MIDDLEWARESPECTRA

incorporating FINANCIAL MIDDLEWARESPECTRA

Contents

November 2004

2 — **Middleware evolution: from point to point to SOAs and ESBs**
Andrew Bainbridge, IBM Software Group

10 — **The Enterprise Service Bus: infrastructure for all seasons**
Roy Schulte, Gartner Group

18 — **SOAs and Web Services: cheap and simple or fiendish and terribly complicated?**
Tom Welsh, Consultant

26 — **Do we need ESBs?**
Mike Beeston, Maven Associates

32 — **Understanding infrastructure**
Peter Bye, Unisys Systems & Technology

38 — **Enterprise service integration**
Keith Jones, IBM Software Solutions Worldwide

46 — **Middleware, complexity and legacy application integration**
Mark Lillycrop, Arcati

Volume 18 Report 4

Middleware evolution: from point to point to SOAs and ESBs

Andrew Bainbridge
Director, WebSphere MQ and ESB Development
IBM Software Group

Management introduction

Andrew Bainbridge is the Director responsible for WebSphere MQ and Enterprise Service Bus Development at IBM's Hursley Park Laboratory (located outside Winchester, in the UK). As such he is responsible for the development of a number of products and technologies, including:

- ***the WebSphere MQ family (what used to be known as MQSeries)***
- ***the WebSphere Business Integration Message Broker family (what used to be known as MQSeries Integrator)***
- ***related technologies providing transaction management messaging, and Web Services capabilities for the WebSphere Application Server platform.***

In this discussion, Mr. Bainbridge examines:

- ***the evolution and state of today's middleware market, especially as it relates to messaging and Enterprise Service Bus delivery***
- ***why Service Oriented Architectures do not have to be complex but, instead, can be a straightforward and direct evolution of what already exists***
- ***how and why Enterprise Service Bus initiatives are being accepted by customers.***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

Middleware market evolutions

In late 2004/early 2005, I think we are reaching an interesting point in the development of middleware and of the overall middleware market. One aspect, which I see as being increasingly apparent, is a shift:

- **away from simply automating business functions**
- **to adapting and integrating business functions at the business process level.**

In practice this means a concerted transition away from traditional, tightly-coupled applications towards the introduction of more loosely-coupled ones. With this comes an increasing focus on the processing of de-coupled events (see Figure 1.1).

From my own discussions with customers, it is clear that large IT organizations are less and less concerned today about connecting individual applications — or even simple point to point messaging. Much of this work was done in the 1990s with products such as WebSphere MQ (WMQ). IBM is now seeing a focus on a broader integration of applications and related services — with this being followed by the implementation of more complex business processes that cross or bridge between traditional application silos and even application flows. WMQ is increasingly used as a store and forward service on which other integration services are layered.

Why has this happened? In part, this is associated with experience. The past ten years has seen the introduction of an immense amount of middleware infrastructure specifically dedicated to connecting pairs or small clusters of applications. You only have to recall the enthusiasm for 'EAI' (Enterprise Application Integration) as a market sweet spot to understand how and why so much was invested in connecting those applications.

Yet the greater part of this effort was initially oriented to connecting pairs of applications. This worked. But it did not deliver full enterprise application integration — in the sense that all applications that needed to communicate with others could do so easily. The model was primarily about point to point messaging rather than multi-point integration. This did, of course, begin to change with the introduction of products like WBI Message Broker (WBIMB).

Over the last several years, there has been a broadening of requirements. Our customers are increasingly talking about a looser, more flexible coupling of applications and integration of a broader range of systems than was possible with simple point to point messaging. The objective now is to provide a coherent set of application integration capabilities that support an end to end business process and the enterprise, rather than a specific line of business or silo.

Avoiding the continuing costs of 're-facing'

Let me give you a couple of examples of this. One aspect we see frequently is the importance companies attach to being able effectively to re-use their existing IT assets — whether individual applications, underlying transactions or even whole systems — that have grown up over periods of many years. In some cases, when they look back, organizations find that they have 're-faced' — or even re-engineered — a particular application or system many times.

The reason for this is that, as new technologies have appeared, the underlying capabilities of a core application need to be used with other systems. One has only to think of the 'green screen' which moved onto screen scraping and then on to clients and servers before moving on again to browser front ends.

In many respects this was quite inefficient. Looking back, most organizations have realized that particular applications have been re-faced or re-engineered far more times than was necessary.

Recently, I spoke to a large financial services company. It had performed an analysis on one core application and discovered that it had been 're-faced' and re-used in 27 different ways in recent years.

Understandably, most organizations cannot afford to keep doing this. They are looking for standardized ways to exploit key activities, or processes, that work with existing IT assets without having to re-face these every time re-use is demanded. In effect, they want to be able to re-use the core processing activities in as many ways as are needed. They want to avoid having to rebuild the interfaces each time to accommodate other developments or applications. In essence, they are looking to separate the core processing from the interactions — with the interactions being driven from a standardized set of services (about which more later).

Their ideal is to offer a structured, or more standard, way of doing this across their whole range of applications within their enterprise — and not look at each application as a single element. The objective is re-use of the application and infrastructure assets that they already possess — to assist and complement the development of new applications and offerings for their customers.

Let me offer another example, which also happens to be from a financial services company. Over a period of several years its IT people had developed what most would regard as 'custom middleware infrastructure' to interpose between their many different forms of front end and their core mainframe applications.

Today, that company's objective is to move away from its custom middleware towards using Web Services in order to:

- **provide a more standardized way of accessing the original core applications**
- **avoid having to maintain and continue to evolve its own middleware over time.**

By moving to Web Services in particular, and by adopting a Services Orientated Architecture (SOA) in general, their IT people's view is that it is feasible to provide a flexible infrastructure, built upon standards, that offers a base from which they can capitalize on existing core IT assets.

SOAs

This brings me to the subject of SOAs. In my view the difficulties of implementing these have substantially been overemphasized. The trouble with the word 'architecture' is that it can presume, or encourage others to presume, that some form of 'big bang', all-embracing, cross-enterprise approach is needed.

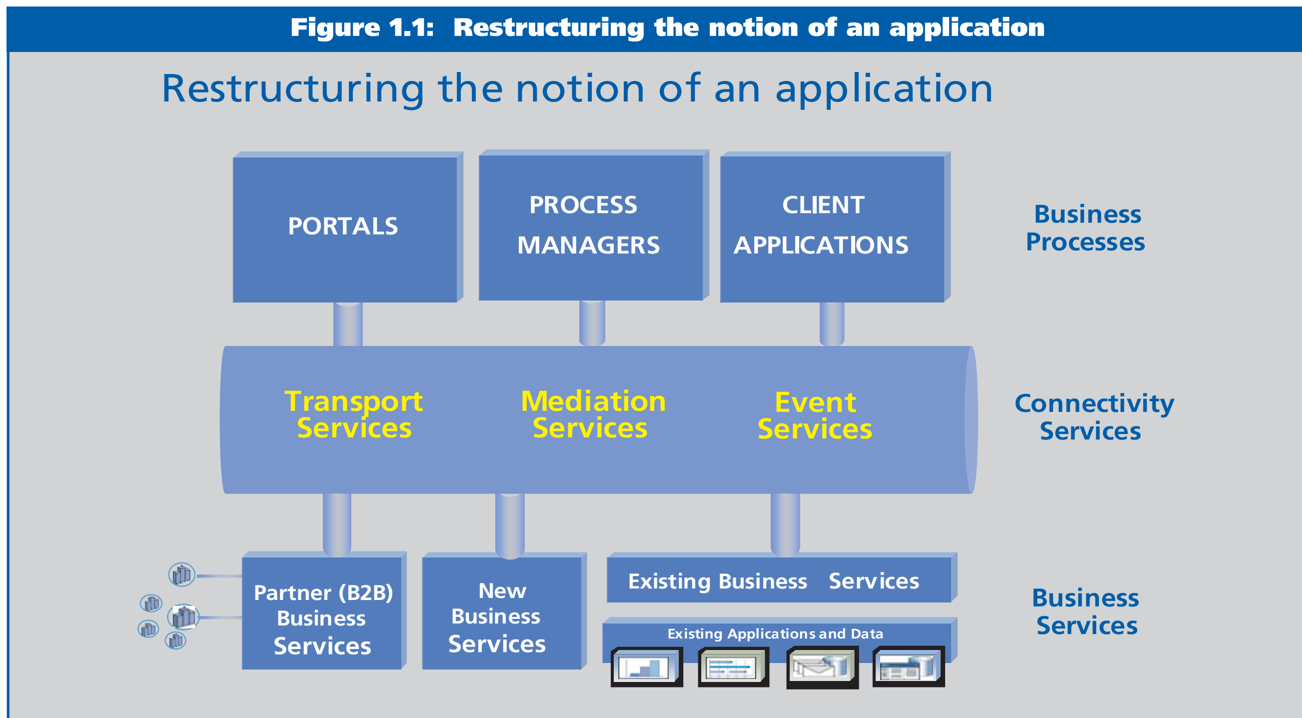
I think there is a real danger of making an SOA into something more complicated than it needs to be. But that does not diminish the fact that an SOA can be a powerful architecture model for:

- **defining software services**
- **then integrating these and using them in conjunction with one another.**

The fact is that IT now has a set of well established standards (the Web Services standards) which support building Service Orientated Architected applications. Web Services are a tool for both re-facing existing applications and building new ones, within a framework of standards that have been broadly accepted and implemented. You can use service orientated technologies at various differing levels.

That said, I do think that a Service Orientated Architecture will be central to the creation of the sort of flexible IT infrastructure that IBM sees many organizations needing if they are to be competitive. Yet, despite sounding as if SOAs require a major effort, to me the concept is essentially simple (even though some do try to dress it up in complex ways and then generate all sorts of clever justifications for why such complexity is vital).

Figure 1.1: Restructuring the notion of an application



Why do I think an SOA is simple? The way I view a Service Orientated Architecture is that it should be about using open standards to represent and expose software assets and services. It really need be no more than that.

Similarly, I think it can be a big mistake to believe that you have to make major commitments of resources to the re-engineering and re-structuring of your IT systems before you can benefit from Web Services and SOAs. We have seen, time and time again, situations where organizations have been able to start to use Web Services as a base for introducing a Service Orientated Architecture. To me, the attraction is that this can be evolutionary and delivered in a step by step manner. There really does not have to be any sort of 'big bang'.

Equally, if you look at the range of middleware capabilities that exists in (say) the WebSphere Application Server platform and in other existing products like IBM's CICS, you can introduce Web Services in a gradual way which enables you to start obtaining some of the benefits quickly. Over time you then adapt and bring in more and more of the pre-existing IT infrastructure and investment.

My point is that you do not need some monstrous, all-embracing architecture before you can start. Experience indicates that you do not have completely to re-engineer your organization (even if you could do this). You do not have to turn your systems upside-down before you can even start to benefit. With the tooling now offered, that facilitates the addition of Web Services capabilities to existing software, you can start swiftly. One of the common elements IBM has observed in many organizations — even in quite small companies — is that you can derive swift benefits from an SOA without having to make substantial investments, not even in a new infrastructure.

I am encouraged that more and more organizations are finding that an SOA lends itself extremely well to new application assembly as well as the looser coupling of existing applications. An SOA explicitly facilitates the re-use of existing IT assets in different and more varied ways. If you move towards the Service Orientated Architecture model, we find that your focus naturally shifts to think about the capabilities you need for application decomposition, re-composition and integration (of both existing and new applications).

