in any large-scale worldwide deployment.

Manageability and associated deployment tools are tied to this characteristic because a widely distributed environment makes it important to centralize definitions and metrics to provide management views of an entire domain from one or many global locations.

Manageability

Many software vendors provide smart technology to address technical problems and challenges, but often these same vendors do not supply the operational tools needed when actually running a production implementation of the product. This is a particularly important issue for ESBs – the highly distributed nature of ESB-based operations and the fundamentally asynchronous nature of ESB communications both contribute to a high degree of management complexity that must be addressed by any ESB.

This area tends to consist of three segments:
- Administration / Deployment
- Monitoring
- System actions

Because of the distributed nature of an ESB implementation, both within a company and outward to its partners, administration and deployment are vital issues for ESBs to address. Support is best provided on a ‘single point of control’ basis so that system definitions can be defined and maintained then distributed across the full breadth of the ESB topology. If this is not the case then it becomes almost impossible to use an ESB in anything but the simplest of circumstances.

While creating a deployment is one challenge, there is a distinct need for definitions to be adjusted, added or replaced in a dynamic runtime environment. This is even more of a challenge, since a particular set of definitions might be ‘in use’ at the time a change is being made. Careful design must ensure that these types of situations are handled effectively without compromising the integrity of the system.

When the system is running in a production scenario, it is very important to be able to monitor the system behavior and performance to ensure that the correct level of service is being provided. Careful monitoring of system activity can help to identify potential problems before they are manifested in overloads or failures. For

example, when a particular resource is in danger of being completely consumed then as long as the operations staff are aware this situation is arising they may be able to take corrective action before a failure occurs. This type of activity is very familiar to mainframe users, who are used to having tools that watch over the overall system performance and warn of emerging problems. In the case of an ESB, the issue is wider than just a company's own systems. System behavior needs to be monitored across company boundaries to effectively maintain the availability of the ESB-based business services, and this monitoring should be capable of at least interoperating and sharing alerts with corporate management frameworks. While specialty toolsets such as those offered by expert systems management vendors provide the recording and visualization of metrics, an ESB product under consideration must have the data collection and notification hooks in place to allow this function to be added at a later date. A good indicator of this potential is capturing of comprehensive system statistics, even if the data is currently in the form of system reports. If an ESB is gathering these statistics then the facilities are in place to allow this information to be gathered dynamically in a live environment.

Once a problem or potential problem is identified, there is a need for functional capabilities in the ESB to take action to resolve the problem or prevent its continued manifestation of an emerging problem. These facilities are ideally available from a central, single point of control to provide an overview of extensively distributed resources. These tools could be more universally usable through a Web-based interface to allow them to be used remotely, since it may be necessary to have someone participate in the problem situation and take action from systems that are not specifically set up to run management tools. Functionality needs to cover the range of operational activities (such as starting and stopping processes, re-routing operations and perhaps even re-booting servers) as well as a set of problem-determination functions such as application tracing and message editing. Without these tools and facilities, the ability to manage the ESB would become observational, handling issues only after things break.

Key Characteristics: Fundamental ESB Services

- SOA implementation
- All-in-one package
- Support for standards
- Bus services
  - XML support
  - Transformation
  - Intelligent routing
  - Communications services
    - Asynchronous
    - Pub / Sub
    - Store and Forward
    - JMS-based
- Support for highly distributed implementations
  - Service-based approach
  - Location transparency
  - Technology transparency
  - Intelligent routing
  - Single point of control
  - Deployment support
- Connectivity services
  - Web services
  - J2EE Connectors
  - JMS
  - WebSphere MQ
- Administration / Deployment
  - Single point of control
  - Dynamic change
- Monitoring
  - Problem determination
  - Problem prediction
  - Internal and external support
  - Support for enterprise management frameworks
- System actions
  - Single point of control
  - Remote access capability
  - Start / stop facilities
  - Manual routing support
  - Tracing
  - Message editing

4.2 *ROBUSTNESS*

Providing the levels of integration possible when using an ESB can deliver huge returns to a business. However, by its very nature, an ESB is involved across widespread business activities, and therefore could represent significant business risk. If all operations take advantage of the ESB to become more efficient and effective, then any disruption in the ESB's availability or level of service could have serious implications. It is therefore extremely important that the chosen ESB is highly robust. The more robust the ESB is, the lower the level of business risk becomes.