Acceptance of loosely coupled processing

Acceptance of this may not always be natural. Many developers think — and, therefore, most applications were built — in a tightly-coupled model. This is the way that most have been trained to think about, and then to create, applications. But this has to change. The loosely-coupled model offers far wider possibilities.

For example, for re-facing existing applications, such as transactions in an existing CICS system, it is possible to use the SOAP protocol to drive those existing CICS transactions — without any need for changing the CICS transactions. This makes those transactions available for integration with, or as part of, any new application you may be building.

Equally, you can develop new applications from scratch. You describe the function of the new application as a service — using WSDL to define that service. In so doing, you have provided an open way for other applications or systems or even enterprises to use that application. There is nothing inherently complex about this.

Of course, core applications that already exist will remain tightly coupled, in terms of their own internal processing and in terms of the internal transaction management they perform. But when you link applications together — to reflect a business process which exists within your organization — then you have to think in terms of looser coupling, or even de-coupled processing between applications, using middleware to provide an assured (and, if necessary, transactional) delivery mechanism.

The trick, from the middleware perspective, is to make sure you have the combination of capabilities both for asynchronous processing — which is increasingly necessary to model the sort of processes organizations typically have — and the operation of internal applications which necessarily are going to remain transactional in most cases. For example, in the case of an existing CICS transaction, you will want to maintain the transaction management and integrity which has always been inherent in that CICS application. But, if you then want to use that transaction as part (say) of a long running work flow sequence that mirrors the actual business process that operates, then you have to be able to do this without, for example, maintaining a lock on a database over an extended period.

So the sort of infrastructure on which we, in IBM, are focused (in our middleware offerings) is increasingly oriented towards:

- **exploiting existing applications**
- **simultaneously being able to couple these and integrate them in either an explicitly asynchronous or a fairly loosely-coupled way.**

What I mean here is that, in the case of the decoupled, you might have two applications that communicate by means of events. The first application might set off an event which the second or another application will react to at some later time — without the two applications ever having to be running at the same moment. Indeed, there are differing degrees of decoupling that you can have — between:

- **the one extreme of the traditional tightly coupled, or peer to peer, integrated applications**
- **the other extreme which brings together applications that work together using decoupled, asynchronous, event-driven mechanisms (like messaging)**
- **somewhere in between where applications communicate using messaging that does not go to either extreme (of being completely decoupled or coupled).**

Web Services, Message Brokers and Enterprise Service Buses

Minimizing change to existing infrastructure matters to customers. They do not want to have to spend on this. Existing middleware products — from WebSphere MQ, through the WBI Message Brokers, to the WebSphere Application Server (WAS) — have complementary capabilities, but the common dimension is that they are all relevant to building an Enterprise Service Bus (Figure 1.2).

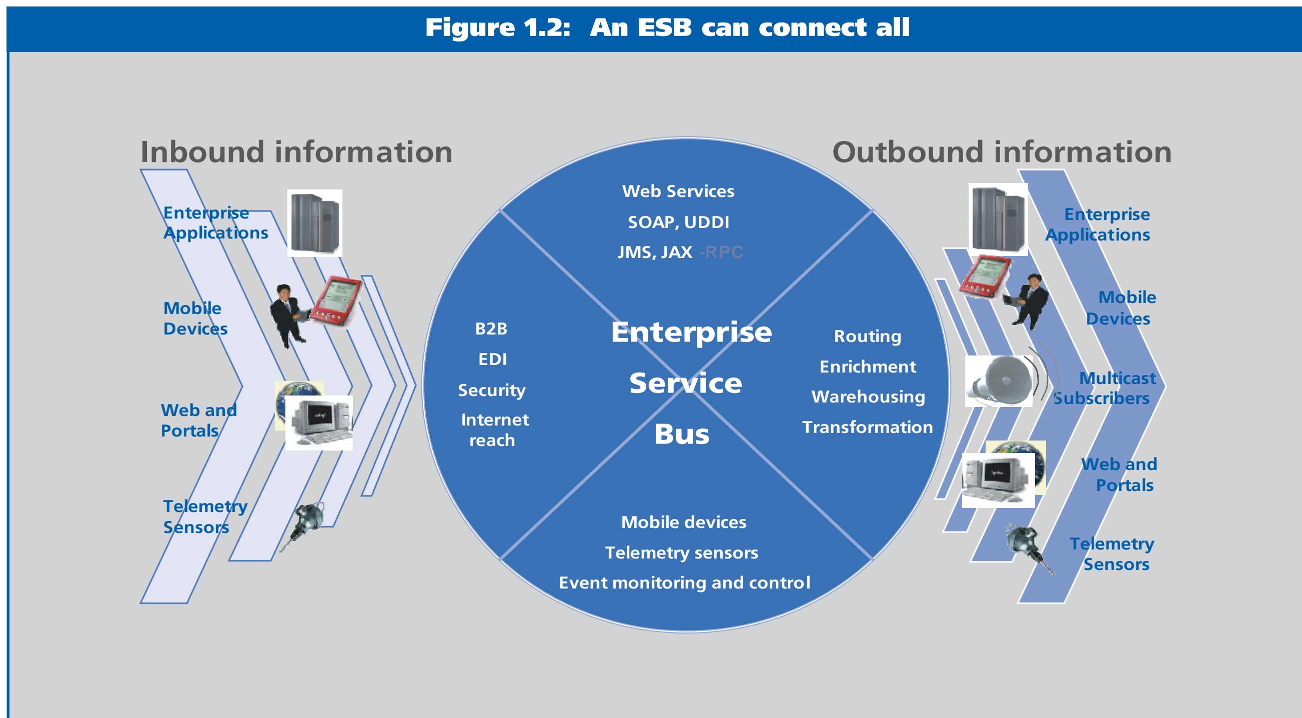
What is an ESB trying to do? An ESB should focus upon delivering increased flexibility (Figures 1.3-1.4 demonstrate this).

I touched earlier on the notion of introducing greater flexibility in the use of existing IT assets — whether to implement different business models or to integrate existing systems or to implement some sort of business transformation. In this context, the key characteristics of an ESB include:

- **support for a range of transport services and communication paradigms (with mapping between them)**
- **mediation services, essentially the notion of doing some form of processing on a message or a service invocation in-flight (for example, 'within' the network)**
- **tooling, for building the functionality you wish to deliver and then for managing it once it is deployed**
- **access to existing systems (there are few, true green field sites and inevitably, when building an ESB solution, it is necessary to accommodate and integrate or interact with some form of existing system — whether CICS, an existing WMQ infrastructure or a database).**

In looking to implement an ESB solution, what you do will

Figure 1.2: An ESB can connect all



depend very much on what existing systems, what existing communication infrastructure and what sorts of capabilities (for example, by way of support for mediation) an organization already possesses. In some cases it may be possible to build an ESB solution in conjunction with existing systems — purely using WMQ and/or WBIMB. Alternatively, if there is already use of Web Services standards (perhaps without an existing messaging or message broker infrastructure), it may be appropriate to use the Web Services capabilities of an application server like WAS to implement an ESB.

The sort of flexibility we are talking about implies the need for a combination of capabilities. These could include:

- **some form of universal connectivity between heterogeneous systems**
- **looser coupling of more granular business components**
- **the ability to monitor the flow of transactions and processes across a business.**

The objective is to combine the disciplines of an SOA with the capabilities inherent in middleware (as generally found in message brokers and application servers). It is this combination of capabilities which produces, logically, what is now called an Enterprise Service Bus.

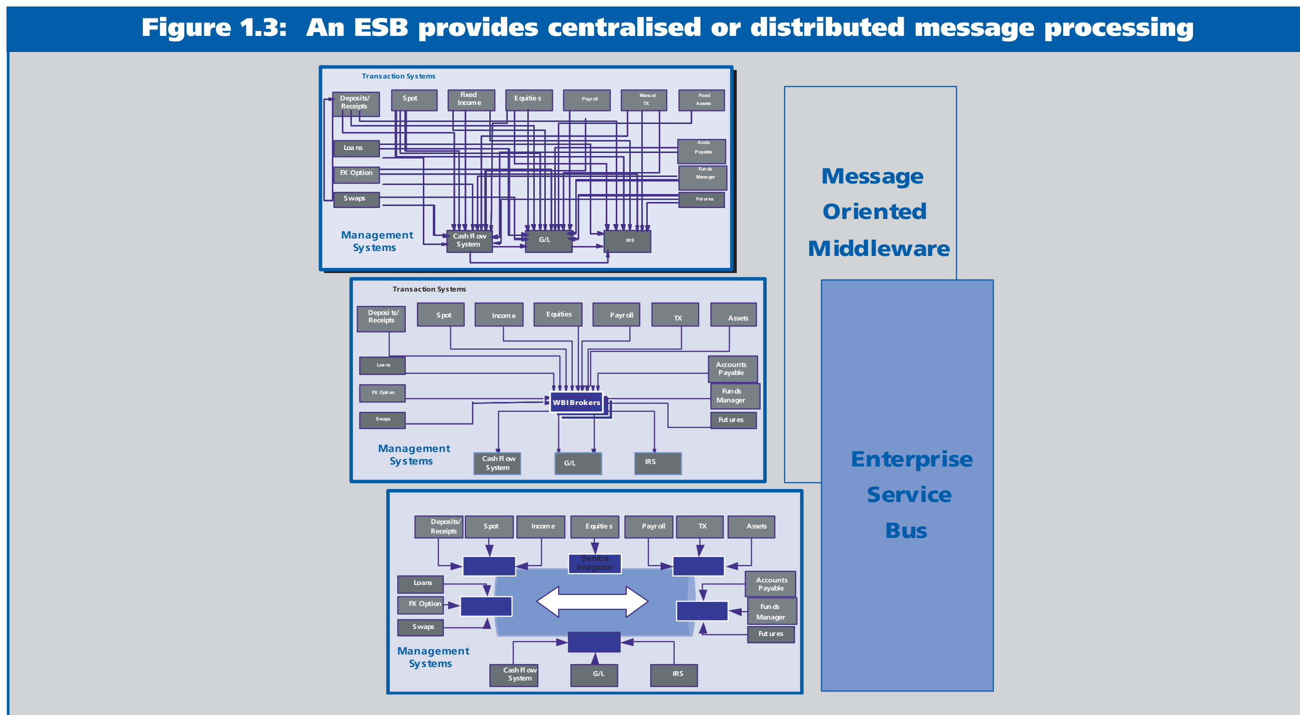
The critical point I wish to make is that the concept of an ESB is more of an integration pattern or an architecture pattern — rather than a unique product category. It is the pattern which describes the set of capabilities which can connect and integrate an enterprise's IT assets — which, necessarily, may involve switching messages between heterogeneous platforms and environments over a consistent backbone that carries events, messages and even service delivery.

So an ESB is a collection of capabilities, not a particular product. An ESB will use the capabilities which are already available in the sort of middleware that enterprises typically possess today, as well as exploit those new capabilities which will be delivered over the course of the next several years.

Across a range of products (whether it is messaging, or message brokers or application servers), there is a set of capabilities which are complementary and with which you can gradually create an Enterprise Service Bus implementation — or, if you prefer, can build a set of enterprise services which realize an ESB pattern. The important point here is that there will be many different varieties of ESB. These will vary, depending on the precise needs, and on what is already available, within an organization.