So robustness is an essential best-of-breed characteristic for ESBs. But what does this mean in practice? Robustness is often considered in two ways:

- **Fault avoidance**, ensuring problems do not happen
- **Fault tolerance**, ensuring that if and when problems do happen, they have little or no impact on service

It is important to consider both these aspects of robustness. Obviously not having problems at all is wonderful, but in reality problems are going to happen, if for no other reason than human error. So it is necessary to be able to address problems quickly and effectively, preferably without the end-user becoming aware that anything is going wrong.

Fault avoidance

Perhaps the most obvious contribution to reducing the occurrence of problems is to ensure that the quality of the software product has been thoroughly and vigorously tested. Unfortunately this can be quite hard to judge when making an assessment,

generally only becoming apparent after some months of experience. However, discussing a candidate vendor's development process and testing procedures provides an indication of code quality, and asking another user is even better. But fortunately there are other factors that can be evaluated more easily and might give additional indications of efforts taken by the vendor to reduce the occurrence of problems.

One such factor is the use of **standards**. The EAI market (in which the ESB concept fits) has seen a lot of standards introduced over recent years, such as XML, JMS, J2EE Connectors and Web services (and their standards for SOAP, UDDI, and WSDL), that cover a wide range of technical interface specifications. Adoption of these standards within an ESB implementation will improve product quality for two main reasons. Firstly, standards specifications are common and therefore have already received a degree of validation in the marketplace, proving that they work. Secondly, use of standards tends to reduce the amount and levels of specialist skills required to implement an ESB, therefore once again reducing the likelihood of introducing new problems through lack of understanding of a new or exotic technical discipline.

This last point leads to another factor that should be assessed in evaluating the area of fault avoidance - **ease of use**. This translates in business terms into the level of skills required to develop, deploy and operate the ESB solution, and the level of productivity of these resources. Having to hire skills from third parties because the product is too complex for in-house staff to handle can cause major problems in the future, because once the hired skills complete the project, in-house staff might unwittingly cause a problem because of a lack of understanding of the implementation. But if the ease of use of the ESB is good enough, existing staff—with some training—can ensure that they are less likely to introduce errors in project deployment or make operational errors once the project is in production. The trick is to identify the skills required to develop and deploy solutions with the ESB under consideration and then to review these skills against those that are already available in-house.

Another key factor in fault avoidance is scalability. It is essential that growth in usage of the ESB does not run into some issue constrained by the limits of the ESB that brings the system down or prevents the ESB from functioning normally.

Fault tolerance

Even with the best fault avoidance plans, problems still occur. The way that an ESB responds to problems is therefore of great concern. Problems should be able to be isolated and resolved without impacting running systems. But that is not always possible. So, a more realistic goal is to ensure that any problems that do occur create the minimum possible impact on operations.

One aspect of fault tolerance of particular importance in an ESB solution is that of **intelligent routing**. The basic feature of an ESB to choose the route for information to flow intelligently means that it is possible for an ESB to offer the ability to route around problems in the network wherever appropriate. For example, if a network problem blocks a particular path between nodes, an ESB should try to reroute information around the failure so that the target can still be reached. The

presence of this type of functionality or the lack of it can be a good indicator to how well the ESB in question can deal with problems.

A common aspect of fault tolerance is **redundancy**. This is the concept that having backup or spare components makes it possible to continue providing a service even in the event of failure. This is just as applicable to ESBs as it is to any other aspect of IT. However, this has a number of particular implications on an ESB. ESBs must have some form of information storage, where they can record the system definitions they require and information about the particular services that are using the ESB. Without this information the ESB cannot function, and therefore it is likely that a fault-tolerant ESB will offer some type of support for mirroring this information in some way. Cluster support for the various High Availability (HA) **clustered servers** adds further redundancy for critical nodes in the enterprise.

Another aspect of fault tolerance is the area of **recovery**. This refers to the situation where, for whatever reason, an application or a system fails. At the application level this generally requires some sort of rollback or checkpoint-based recovery mechanism to be supported by the ESB.

For example a business service that consist of multiple components on multiple platforms may have tied file updates in different locations that form a single logical operation, called *a single unit of work*. If money is being transferred from one account to another, for instance, then it either must be removed from one and entered into the other, or left alone, unchanged. It must never be allowed to be removed from one account and then fail to be entered into the other. The ESB will therefore have to participate at some level in synchpoint management and **unit-of-work management** to ensure that the system integrity is maintained. In addition it may be necessary for **compensatory transactions** to be supported that can correct out-of-step actions in the event of an unrecoverable failure.

This recovery issue is particularly complicated in the event that long-running transactions are being carried out over the ESB. These are transactions that have a long lifetime as opposed to the normal sub-second lifetime of an operation. A problem resolution service for instance might need to be ‘in the system’ for a matter of days while various activities are carried out as part of the service operation. This means that state has to be maintained over this period in case recovery is required, another issue for a best-of-breed ESB to handle.

Key Characteristic: **Robustness**

• Fault avoidance
  ○ Vendor quality procedures
  ○ Standards adoption
  ○ Ease of use
  ○ Scalability (see later)
	• Fault tolerance
	  ○ Routing around failures
	  ○ Redundancy support
	  ▪ Mirroring
	  ▪ HA clusters
	  ○ Recovery
	  ▪ Unit of work management
	  ▪ Compensatory transactions

4.3 SCALABILITY AND PERFORMANCE

Given that an ESB is likely to become an IT cornerstone of business operations, it is of paramount importance that an ESB provide the optimal levels of scalability and performance. A high level of **performance** is essential to ensure that newly integrated and automated operations can be carried out effectively and efficiently, despite the inevitable spikes in demand for particular services. **Scalability**, on the other hand, is critical in ensuring that an ESB can deal not just with current projects (likely in themselves to be highly distributed, probably across company boundaries) but can also provide an extensible, adaptable platform for future growth.

Scalability is particularly important for any pervasive integration solution. Experience has shown that once this type of integration backbone is in place, perhaps for a relatively small number of projects, it often triggers a sudden surge in new projects being added as different areas of business operations see the value in the increased level of business integration. ESB traffic and overall usage is likely to start to grow very rapidly as the benefits are proven, and the worst possible scenario is for this to cause the ESB implementation to break down or be unable to manage the volume of work. If this were to happen then it would cause major setbacks to corporate integration strategy and plans, with serious repercussions on bottom-line company performance and competitiveness.

Whenever a new layer of technology is provided in an IT solution there is a legitimate concern about the additional overhead introduced. However, in the case of an ESB, these concerns translate into how well the particular ESB implementation has been designed. For example, asynchronous communications mechanisms introduce the possibility of carrying pieces of a business operation in parallel rather than serially as would be the case with synchronous communications. If this is achieved effectively then it is likely that any overhead introduced by the ESB will be more than adequately balanced by more efficient business operations realized from multi-tasking.

However achieving multi-tasking and parallel processing of business operations requires some sophisticated design and development work. If elements of a business operation are occurring in parallel, it is important to keep track of the status of these operational steps to maintain the integrity of the overall operation. In addition there might be steps in an operation that must be done sequentially—for example, not shipping goods in stock until the credit authorization has been received. These factors must be addressed by the ESB to make these performance advantages worthwhile.

Then there is the question of performance spikes. Most operations have a band of tolerance for response time, beyond which customer satisfaction could be compromised or competitive responsiveness lost. Unfortunately many businesses have to struggle with heavy fluctuations in demand. Sometimes system resources are overloaded by data volume spikes rather than traffic volume because some operations require the transmission of very large amounts of data between operational steps. Therefore any ESB implementation has to cope with these situations while maintaining an acceptable level of performance and response time. With load-balancing intelligence, ESBs can allow new threads and even new servers to be employed automatically when required.