So, for example, in one case one organization may be interested in providing services to the outside world using Web Services. It would need, therefore, a control point through

Figure 1.3: An ESB provides centralised or distributed message processing



which services from its existing internal systems can be manifested to that outside world — and through which access can be controlled. The approach for this would be quite different in a second organization that wanted to establish a bus architecture internally to integrate its internal systems and perhaps to provide new business logic based on the transformation of messages or interaction of messages between those internal systems.

If you look closely, you can see that many organizations have built ESBs already — using the likes of WebSphere MQ, WebSphere Business Integration Message Broker or the WebSphere Application Server, and even (in a more limited way) other vendors' products. This demonstrates, in my view, that there is no single ESB pattern. An ESB can be implemented in a variety of ways using the capabilities in a variety of products. Equally, it is incorrect to suggest that you need all the middleware products — from IBM or any another vendor — before you can start to implement an ESB.

For example, yet another financial services company customer I have met with recently has used a capability (that IBM calls the Web Services Gateway, which is part of WebSphere Application Server Network Deployment Edition) to create a control point to provide access to existing CICS transactions by using Web Services. This is not unique.

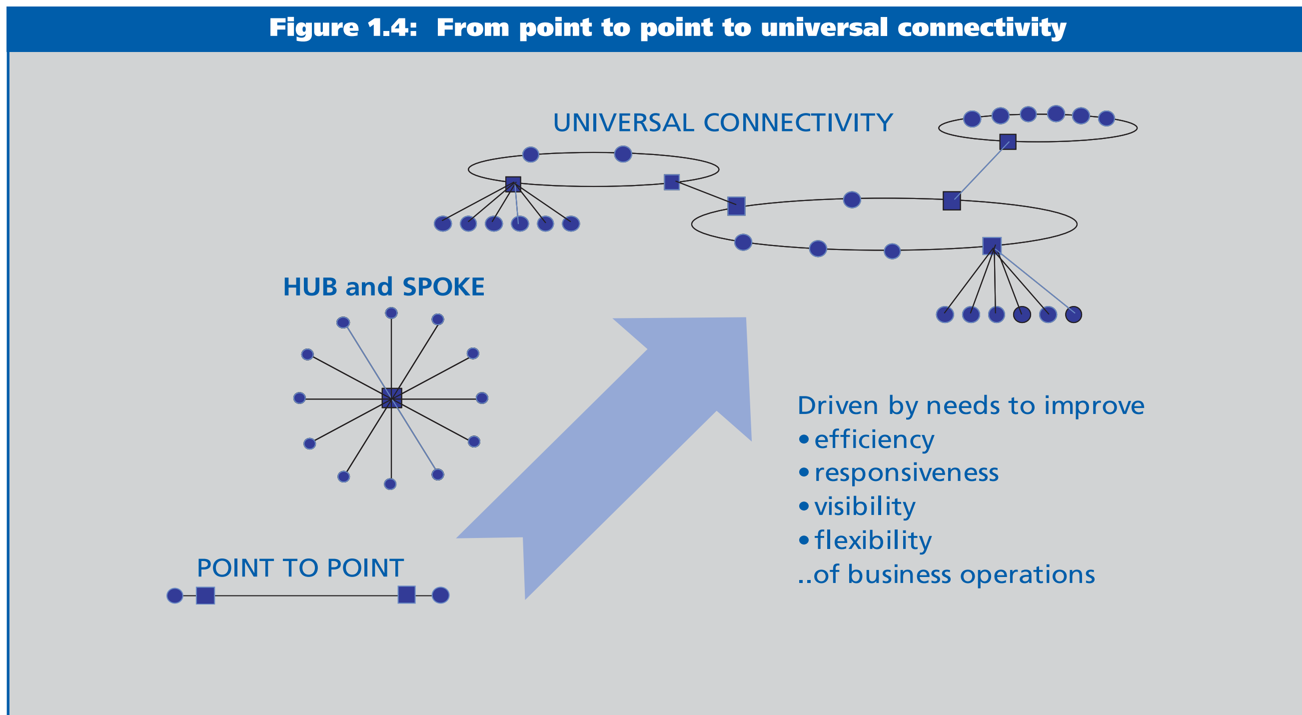
Using Web Services, this organization has combined new and existing technologies and applications. Using that combination of these products, it is possible right now to securely and reliably mix Web services with existing CICS COBOL, WebSphere MQ, Java and .Net applications — with little or no change to the existing applications or middleware infrastructure.

Indeed, my own perception is that we will see Web Services being used more and more as a common link in implementing ESBs. That is why we have been adding greater and greater Web Service capabilities to WebSphere MQ, WBI Message Broker and the WebSphere Application Server — so that each (or all) of these can be used to deliver an ESB. In 2005, for instance, WBI MB will extend its support for the definition of Web Services using WSDL.

Where will IBM be in 2-3 years' time?

Over the next 2-3 years there is going to be a great deal of evolution of the sorts of capabilities that already exist in the various products we offer today. For example, the recently launched WebSphere Application Server Version 6 will add significant new capabilities for messaging — with a new all Java-messaging engine that will be tightly integrated into the application server platform. This is relevant to building reliable messaging-based solutions, something where some application servers have not had a great deal of capability in the past. Indeed this new messaging capability in the

Figure 1.4: From point to point to universal connectivity



Application Server will inter-operate smoothly with existing WebSphere MQ-based messaging infrastructures.

In the case of Web Services there will be enhancements to the capabilities already offered. For example, Version 6 of IBM's WebSphere Application Server supports Version 3 of the UDDI specification as well as the Web Services transaction specification (for the co-ordination of applications).

Similarly, in WebSphere Business Integration Message Broker, IBM will be providing a number of enhancements that extend its Web Services support. There will be more sophisticated modeling and handling of SOAP messages, as well as greater flexibility in generating WSDL to define services.

From this it should be apparent that there is a common theme. This is that all of these products — separately and in conjunction — are being broadened and deepened so that their particular strengths are complemented by the addition of new capabilities that support the implementation of ESB patterns. This gives the flexibility about which customers talk, and which they need.

So, from an IBM perspective, we are intent on offering a range of complementary products, each of which can be the basis for an Enterprise Service Bus. Most of all, progress can be rapid. An SOA does not require total re-engineering of an enterprise. Organizations can readily exploit their existing middleware infrastructure and applications in building ESB solutions.

Management conclusion

There is an often expressed view that there is nothing new in computer science. If you look over a period of a number of years, through the evolution of technologies such as DCE and then CORBA and then Web Services, one of the more unlikely results of the evolution of middleware is that, over time, it (middleware) has actually become less — rather than more — prescriptive.

As Mr. Bainbridge argues, today you can use Web Services in ways that do not dictate the way in which applications are internally structured. This lends itself extremely well to integration of both applications and processes — because it does not mandate in any way how existing or new systems must be engineered. Using Web Services does not require any form of fundamental re-engineering of what you already have or prescribe how you should write new applications. Similarly, an ESB can be implemented with a range of products, many of which are already in use.

The Enterprise Service Bus: infrastructure for all seasons

Roy Schulte
Vice President and Distinguished Analyst
Gartner Group

Management introduction

Commercial Enterprise Service Bus (ESB) products have only been available since 2002, but they are poised to have a huge impact on software strategies during 2005 and 2006. ESBs are a new kind of middleware that combines features from several previous types of middleware into one package. Many observers and vendors describe ESBs as the enabling technology for Service Oriented Architectures (SOAs), which indeed they are.

But ESBs are more than this. ESBs will also be the backbone for Event Driven Architecture (EDA) solutions and for connecting into legacy applications. A technology upheaval like this is rare, and worth understanding in more detail.

In this comprehensive analysis of ESBs, what they are and where they fit, Roy Schulte (arguably the doyen of middleware analysts):

- ***starts by offering a specific definition and description of ESBs***
- ***follows by comparing and contrasting ESBs with previous generations of middleware***
- ***categorizes some 19 ESB products, between three principle varieties***
- ***positions ESBs within the context of related software products and larger trends***
- ***concludes by examining the likely future evolution of ESBs.***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

Definition

An ESB is a Web Services-capable middleware infrastructure that supports:

- **intelligently-directed communication**
- **mediated relationships**
- **loosely coupled and decoupled business components.**

To elaborate: ESBs are designed to work with software that is organized as a set of coarse grained components rather than as large monolithic application programs. Components are:

- **modular — to enable a 'divide and conquer' approach to big problems**
- **encapsulated in accordance with the familiar 'black box' principle which shields an external developer from having to understand the potentially complex internals of any given module.**

The 'implementation' (application code and data) is separated from the interface definition. An ESB must, therefore, support metadata for documenting component interfaces and message schemas. ESBs commonly use Web Services Description Language (WSDL) and XML Schemas for this purpose, although other mechanisms may also be supported.

Web Services are the most important middleware standards for interoperability today, particularly WSDL and the Simple Object Access Protocol (SOAP). Any ESB must support WSDL and program to program communication using SOAP/HTTP, along with associated foundational standards such as XML, HTTP and TCP/IP. An ESB may optionally support other standards such as the Java Message Service (JMS) and protocols such as SOAP/SMTP or a proprietary messaging protocol, such as WMQ or the TIB.

Components that interact in a bi-directional manner are considered to be involved in a SOA. An ESB must support request/response communication and some way to allow or enable more-complex tailored microflow message exchange patterns (Figure 2.1).

At the same time, ESBs must also support one-way message delivery to enable EDA notification among business components. The better ESBs will also support reliable, store and forward delivery as well the publish and subscribe mechanism for supporting demanding EDA solutions.

Intelligently directed communication is another essential. All ESBs are capable of applying some type of intelligent routing and address indirection to make SOA clients and EDA sender programs more independent of (and so less coupled to) their respective SOA servers or EDA event consumer programs:

- **for an SOA, this requires some type of name space or service directory which resolves a logical service name to a specific server implementation at run time**
- **in an EDA, this requires some repository (ideally part of the same metadata facility used for SOA interactions) that maintains the message schemas and the rules for subject-based, property-based or content-based message routing at run time.**

An ESB may also supply an optional (separately packaged or integrated) business service repository (for example, implementing the Universal Description, Discovery and Integration (UDDI) standard) to be used by developers to register and then discover services or event descriptors at development time. If the ESB does not include this, then architects in a large development environment can implement a complementary third party repository for this purpose.

Mediation: an ESB, like a MOM, interposes a software intermediary between the sending program and the receiving program — thereby making the relationship 'connectionless' (more precisely, there is a connection between the client/sender and the ESB and another connection between the ESB and the server/receiver). This architecture is essential to intelligent routing, address indirection and other optional value added ESB services. A direct, point to point, connection between sender and receiver would make most ESB features impossible to apply.

Furthermore, using the metadata which documents service interfaces and message schemas, an ESB intermediary engine or layer can transform messages and documents, either through its own facilities or using an optional third-party accessory (for example, an XSLT engine).

Almost all ESBs provide some form (or forms) of additional value-added communication, management and security services — although these are not required by definition. Examples of such features include:

- **message validation**
- **load balancing**
- **fail-over**
- **logging**

- **performance and availability monitoring**
- **fault management**
- **configuration management**
- **accounting**
- **service-level agreement (SLA) tracking**
- **metering and billing.**

ESBs versus previous generations of middleware

Some architects have had the goal of using a business component architecture in distributed applications since the middle of the 1980s. Several previous generations of middleware came to market to enable this, but all middleware prior to ESBs had significant limitations that impeded their usefulness for this purpose (Figure 2.2).