Heavy volumes will almost certainly make some level of activity prioritisation capability essential, and indeed varying classes of service are also likely to prove useful. Another aspect of scalability that is often overlooked is that, as deployments become more and more complex, it is no longer practical to shut down the ESB temporarily while new components are added or changes are made to definitions. Once changes have been validated and their quality has been assured, they need to be deployed transparently without any disruption to ESB operations.

Key characteristic: Scalability and Performance

• Performance
  ○ Asynchronous messaging
  ○ Multi-threading
  ○ Load balancing
  ○ Large data handling
  ○ Optimized path selection
	• Scalability
	  ○ Prioritization services
	  ○ Classes of service
	  ○ Dynamic change support
	  ○ Transparent resource addition

4.4 *SECURITY*

With ESB integration projects spanning multiple departments and quite possibly companies, and the likelihood that these projects are fundamental to business performance, security is of major importance. There is a realistic instinctive fear that improved connectivity means that sensitive information might now be intercepted or mutilated by malicious lurkers. This is an issue that any best-of-breed ESB must address.

Security concerns with integration projects center around three specific areas of concern:

- Access to components or services
- Exposure of information as it passes from one component to another
- Control of system tools and facilities

The universal connectivity theme behind ESBs naturally causes concerns over access to the components hooked into the ESB and the overall services offered that flow across the ESB, particularly if these are being made available to external companies. This makes it very important to be able to restrict usage only to those who are authorized. There are a number of security technologies available in the market that can be used to carry out this authorization task and to control access based on the level of authority granted by the system, so it is only necessary to ensure that the ESB supports the techniques that the particular user wants. However there are one or two specific issues here that the ESB needs to address, such as the question of a *component*'s authority to access another component as opposed to a *user*'s authority.

Also, another feature worth considering is whether any form of non-repudiation is provided as part of the access control capabilities. Non-repudiation is particularly important in the financial industry, and refers to the ability to be able to prove that a request really did come from an authenticated and authorized user. As an illustration of this point, suppose a bank receives a request from an account holder to transfer money to someone else. If that account holder were now to approach

the bank claiming that authorization for this transfer had never been given and seeking recompense, it is important that the bank can show proof that it did indeed receive a transfer request and that it indisputably originated from and was authorized by the account holder.

The issue of exposure of information is a very sensitive one. The concerns here are twofold – has anyone been able to see the information as it passed along the connectivity pipe from one component to another, and has anyone been able to alter it as it was transferred. To address the visibility of information, the most common tactic is some form of encryption capability so that sensitive information can be encrypted before transfer, and decrypted on receipt. However, an ESB needs to ensure that the definition tools used to build the integrated service can specify for any component whether encryption is required. The reason is that encryption is expensive in performance terms, and hence it is preferable to encrypt only the sensitive pieces of information rather than an approach where everything is encrypted.

The concern over information integrity, whether threatened by malicious outside activity or the possibility of some sort of transmission glitch, is addressed through the provision of some sort of checksum-type mathematical algorithms that can validate that the information received by a component is the same as the information that was sent. This allows any integrity failure to be detected immediately and recovery actions to be called.

The last area to consider in this section is the control of system tools and facilities. This simply refers to the fact that since the ESB is of critical importance to business operations, the ability to alter the system definitions or deploy new elements of a business service is obviously highly dangerous and therefore needs to be policed rigorously. This requires security protection on access to the system and administration tools in addition to that provided for general ESB. It must be ensured that only personnel with the required level of authorization and clearance can change system definitions or use any operational tools other than purely passive ones.

Key characteristic: Security

• Access control
  ○ User authentication
  ○ Component authorization
  ○ Non-repudiation
	• Information security
	  ○ Privacy (encryption)
	  ○ Integrity checking
	• Tools usage
	  ○ Authorized users only

4.5 *BREADTH OF CONNECTIVITY*

Connectivity is an essential part of ESB functionality. Basic connectivity was discussed in the section on fundamental ESB services. However, beyond these basic requirements, the extent and breadth of connectivity options is a useful best-of-breed differentiator in selecting the most suitable ESB offering.