Remote procedure calls (RPCs) — such as Sun's Network File System (NFS) RPC and the Open Software Foundation's (OSF's) Distributed Computing Environment (DCE) RPC — and object request brokers (ORBs) — such as Microsoft's Component Object Model (COM) and the Object Management Group's Common Object Request Broker Architecture (CORBA) — were very good at implementing SOA applications. They enabled business component principles through encapsulation and documented interface definitions (although only a minority of application developers fully appreciated the possible benefits of this). However, RPC and ORB coupling between software components was still too tight for large, rapidly evolving applications. Furthermore, there was almost no support for asynchronous communication or EDA, and major vendors were unable to agree on common standards for protocols, metadata and programming models.

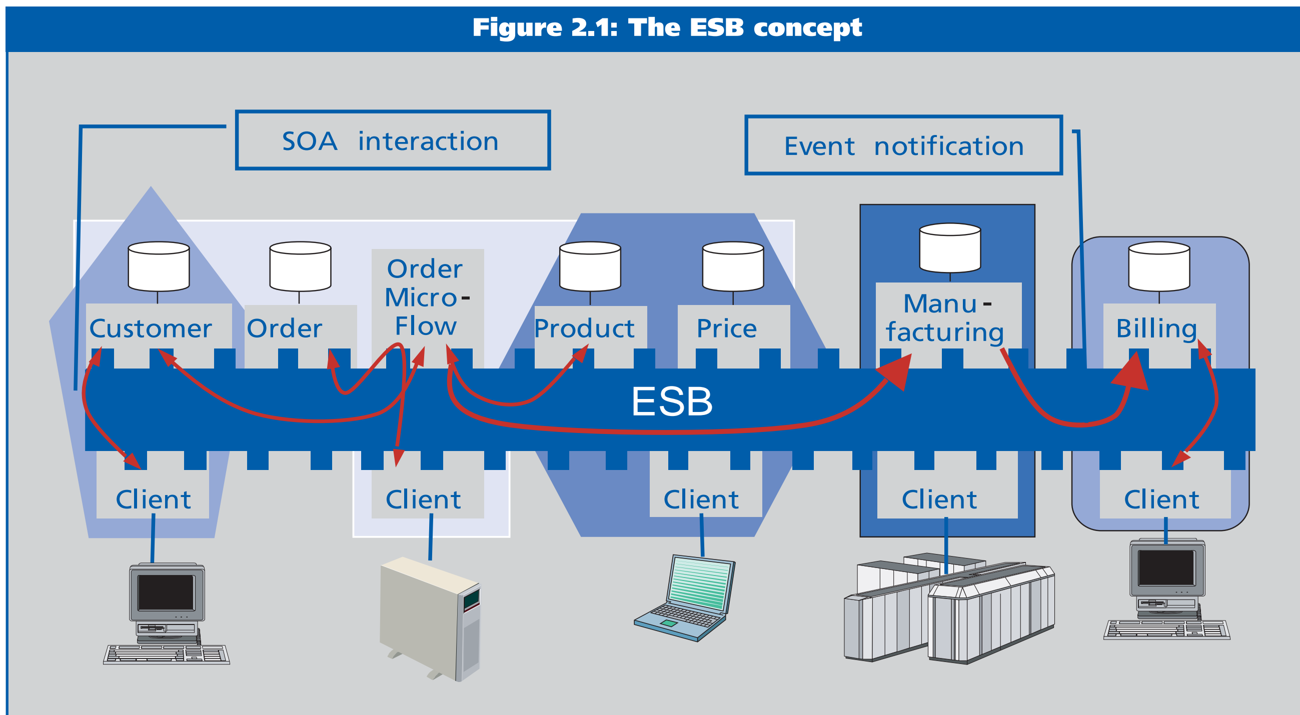
MOM products were, and are, more flexible than RPCs and ORBs because of their decoupled, dynamic nature. Highly skilled development teams have been able to implement a mix of SOA and EDA applications on a MOM backbone. But this requires the invention of a custom programming model plus a metadata mechanism because, unlike RPCs and ORBs, MOM products do:

- **not support the notion of a service or business component**
- **not have any internal mechanism to document interfaces or message contents.**

Moreover, as with RPCs and ORBs, vendors differed in their protocols and programming models. Thus MOM standards did not get very far.

Web Services began as an Internet-friendly RPC, meant for SOA. Their greatest strength remains that virtually every vendor has agreed to support at least the basic protocol and metadata standards (SOAP, WSDL and XML). Web Services implement loosely coupled SOA interactions effectively. However, standards for messaging and notification

Figure 2.1: The ESB concept



are just being developed so Web Services do not support EDA well, as yet. Moreover, basic Web Services are essentially a point to point communication mechanism with no inherent support for intelligent routing, service indirection or mediation for value-added services. Web Services standards (for example, SOAP) have explicit provisions that allow an architect to add mediation software to provide such services, but simple SOAP runtime platforms do not implement such features and standards do not specify their characteristics in any detail.

ESBs are a superset

What attracts about ESBs is that they are a functional superset of previous generations of middleware:

- **like RPCs and ORBs, they separate the interface from the implementation and enable loosely-coupled SOA interactions well**
- **like MOM, they support decoupled EDA notifications, qualities of service (for example, reliable messaging), scalability and dynamic reconfiguration**
- **wherever possible, ESBs adopt the Web Services standards, as defined by W3C, OASIS and WS-I.**

Because ESBs interpose a software intermediary, they are naturally superior to any of the previous forms of middleware as a platform for applying the wide variety of additional features that are helpful for distributed computing.

Why call it an ESB?

The 'Enterprise' in ESB refers to the fact that an ESB will generally be installed and managed within one virtual enterprise — one organization and possibly some of its customers and suppliers (third-parties). An ESB deployment will generally have a single virtual name space and a single image for administration, even if there are hundreds or thousands of physical ESB nodes.

The 'Service' carries three meanings:

- **first, the ESB supports SOA applications (as well as EDA applications)**
- **secondly, ESBs support Web Services standards**
- **thirdly, the ESB provides a variety of routing, security and management support services which facilitate communication and the relationships among business components.**

'Bus' refers to its support for 'plugability'. The most important benefit of an ESB network is its ability to add, change, move or delete business components dynamically — without disrupting other clients/senders or servers/receivers.

Figure 2.2: Looking back, and forward — the ESB heritage

	TCP/IP	RPC, COM CORBA	MOM	Web Services October 2004	ESB
Documented interfaces and events		Y		Y	**Y**
Service and event registration and discovery		Y		Y	**Y**
Industry standards	Y	1/2	1/2	Y	**Y**
Qualities of service		1/2	Y	1/2	**Y**
Management		1/2	1/2		**Y**
SOA Interactions		Y	1/2	Y	**Y**
Messaging and notification			Y		**Y**

Positioning ESBs

At Gartner, we consider an ESB to be a type of middleware: it is real software. In the next section, I identify:

- **14 commercially available general purpose ESB products**
- **2 special-purpose ESB products**
- **3 more ESBs that are expected to ship from major vendors during the next three years.**

With minor areas of exception, all of these products conform to the definition of ESB given above.

However, I know of many Gartner clients who have developed their own ESBs and near-ESBs by combining custom code with some off-the-shelf software — such as a MOM, a plain Web Services platform and a security or directory product. Home grown ESBs were especially helpful to leading edge developers in the early days, before commercial ESBs shipped. They can still be a viable alternative in some circumstances where none of the commercial ESBs meets unique local business needs. Gartner believes that most IS groups will buy rather than build their ESB, especially by 2006 and beyond.

Some architects consider an ESB to be a design pattern rather than a set of software modules. This perspective can be useful especially in the early stages of application design, for planning requirements for upgrading a software infrastructure and for developing an IT strategy. However, this perspective can be confusing because an ESB can be used for implementing a wide range of application design patterns, such as SOA and EDA (so an ESB would be a middleware pattern for implementing other application architecture patterns). Moreover, the industry needs the term 'ESB' (or a replacement) as a label for a particular class of middleware products. Without this term, what would we call the products we are about to buy?

ESB products could have been called 'distributed computing environments' (DCEs), but that term was used by OSF more than a decade ago to apply to one of ESB's antecedents. Sonic suggests (in jest) that the overall ESB category be called the "intergalactic service cloud". Cape Clear envisions that the ESB term may evolve to something like the "service delivery platform" in the future.

Some vendors (like Blue Titan, Microsoft and webMethods) have referred to their ESBs as 'fabrics', a richly descriptive and apt term. Some ESBs have also been called 'Web Services Brokers' or 'Web Services Management' (WSM) products. The latter term (WSM) is particularly confusing because some WSM products are ESBs by our (Gartner) definition while others are not because they do not provide a critical mass of communication and mediation functions.

The vision of ESBs described in this analysis is based on the belief that communication — sending data between programs — must be at the core of a holistic middleware infrastructure. Communication is the enabler for:

- **service substitution (dynamically swapping one implementation for another)**
- **content-based routing**
- **publish and subscribe**
- **failover**
- **load balancing**
- **other important services.**

A business component infrastructure which supports Web Services plus SOA plus EDA will first and foremost be a communication mechanism. Other important functions can be added because the platform and the hooks will be there.

Market forces beyond the control of any one person will determine if 'ESB' remains the primary label for this kind of middleware. However, more than semantics is involved in this discussion.

The packaging of ESBs will be an increasingly important issue, both to user enterprises and vendors. Gartner believes that ESBs will be widely embedded in other, larger kinds of middleware — and that most ESBs will be bought in this form rather than as standalone products. Yet standalone ESBs will be a valuable part of the enterprise infrastructure architecture in many situations, especially where there are heterogeneous application servers.

Nevertheless, most enterprises will use multiple ESB products, whether standalone or embedded. This will probably work fine where there is a unified infrastructure strategy and good guidance from an integration competency center. An enterprise can have one logical enterprise nervous system consisting of multiple physical ESB deployments. However, those enterprises that fail to co-ordinate their ESB strategy effectively will probably produce high software and support costs as well as inflexible and hard to maintain applications. Gartner expects that many application servers and most or all application platform suites (APS products) will bundle an ESB (Figure 2.3). We also expect that some large packaged applications will be delivered with ESB-like middleware as a standard part of the product.

Furthermore, Microsoft has already indicated that its Indigo software (which we consider to be equivalent to an ESB)

will be implemented as a subsystem within the Longhorn version of the Windows operating system. This means that all new Windows servers and most or all Windows clients will automatically have an ESB equivalent by 2007 or so.

Sun's Project Kitty Hawk is aimed at delivering 'Java Business Integration' which includes ESB-like features in Sun's Java Enterprise System. There is also increasing overlap in function from the XML appliance vendors such as Cast Iron, Data Power and Sarvega, although we do not consider the products from these vendors to be hardware ESBs (at least, not yet). These XML appliances may ultimately complement, rather than compete with, the software ESBs.