Firstly, connectivity to the major database brands is valuable. There are situations where a component step in a business service flow involves running a stored procedure within a database implementation, or perhaps just a set of database operations. A connectivity mechanism to access the DBMS would be highly beneficial in satisfying these needs.

Legacy systems are of great importance to any integration solution, due to the fact that a lot of business value is currently contained within legacy operations and components. Perhaps top of the list for these legacy components are leading transaction processing environments such as CICS, IMS and Tuxedo. Any company with an IBM mainframe is almost certainly using CICS or IMS to run a major portion of its mission critical operations. Being able to leverage this value in new ESB-based solutions would provide an extremely attractive return on existing investments. Therefore a best-of-breed ESB characteristic is to offer connectivity for some if not all of these key environments. IBM provides various types of interfaces to help make this as easy as possible for CICS in particular, but the ESB will need to provide functionality to access these interfaces.

Another area for differentiation in connectivity services is that of the other EAI providers. Installations may already have considerable investment in the older, proprietary forms of EAI such as message brokers. Although the benefits of an ESB may well be convincing for new integration deployments, it is certain that companies in this situation will need to have some form of handshaking interoperability between the ESB and the existing implementation. This requires specific ESB support.

Application Servers often drive Web services. However if a company has invested heavily in a particular application server and is now interested in ESBs, some form of more general interconnectivity support for application servers is valuable. It is likely that many of the applications in this environment will not yet have been converted into Web services, but it will still be important for the ESB to be able to drive these components.

There may also be other ‘standard’ forms of communication interfaces that could be listed within the best-of-breed connectivity requirements. COM and CORBA based applications were quite popular in the late 1990s, and connectivity to these is advantageous in many project implementations. Also there is the much more interesting question of support for Microsoft’s .NET architecture. Connectivity to this environment should almost certainly be a feature of any ESB best-of-breed connectivity list.

Finally, many ESB implementations are likely to extend outwards through some sort of Internet-based communications capability. This could be out to a portal or perhaps to some other Internet-based form of service. This implies that a best-of-breed ESB should also have direct connectivity to the more common Internet services, supporting such interfaces as HTTP and even email and FTP services.

Key characteristic: Breadth of Connectivity

• DBMS access
• Legacy systems
  ○ CICS
  ○ IMS
  ○ Tuxedo
• Other EAI solutions
  ○ Message brokers
  ○ Message-oriented middleware
  ○ BPM solutions
	• Application servers
	  ○ WebSphere
	  ○ WebLogic
	  ○ Others
	• Other 'standards'
	  ○ .NET
	  ○ COM / CORBA
	• Internet facilities

4.6 *DEVELOPMENT / DEPLOYMENT TOOLSET*

Major benefits possible when utilizing an ESB include achieving integration aims quickly, easily, cheaply and with less risk. The best of breed solutions will offer tools and functionality in the areas of development and deployment to help achieve these aims. For instance, most integration infrastructures require the use of exotic, expensive skills that are often not available within the company. The intention behind the packaged, standards-based ESB approach is to empower existing IT staff to carry out the necessary integration work instead. This introduces a strong requirement for easy-to-use development tools as well as functionality to assist in the deployment of the newly-built integration solution components.
There are four important areas in the development and deployment toolset:

- Configuration
- Connectivity
- Incremental deployment
- Life-cycle management

Firstly, developers need tools to make the definitions required to support an ESB-based integration solution as easily as possible. As business services are assembled, it is necessary to define aspects like the linkage between the different components, the rules that govern application flow, the transformation mapping required, and the mechanism to be used to link to particular components. GUI-based tools are generally the most productive for achieving these aims, supporting such facilities as drag-and-drop, cut-and-paste and the use of templates. It is important that these tools are as easy to understand as possible to support the desire to utilize existing IT skills.