ESB variations

There are many variations of ESB. These can be grouped into three main categories:

- **ESBs focused mostly on SOAP/HTTP**
- **multi-protocol ESBs, which support SOAP/HTTP plus other protocols**
- **ESB extenders that reach outside firewalls to desktops and client devices.**

SOAP/HTTP ESBs

Some ESBs, particularly the 'Web Services Brokers' or 'WebServices Management' tools, originally were created to work specifically with SOAP/HTTP. They still use SOAP, but some now also support alternatives to HTTP as a transport. It was clear from the early days of Web Services that plain, unbundled Web Services platforms — such as Apache's Axis or Systinet's Web Applications and Services Platform or similar platforms that are bundled into Microsoft's SOAP Toolkit or a Java Application Server — are sufficient only for simple applications.

SOAP/HTTP ESBs implement the notion of an intermediary to intercept messages within the local SOAP runtime platform, or in an intermediate proxy server, to provide value-added management and security services and sometimes intelligent routing and service indirection. Some of the early SOAP/HTTP vendors also supplied Web Services development tools.

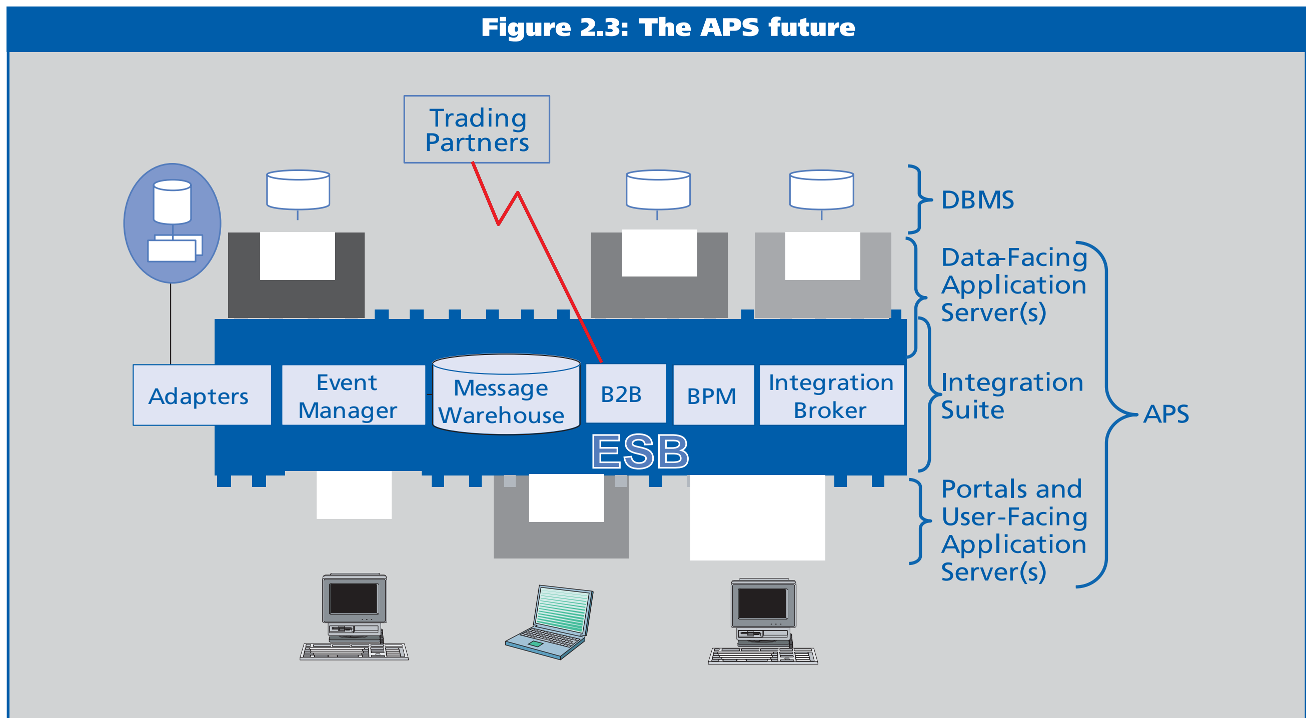
Examples of such ESBs include:

- **Blue Titan's Network Director**
- **Cape Clear's 5 Server**
- **Digital Evolution's DE Management Server**
- **Hewlett-Packard's Talking Blocks Service Oriented Architecture**
- **Primordial's Web Services Network.**

Multi-protocol ESBs

Multi-protocol ESBs support SOAP/HTTP and additional

Figure 2.3: The APS future



protocols and programming models. Almost all of these implement messaging features, such as guaranteed delivery and publish and subscribe, and follow the JMS standard (because the Web Services standards are incomplete is those areas). These products also provide various value-added services.

Examples of such ESBs include:

- **BEA's QuickSilver (a future product)**
- **Fiorano Software's ESB**
- **IBM's WebSphere ESB (a future product)**
- **IONA Technologies' Artix**
- **Microsoft's Indigo (a future product)**
- **PolarLake's JIntegrator**
- **SeeBeyond's eInsight ESB**
- **Software AG's EntireX**
- **Sonic Software's ESB**
- **SpiritSoft's Spiritwave**
- **webMethods Enterprise Service Platform**
- **WebV2's Process Coupler.**

ESB Extenders

These products are not designed to be the enterprise server-to-server backbone. Rather, they are best understood as extensions to other ESBs, application servers or MOM.

ESB extenders are highly scalable to reach large numbers of desktops (and servers) across a WAN and the Internet. They are easily configured with 'zero install effort' clients when desired. They can use HTTP tunneling to penetrate firewalls where appropriate.

Examples of such products include:

- **Kenamea's Web Messaging Platform**
- **KnowNow's Event Routing Platform.**

Figure 2.4: ESBs at AIWS Conference

Roy Schulte will be discussing ESBs in greater detail than this analysis makes possible at Gartner's Application Integration and Web Services conference, from November 15-17, 2004 in Orlando, FL (http://www.gartner.com/us/aiws). This conference will also cover other current middleware and application integration topics, including:

- **the future of application integration**
- **case studies: the good, bad and ugly of real experiences**
- **analysis of the growing role of business process management**
- **positioning vendors, understanding strengths and risks**
- **update on Web Services, official standards and actual use**
- **does integration technology pay, and how?**
- **the role of the integration competency center.**

ESB futures

SOAP/HTTP ESBs and multi-protocol ESBs largely will converge by 2006. The Web Services standards have become more open to alternative protocols beyond SOAP and HTTP. Standard SOAP documents can be sent over private protocols when HTTP is not scalable, reliable or efficient enough. Furthermore, WSDL can be used to describe an official Web Service, even if SOAP is not used. This enables 'pure' SOAP ESB vendors to implement other protocols without violating their principle of standards compliance — and it lends credibility to multi-protocol ESB vendors' strategies.

Most SOAP/HTTP ESBs will become multi-protocol ESBs. Rather than developing all of the new features on their own, some HTTP-focused ESBs will merge with multi-protocol ESBs or with MOM.

Coming from the other direction, some of the multi-protocol ESBs will acquire or merge with a SOAP/HTTP-centric ESB, even if their multi-protocol ESB already has some basic SOAP and HTTP features (because SOAP/HTTP ESBs generally have better support for Web Services standards and development tools). SOAP/HTTP ESB vendors are attractive acquisition targets. For example, in 2003 Hewlett-Packard acquired Talking Blocks and webMethods bought The Mind Electric. This acquisition trend likely will continue.

By 2006, ESB vendors will compete mostly on the strength of their value-added services and performance, rather than on their standards compliance. The better ESBs will offer developers a choice of programming models, communication patterns and application programming interfaces in a combined, SOA- and EDA-capable infrastructure.

In some cases, the ESBs also will merge with end point Web Services platforms.

ESB vendors increasingly will shield application developers from having to know what protocol is being used. Some advanced ESBs will select a protocol dynamically, or will transparently route across multiple protocols.

Integration suites

A comparison between ESBs and integration suites is especially important because of the overlap in the purpose and design of the two types of products. Integration suites are functional supersets of ESBs. They implement all of the ESB characteristics described above, including a mediation architecture and support for intelligently directed communication, SOA and EDA.

However, integration suites have additional features, including:

- **business process management engines (BPM)**
- **adapters**
- **adapter development tools**
- **more sophisticated transformation tools**
- **business activity monitoring (BAM) tools**
- **business to business features, such as trading partner management.**

Naturally, these integration suites cost a lot more than the ESBs. Early integration suites were built on a MOM core because Web Services standards were not available at the time. During the past four years, virtually all of the integration suite vendors, including :

- **BEA**
- **IBM**
- **Microsoft**
- **SeeBeyond**
- **Tibco**
- **webMethods**
- **Vitria**
- **and others**

have added increasingly rich support for Web Services while maintaining their original MOM technology for:

- **scalability**
- **performance**
- **backward compatibility reasons.**

As these vendors implement ever-more-native Web Services features, their internals will have more in common with the products that were designed from conception as ESBs. In the meantime, some integration vendors offer low-cost subsets of their suites which they refer to as an ‘ESB’, even though such products are generally more complex and powerful than products designed as pure ESBs.

Coming from the opposite direction, some of the early ESB vendors, such as Fiorano Software and Sonic Software, have added some or all of such integration features to their ESBs. These vendors now can offer either type, ESB or integration suite, on their respective technology bases.

Management conclusion

There will be an ESB — or several ESBs — in the future of most large enterprises because most enterprises will make increasing use of SOAs and EDAs in their new applications. ESBs will be ever more essential as the complexity of business processes increase and the boundaries between application systems blur:

- ***more than half of all large enterprises will have an enterprise service bus running by year-end 2006 (0.7 probability)***
- ***one-third of all application development projects in 2007 will use an ESB (0.6 probability).***

But, as Mr. Schulte makes quite clear, do not let a superficial examination — or a common category name (‘ESB’) — fool you. Some ESBs are remarkably more powerful than others in their architecture and capabilities.

SOAs and Web Services: cheap and simple, or fiendish and terribly complicated?

Tom Welsh
Consultant

Management introduction

While general excitement about Web Services continues to persist, the IT industry has a new rallying cry: Service Oriented Architecture (SOA). Web Services, though still new and immature, are straightforward enough from the technical point of view. But what is an SOA, once we move beyond the slide projector and the analysts' reports? Is it a blueprint for turbocharging Web Services or an entirely new approach to automating the enterprise — or just a more effective way of marketing existing products and services? And, as for Web Services, are they the cheap, simple and user-friendly technology we were led to believe? Or are they growing uncontrollably, like Jack's magical beanstalk (in the fairy tale)?

In this analysis, Tom Welsh reviews the evidence in an attempt to make up his own mind as to whether Web Services are cheap and simple — or have grown fiendish and terribly complicated.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

The evolving vision of Web Services

The Web Services concept has been transformed since its earliest days in 1998. Originally conceived as a fast, cheap and convenient way of enabling applications to communicate across a network, it has gradually evolved into one of the most ambitious IT architectures ever seen. In the February 2003 **MIDDLEWARESPECTRA**, I made the following observation:

"It is easy to criticize mature, fully operational standards like CORBA and J2EE for being too complicated. Web Services looked light as thistledown by comparison, when they were merely an exciting concept on a slide projector. But CORBA and J2EE began small, too. Then they grew, year by year, as practical experience indicated the need for changes and additions.

"David Young, chief evangelist of Lutris Technologies, has expressed what we might call the 'back to Nature' argument very clearly. "The problem with CORBA was it was a little too big... [It was] boiling the oceans, being everything to everybody. SOAP is a much simpler notion of implementation independence. SOAP is absolutely key to creating a bold, beautiful, interoperable world."

"These words are becoming increasingly ironic, as IBM, Microsoft, OASIS, W3C, WS-I and others stack specification on specification, profile on profile, and vision on vision. It will be impossible to make any reliable statements about Web Services in general, until we reach a consensus about what Web Services really are."

In the intervening 21 months, Young's remarks about CORBA vis-a-vis Web Services have grown still more ironic. There are now far more specifications related to Web Services than there ever were for CORBA (although many more production systems have been built with CORBA).

Without making a full-time occupation of tracking Web Services standardization efforts it is hard to be sure of not missing any; but at the last count I identified slightly more than 100 specifications directly or closely related to Web Services. Of these, about 45 had been submitted to various standards consortia, and a relatively small number (20 or so) had been adopted as formal standards. Twelve have been adopted by OASIS, 7 by the Java Community Process (JCP), and 2 — XML Digital Signature and SOAP 1.2 — by W3C.

Jong-Hong Jeon, a Senior Member of Research Staff at ETRI in Daejeon, South Korea, has done a magnificent job of documenting most of the specifications and other documents relevant to Web Services. A PDF diagram, suitable for printing, can be found at http://blog.webservices.or.kr/hollobit/data/WS_Advanced-v40.pdf (Figures 3.1 and 3.2). (As can be seen, this diagram needs to be printed on a large sheet of paper, unless you have a magnifying glass at hand). Still better is Jeon's live version at http://www.w3c.or.kr/~hollobit/roadmap/ws-specs/. This diagram is dynamic (you need Java to see it properly); clicking on any entry in the table of contents, top left, brings up a Web page giving detailed information about the specification in question.

Arguably, the history of Web Services standards is little more than the history of IBM's collaboration with Microsoft. In April 2001, W3C held a Web Services Workshop in San Jose, California, to study the long-term implications of SOAP, WSDL and UDDI. IBM and Microsoft consolidated their position as joint leaders of the Web Services movement by presenting a paper entitled 'Web Services Framework' (WSF). This proposed a consistent architectural framework into which individual specifications could be slotted.

"While most descriptions of Web-based solutions emphasize their distributed characteristics," the WSF paper remarks, "their decentralized nature — they have distinct management and control environments and communicate across trust domains — has much more impact on architecture of this framework and the requirements of the underlying protocols. So we focus our framework first on supporting application-to-application integration between enterprises having disjoint management, infrastructure, and trust domains."

Next, the authors ask: "Why have a framework?" They answer this question by observing that a common framework "divides the problem space into sub-problems with specified relationships". Thus, different teams will be able to work on various parts of the overall problem without continuously co-ordinating their activities with all other such teams. Instead, they can aim to fulfill the requirements for their own piece of the framework, secure in the knowledge that if they do so, their own work will dovetail neatly with everyone else's.

Moreover, the intention was always that developers — who write and invoke Web Services — can add (or remove) additional features such as security, transactions, choreography and quality of service (QoS) independently of each other. That is, the code required to add security does not interfere in any way with that needed for QoS, and so on.

This principle of orthogonality permeates the entire Web Services architecture. Ever since that Web Services

C:\Documents and Settings\Administrator\Local Settings\Temporary Internet Files\OLKD40\WS_Advanced.mmp - 2004-04-02 - 전종홍

Figure 3.1: Documenting Web Service specifications

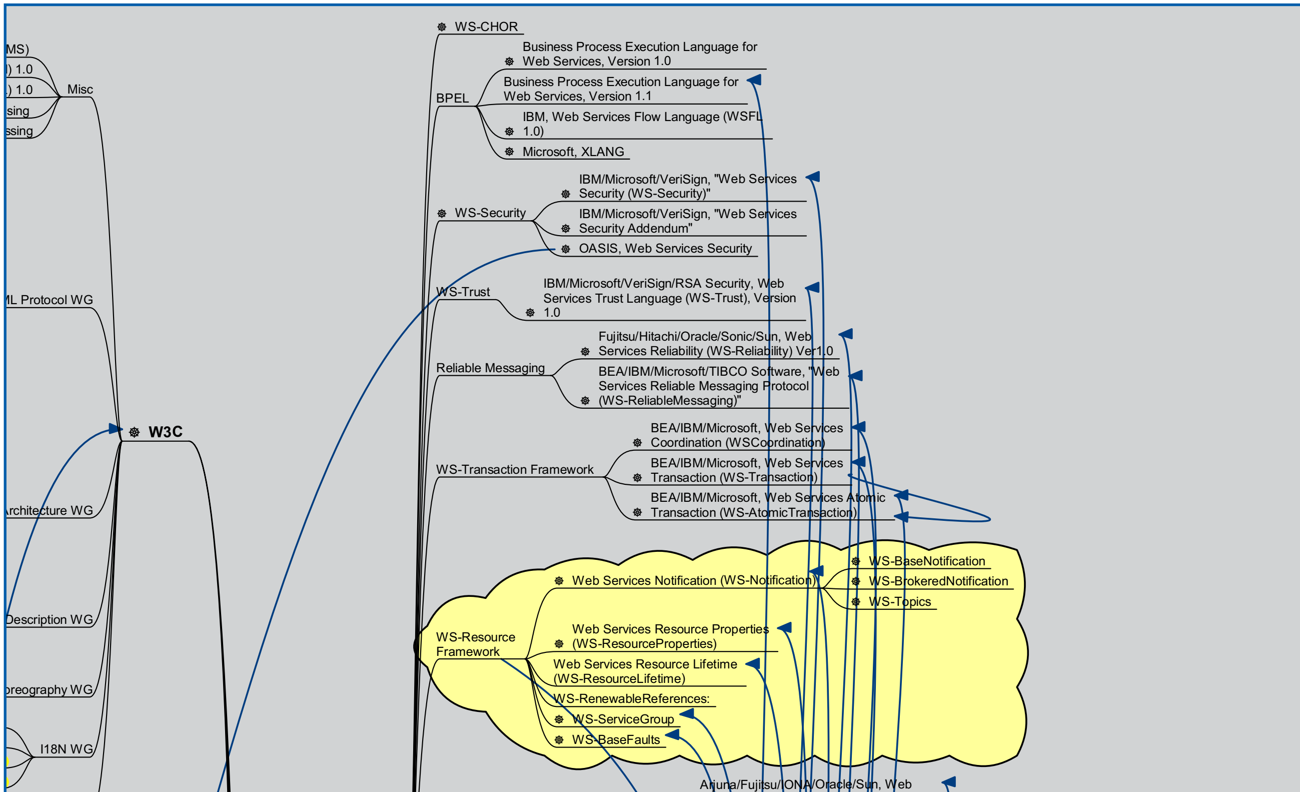
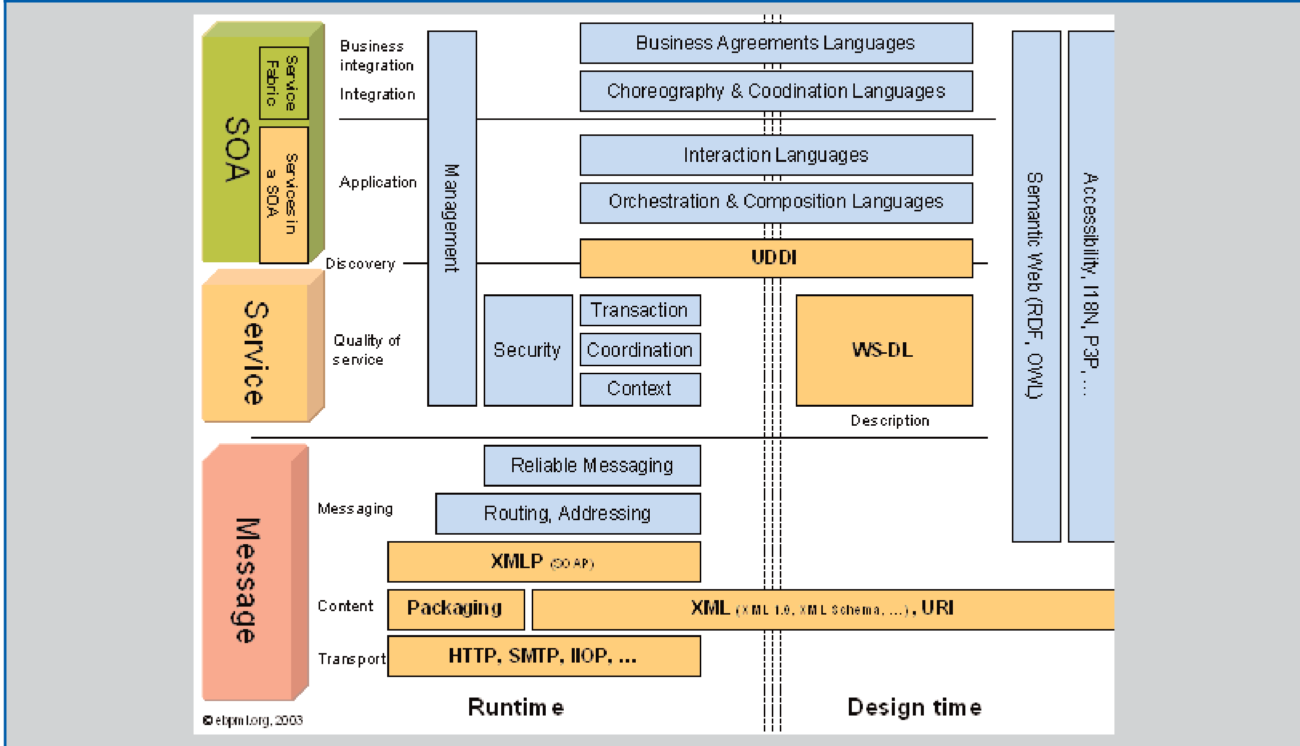


Figure 3.2: A close up of part of Figure 3.1

Figure 3.3: Web Services (Source: WebServices.Org)



Workshop, IBM and Microsoft have concentrated on delivering the vision outlined in the WSF paper. They have been assisted in these efforts by a growing set of allies including:

- **BEA**
- **Bowstreet**
- **Computer Associates**
- **Fujitsu**
- **HP**
- **Oracle**
- **RSA**
- **SAP**
- **Siebel**
- **Sun**
- **Tibco**
- **VeriSign.**

Recently, companies such as Arjuna, BEA, Fujitsu, Iona, Oracle and Sun have also taken to publishing selected specifications of their own, in what appears to be a form of competition with those of IBM and Microsoft.

Web Service specifications

A full explanation of all the Web Services specifications in circulation, and the related activities, would fill a book. Nevertheless, Figure 3.3 gives a vivid impression of the subject area and its complexity. Drawn by Jean-Jacques Dubray (Technical Architect, Attachmate, and editor of the OASIS ebXML Business Process Specification), it proposes a taxonomy for the Web Services stack.

Specifications are classified along two axes:

- **one axis deals with the specifications' applicability: run-time, design-time or both**
- **the other axis features what Dubray considers "the three fundamental concepts of modern computing" — messages, services and SOA.**

The most important single fact (and attraction) about the Web Services architecture is that it was conceived, and designed to be, as modular as possible. In accordance with the principle of orthogonality, Web Services can be used with different underlying protocols, such as:

- **plain TCP/IP**
- **HTTP**
- **SOAP over HTTP**
- **SOAP over JMS**
- **even (in theory) SOAP over IIOP.**

A Web Service can be 'pure vanilla'. Or it can be enhanced with security, transactions, choreography, QoS and other optional features.