While connectivity has been discussed in its own section, there is one aspect of connectivity that is absolutely vital to the success of an ESB-based project. Beyond the many standard forms of connectivity supported, there is a need to tie in legacy components that were produced long before such standard forms of connectivity were introduced. For example, many companies still have huge numbers of mainframe-based CICS applications that provide essential services to the business, and these are certainly not going to be replaced or rewritten unless absolutely necessary. In order for an ESB to deliver results quickly with the minimum possible effort, it is therefore necessary for the ESB to be able to accommodate chunks of legacy application code in the integration solution. Indeed,

the objective is to make the legacy code look to other components on the ESB as though it were just another ESB component. The way this is achieved is by providing 'wrappering' tools. The principle behind wrappering is that software is put in place to surround the legacy component in such a way that from the outside the legacy code looks and behaves like any other ESB component whereas from the inside the legacy code is not aware that it is participating in an ESB solution at all.

Not only does wrappering reduce the cost and difficulty of integrating legacy components, but it is also a major contributor to easing the task of migrating to an ESB-based integration. It allows legacy components to be accessed quickly but non-intrusively, allowing more permanent re-architecting of the wrappered components in the future if required. This 'incremental deployment' approach is a critical aspect of any best-of-breed ESB.

Apart from wrappering, the overall SOA approach, location and technology transparency and manageability already discussed all contribute to achieving this goal. By using this approach a company can avoid the business risk implications of a revolutionary, 'big bang' approach to integration but can instead adopt a more controlled and less risky evolutionary approach of staged deployment. In fact, one of the great advantages of an ESB approach is that it offers this incremental deployment capability while at the same time leading to a flexible infrastructure capable of supporting enterprise-wide usage across all business operations.
The final area of tooling is life-cycle management. In highly distributed ESBs that cross multiple locations and companies, great care must be taken when new developments are introduced to the overall implementation.

For example it could be that a company's quality of service practices require it to validate new changes internally for a period of time before they are exposed to partner usage. Therefore tooling that can manage this move through the various different stages of development and deployment will be of value, particularly it is quite likely that there will be a single repository for holding all system definitions which introduces considerable scope for confusion in this case. There must be clear demarcation lines between new definitions and components that are in the development, testing, quality assurance or production phases, and these lines must be enforced by software.

Key characteristic: Tooling

• Configuration
  ○ ESB definition tools
  ○ Flow control definitions
  ○ GUI interface
  ○ Ease of use
• Connectivity
  ○ Wrappers
  ▪ CICS
  ▪ Other legacy systems
	• Incremental deployment
	  ○ Wrappers
	  ○ Location/technology transparency
	  ○ Service-based approach
	  ○ Manageability
	• Life cycle support
	  ○ Development, Test, QA, Production
	  ○ Integrity of phases
	  ○ Process for progression across the phases

5.0 Summary

ESBs offer an excellent basis for integration, delivering a flexible and adaptable environment that enables integration projects to be put in place productively, effectively, and in a staged manner. They leverage many of the developments made in integration technology over the last ten years, packaging them in a standards-based and affordable way. As a result it is likely that ESBs will be used extensively over the coming years, expanding both in numbers of deployments and pervasiveness within the companies adopting them.

It is therefore essential that the greatest possible care is taken in selecting the most suitable ESB for both today and tomorrow. By reviewing the best-of-breed ESB characteristics listed above, and the implications of those characteristics on functionality, the prospective purchaser can reduce the business risk associated with this decision and ensure a successful implementation. Perhaps not all the desired functions are completely satisfied today, yet if the basic architecture, design and implementation of an ESB and its vendor profile fit the pattern of your business needs, then you are ready to start developing and implementing an Enterprise Service Bus in your business.

About Sonic Software Corporation

Sonic Software provides the first comprehensive business integration suite built on an enterprise service bus (ESB). The Sonic product line delivers a distributed, standards-based, cost-effective, easily managed infrastructure that reliably integrates applications and orchestrates business processes across the extended enterprise.

Sonic is the world's fastest growing integration and middleware company and counts global leaders among over 500 customers in financial services, energy, telecommunications and manufacturing. Sonic is an independent operating company of Progress Software Corporation (Nasdaq: PRGS), a $300 million software industry leader. Headquartered in Bedford, Mass., Sonic Software can be reached on the Web at www.sonicsoftware.com, or by phone at +1-781-999-7000 or 1-866-GET-SONIC.