The advantage of this arrangement is that the same communications model can be used across a wide range of different scenarios:

- **from infrequent, informal communications**
- **to robust, fault-tolerant, highly scaleable corporate business interfaces.**

We already know that Web Services work well at the low-power, informal end of the spectrum. There are plenty of examples to choose from. Every vendor's Web site lists a few. But perhaps the most striking case studies come from Amazon and Google, both of which offer Web Service programming kits free of charge to all comers.

Yet, quite how satisfactorily Web Services will perform at the high-powered end of the spectrum remains to be ascertained. This question brings us to the perplexing topic of SOA.

What is an SOA and what is the relationship to Web Services?

After several years, when Web Services were at the top of the public relations/analyst totem pole, they now look like being replaced by SOAs. The SOA market will be worth $4.5 billion by 2005 and $43 billion by 2010, according to ZapThink, an analyst firm that specializes in Web Services and SOA. Gartner Group's Yefim Natis and Roy Schulte, in a document posted on BEA's Web site, predict that "by 2006, more than 60 percent of enterprises will consider SOA a guiding principle in designing their new mission-critical business applications and business processes (0.7 probability)". Natis and Schulte also foresee that, by 2006, "more than 75 percent of midsize and large enterprises will have deployed SOA-enabled development tools and middleware".

But what is the difference — if any — between SOAs and Web Services? An SOA is said to be more abstract. For instance, most vendors admit that you do not need Web Services in order to build an SOA. Bob Sutor, IBM's director of WebSphere infrastructure software, says that "...SOA is not a new thing. By most accounts we're actually in the third decade of thinking about SOA".

It is an interesting (though frustrating) exercise to seek a concise, exact definition of SOA. The roundabout description provided by Microsoft's Don Box is typical: "Service-orientation is an important complement to

object-orientation that applies the lessons learned from component software, message-oriented middleware and distributed object computing. Service-orientation differs from object-orientation primarily in how it defines the term 'application'. Object-oriented development focuses on applications that are built from interdependent class libraries. Service-oriented development focuses on systems that are built from a set of autonomous services".

Although IBM employees talk freely about SOA and its merits, they rarely venture an explicit definition. Among the best is the following, from an IBM training course, 'Connecting the Enterprise with WebSphere Studio':

"An informal definition of SOA is simple: an application that models business activities as services and uses an infrastructure based on WSDL has a service-oriented architecture".

In another IBM document — "Migrating to a service-oriented architecture, Part 1" (dated December 2003) — Kishore Channabasavaiah, Kerrie Holley and Edward M Tuggle, Jr. start by saying what SOA is not. "First, though, it must be understood that Web Services does not equal service-oriented architecture." This is because Web Services are "a collection of technologies", not an architecture — although "they are, at the very least, a proof of concept that SOAs can finally be implemented".

"So what actually does constitute a service-oriented architecture?" our researchers go on to ask. "SOA is just that, an architecture. It is more than any particular set of technologies, such as Web Services; it transcends them, and, in a perfect world, is totally independent of them. Within a business environment, a pure architectural definition of a SOA might be something like 'an application architecture within which all functions are defined as independent services with well-defined invokable interfaces which can be called in defined sequences to form business processes'."

SOA is often described as a business architecture rather than a technical architecture. In theory, this means that each service should be aligned or identified with a specific business process or task.

In the words of Scott Cosby, IBM's director of WebSphere Business Integration product management, "You have to be able to break down business processes such that they are aligned with what the business wants to do. SOAs allow you to build off of these business processes and allow you to do so more quickly and easily".

If an SOA is a technical architecture, it would (surely) tell one which languages, protocols and modes of communication must be supported. It would define exactly what is, and what is not, within its remit.

But this is just what no one seems ever willing — or able — to tell us about an SOA:

- **is it synchronous or asynchronous?**
- **is it RPC-based or message-based?**
- **is it proprietary or vendor-neutral?**
- **does is use a bus or hub and spoke or any to any approach?**
- **does it have language bindings or not?**
- **must it use XML?**
- **must it use SOAP?**
- **must it use WSDL?**

The answer to all of these questions, so far, appears to be: 'Perhaps. We do not really know.'

If an SOA is not a technical architecture ...

So, if the 'architecture' in SOA is not a technical architecture, what kind is it? Presumably it must be some kind of business architecture. But where is that defined?

At the level of technical architecture, we had plenty of questions (see above) — just no solid answers. But as far as the business architecture is concerned, there are hardly even any questions.

There are serious contradictions, though. We are told that an SOA:

- **must be based on an enterprise-wide model and plan; but also that it should be undertaken a little a time, starting small**
- **will enable large-scale software re-use, as new projects can rely on existing services to supply part of their functionality; but there is no explanation of how responsibility and funding will be allocated, or how fixes and enhancements will be co-ordinated.**

Appearance and reality

But surely, you may be tempted to say, there must be something to this SOA business? After all, everyone speaks well of it; and the other day I read of a very impressive success story. This is a rather naive attitude which overlooks the fact that, with enough resources, success stories are almost always possible to find.

One of the biggest sources of systematic error when assessing the value of new technology is the '200% RoI' syndrome. Enthusiastic analysts, journalists and marketing people quote some CIO as saying 'This investment earned us a 200% RoI' (although they usually fail to mention how long it took). That does not necessarily prove anything except (perhaps) that their old way of delivering was less than optimal.

Many organizations have criteria for even evaluating new projects, one of which is to set a fairly high level of RoI as a target mechanism before capital expenditures can be made. To say that 'Our SOA project earned a 200% ROI' may sometimes mean no more than 'It met (one of) the corporate criteria for being allowed to go ahead'. One might even add 'We improved one of the most neglected and inefficient areas of our enterprise, with the aid of some cash and the latest IT techniques".

One possible reason for the new emphasis on SOA is that the profit margins to be earned from Web Services are under serious pressure. People are realizing that there is not all that much to Web Services, and they not necessarily that difficult to build. Unfortunately, it is difficult to make much money selling simple, well understood technology. Vendors had to come up with something more ambitious and complicated. A cynic could well argue, as a result, that is why we have SOA and Business Process Management (BPM).

John Crupi, the Sun MicroSystems distinguished engineer, explained at JavaOne how Sun sees an SOA. "The idea behind it is we go in to a customer and help them understand that SOA is not just an architectural style but real stuff, so they have to think about the implications for their current infrastructure, and we score them in terms of readiness. We're expressing the reality that you don't just press a button and say you're SOA ready."

Other vendors' spokespeople have expressed similar sentiments. 'An SOA does not come in a box.' 'An SOA is not a single product or technology.' And so forth.

In his paper, 'Web Services: Pathway to a Service-Oriented Architecture?' Gregor Hohpe (of the consultancy ThoughtWorks) writes that "Service-Oriented Architectures model the enterprise as a collection of services that are available across the enterprise". This is a key insight, which highlights the need for SOA to be based on an enterprise model. But modeling a whole enterprise is not something you undertake lightly. So adopting an SOA is likely to be a lengthy, expensive and potentially 'bet-the-business' exercise (do not forget those enterprise data models — so beloved of first CASE and then all those Information Warehouses).

Steve Vinoski, chief engineer of product innovation at Iona Technologies, nails the problem with his description of the way in which SOA is being sold:

"Poor abstraction skills can be particularly troubling in the context of service-oriented architectures. Developing an SOA requires that you first identify service abstractions, but this reverses the typical approach of writing applications first and then writing the specific services that application needs. With an SOA, the goal is to put appropriately abstracted services in place for reuse by numerous applications, rather than tightly coupling the services to a single application".

In stark contrast

In stark contrast to the dictates of SOA, some of the most successful Web Service projects have been those with the least ambitious goals — for instance, amplifying the services already provided by the Web. The sheaves of Web Services offered free by Amazon and Google are good examples: they 'merely' allow users to feed the results of Web site queries into programs of their own devising.

Many applications with similarly limited functionality have been set up within corporations — often with a good deal of success. But such Web Service applications are almost always the antithesis of an SOA. They are ad-hoc, tactical, and independent of any over-arching enterprise strategy.

Management conclusion

As Mr. Welsh argues, the good news is that, as usual, the state of the art is moving onwards and outwards. This is just as true in software engineering and, in particular, distributed systems engineering, as anywhere else.

Web Services add a new and extremely useful tool to the developer's box of tricks, and their potential power is growing with every new specification that becomes a standard. There is already plenty of evidence that Web Services are practical and cost-effective in the role for which they were originally designed — whipping up ad-hoc connections between random applications across a network. But how much more can be asked of them?

IBM, Microsoft and allies are heavily committed to a long-term program of enhancing Web Services with security, transactions, QoS and many other features that are vital for production systems. Ever more sophisticated capabilities —

such as choreography and orchestration — are being added to the basic Web Services stack, theoretically providing everything needed by BPM systems. While nothing can be taken for granted until it has been proven in action, there seems no reason why this should not successfully deliver.

SOAs, in contrast, are a more of a puzzle. On the one hand, we are told that SOA-like disciplines have been practised for decades. On the other, that it has only become truly desirable with the advent of Web Services. An SOA is best adopted cautiously, one little piece at a time; yet it must be based on a comprehensive model of the entire enterprise.

Leading vendors are all eager to talk about their leading-edge SOA toolsets; but none of these are quite ready yet. However, they are all ready to enter into large-scale consulting engagements to scope out the possibilities. No enterprise can afford to ignore SOA. But it would be wisest to ask for detailed explanations from any eager vendor, and to double-check the promised RoI of any SOA project.

Do we need ESBs?

Mike Beeston
Principal
Maven Associates

Management introduction

Most organizations have recognized the benefits of adding application integration capabilities to their infrastructure services. Many have implemented (or at the very least are planning for) commercial technologies to help deliver integration services.

The core middleware technologies for integration continue to mature, providing more extensive and robust capabilities. Today the focus of many vendors is on the 'Enterprise Service Bus' (ESB).

But, as Mike Beeston explores from his own consulting experience:

- ***is an ESB something new?***
- ***is it anything more than a concept or a design/deployment pattern?***
- ***is it even what organizations have been seeking when they embark on integration projects?***
- ***in what way might it be something that is needed?***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

ESBs: something new, something old

There are grounds to argue that organizations have been seeking to deploy an 'ESB' for some time. A number of years ago, when IBM's MQSeries (now WebSphere MQ) was still a relatively new product, I was part of a consulting team that worked with blue chip clients to develop solutions exploiting messaging middleware. Many clients understood its potential. They were keen to embrace new approaches to application design — after earlier attempts to create distributed systems using tightly coupled synchronous technologies had mostly failed (or, at best, had 'sort of worked').

This was before solutions — that might be broadly categorized under the Enterprise Application Integration (EAI) umbrella — arrived. It also long pre-dated the arrival of today's J2EE and .NET application frameworks (Figure 4.1).

Even then, clients were rightly cautious that, as early adopters, they should guard against designs which might tie them to specific technologies that could all too quickly change (or disappear). But, at the same time, they also recognized that there were opportunities to gain real competitive advantage via integration, ones which business competitors might be slow to follow.

One such project in the mid to late 1990s demonstrates how the aspirations of a client organization could be facilitated through a progressive design of a messaging solution. Driven by increasing market competition, plus the needs of an imminent acquisition, the team worked with the client to develop services that extended a base messaging product to enable capabilities that today would be quickly categorized as an ESB — albeit developed as a bespoke rather then an open solution.

Two key features of the design justify it for consideration as a youthful ESB:

- **one was the goal to ensure that the complexities of the routing and transport capabilities were isolated from the application code that accessed them**
- **the second was the provision of administration tools that, utilizing the underlying messaging system, were able to manage the distributed application configuration.**

Together these provided the client organization with the benefit of avoiding technology lock-in. At the same time this 'youthful ESB' solution simplified application development as well as provided flexible control of the distributed environment.

Designs like these were by no means unique at the time. Indeed, in my experience, it was quite common to include some degree of isolation code between the application and middleware. The major distinction to make between contemporary solution design and those of the mid 1990s is that, today, open standards and commercial products are the building blocks — rather then custom code and bespoke development of projects like those on which I worked.

The concepts behind an ESB are, therefore, not entirely new. The key difference today is the maturity of the underpinning technologies and standards which enable the potential value of an ESB to be achieved more easily by a far greater number of organizations. In comparison to the early days of mainstream messaging, these benefits now come with significantly lower technology risk as well as lower acquisition costs.

What is behind an ESB?

A common problem for many considering an ESB for the first time is: 'what is it?'. This difficulty is compounded by the range of answers, which seem to depend on whom you ask.

For example, there several vendors have sought to popularize 'the ESB concept' and provide products that are possibly named, or certainly described, as ESBs:

- **Sonic Software has a product family including Sonic ESB**
- **Cape Clear offers a suite of products to address the ESB problem.**

At the same time, there are other vendors which prefer to maintain that 'an ESB' is a design pattern that can be implemented through use of their products. Particularly notable in this second group is IBM which has provided commercial messaging technology for over 10 years and enjoyed widespread market acceptance.

Yet, when drawing up a list of ESB functionality, there are few surprises when one considers the many contemporary middleware offerings that support application integration. ESBs all seem to include most, if not all, of:

- **communications capabilities, with support for various interaction models (from fire and forget to publish/subscribe and so on), most typically based on asynchronous messaging**
- **sophisticated routing**
- **transactional integrity**

- **broad connectivity — from application frameworks (.NET, J2EE) to application suites (ERP, CRM, etc.) to raw technologies (databases, transaction managers, etc.) to language bindings**
- **support for broadly accepted standards, particularly Web Services and XML, as well as for industry-specific standards for data exchange and business process models**
- **rule driven processing**
- **message mediation (transformation, enrichment, etc.)**
- **security (authentication, authorization and so on)**
- **development tooling (.NET, Eclipse and others)**
- **operations and systems management.**

As I have already discussed, many large organizations had aspired to integration solutions with ESB-like qualities for some time. In turn this helped to develop the maturity of middleware technologies and produced capabilities of a high quality. Middleware vendors had to respond to these expectations by packaging different middleware technologies together as coherent product offerings — in increasingly useful combinations.

In other words, an ESB can be considered as the synthesis of many capabilities, particularly many of those listed above, that are implemented to a well-defined design in support of architected objectives. Where ESB function can be satisfactorily provided by a single product deployment then an ESB might also be considered a product.

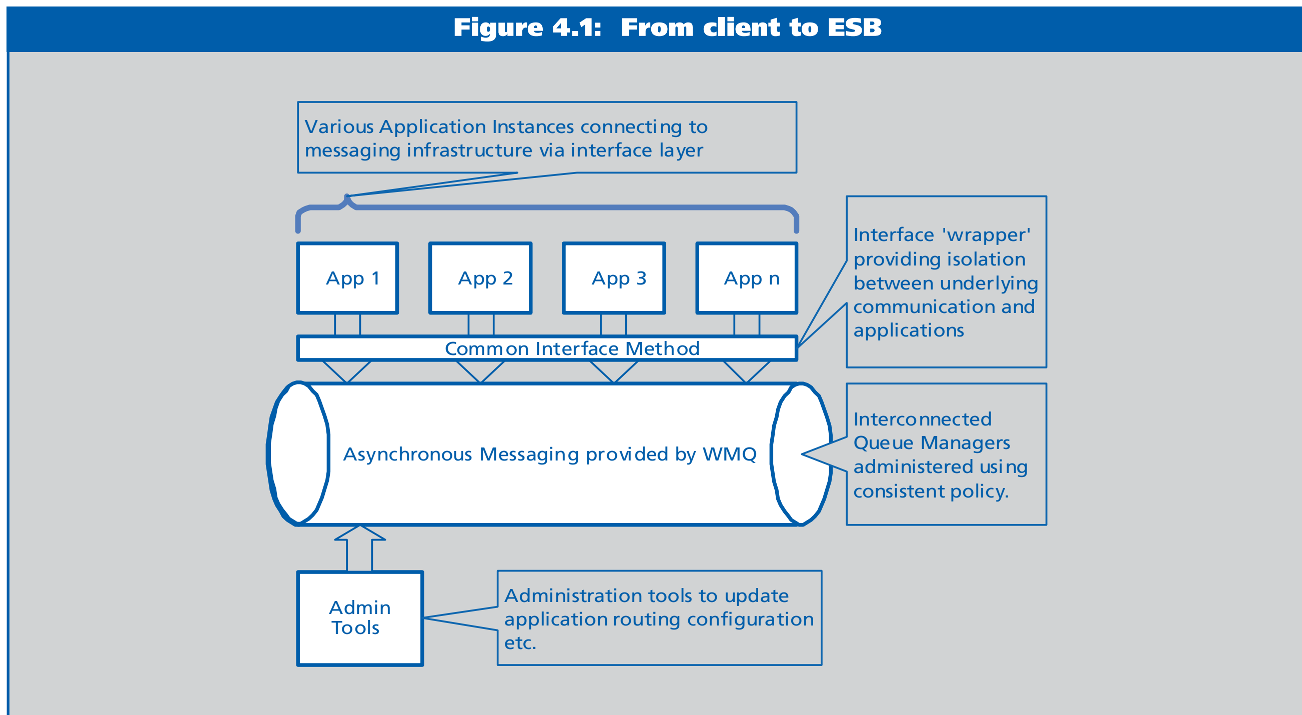
ESB in an EAI context

The qualities of an ESB appear to hold much in common with EAI technologies. As I have already described, it was several years ago that adventurous organizations were taking the initial ground breaking steps to exploit message queuing middleware using 'design styles' that shared a number of common elements with today's ESB design patterns. Application integration technologies certainly exhibit a number of similar functional capabilities.

For these reasons, EAI deployments — especially when considered in the light of the current understanding of ESBs — show the origins of today's ESB design patterns. However an ESB is about more than purely an application integration capability. While an ESB shares much in common with EAI solutions, and a common lineage, an ESB ranges further than EAI. An ESB can be distinguished from traditional message oriented middleware and EAI technologies in a number of ways.

ESBs promote the use of open standards wherever possible. Think of Web Services and their core first generation

Figure 4.1: From client to ESB



standards of SOAP, WSDL and UDDI along with along with second generation standards such as Business Process Execution Language for Web Services (BPEL4WS) and security specifications. While established middleware technologies have, to various extents, successfully added support for these standards, they are not necessarily available as the preferred usage methods.

ESBs provide a foundation for 'service oriented' and 'event driven' architectures. Traditional middleware provides capabilities to support such efforts, but not without each individual organization making its own particular effort to identify how to use each technology in the relevant way. The ESB design pattern naturally supports architectural initiatives. As such it can drive improvement in the provision of IT solutions.

ESBs incorporate tooling to address distributed administration and management. Traditional middleware tends to be implemented to satisfy a 'management by node' approach where specific instances of products(s) are controlled independently of each other. Co-ordination is achieved not by the middleware itself but by the design and administration procedures. The deployment topology of hub and spoke, for example, fits this description, not least where hubs are constructed from (say) IBM's WMQ family of products. Hubs may be interconnected, but tend not to share configuration data unless explicitly distributed by the administrator.

Conversely an ESB contains many nodes that are implicitly connected by virtue of an ESB implementation, and this can share common configuration information. Thus an ESB provides a policy based mechanism to enable improved communications and service provision in a dynamic business environment.

This can be put another way. When contrasting an ESB solution with a traditional EAI implementation (consisting of messaging middleware and one or more brokers), we see that an ESB:

- **promotes the adoption and use of open standards**
- **facilitates new application designs, specifically those directly supporting an SOA**
- **provides inherent administration and management tooling which address distributed management from the start of an ESB's introduction.**

At the same time, ESBs provide the capabilities of a traditional EAI solution, for example:

- **non-invasive mediation between disparate applications — message handling and routing facilities enable in-flight processing of service requests and responses; applications can be developed free of integration code**
- **connectivity across multiple environments**
- **multiple qualities of service — scalability, security, availability and so on.**

An ESB represents, therefore, much of what has been aspired to by many forward looking EAI projects, in that such projects recognized that they had to facilitate change in the provision of IT services. In this sense EAI function should be considered a sub-set of an ESB. In effect EAI was a necessary step in the evolution towards ESB-based solutions. As organizations have developed comprehensive architecture programmes — and become more acutely focused on achieving decent return on investment — it is appropriate that the role of the ESB has become more clearly defined.

ESB — enabling both EAI and SOA

The ESB design pattern represents the natural evolution of EAI technologies converging with Service Oriented Architecture. Integration technologies are already widely accepted as an integral part of many organizations' IT capability, so the groundwork is laid for adoption of the ESB. In turn this indicates that, as organizations continue to develop their EAI capabilities to match the ESB design pattern, they will be significantly better equipped to support new architecture initiatives which can exploit a service orientation.

We certainly need the ESB. But its success in any organization will depend on the recognition that an ESB will only deliver its full potential when it is included in an overall IT architecture and infrastructure that is aligned to business strategy.

Such an architecture will address key business requirements for change, particularly the establishment of holistic management of a process-oriented business. As processes are optimized across departmental and organizational boundaries, the underlying IT architecture will increasingly depend on ESBs to link service-oriented applications together.

The industry movement to deliver the capabilities of the ESB should, therefore, provide a catalyst for business change. With the ESB design pattern implemented in some form by commercial technologies, organizations can focus on innovation to create new application solutions which

will meet the ever more important business demands of agility and process efficiencies.

That said, developing a strategy for adoption of an ESB requires consideration of multiple factors. While an ESB can provide EAI-like capabilities, its real value will be derived from the ability to facilitate new service oriented application designs. It is to be expected that these designs will also be influenced significantly by the application frameworks (.NET, J2EE) in which they are developed and deployed. From this one can reasonably conclude that, over time, the dominant providers of ESB technology are likely to be aligned to these frameworks in some way — particularly IBM for J2EE and Microsoft for .NET.

As the momentum for provision of commercial ESB solutions progresses, product offerings will continue to evolve. Well established middleware providers with existing products are pursuing development strategies that will result in new products that are built upon and assimilate open standards. For example, consider IBM. Today an ESB pattern can be realized through the deployment of multiple family members of the WebSphere Business Integration suite (particularly WBI MQ, WBI Message Broker along with WebSphere Application Server Web Services Gateway) combined with a strategy that incorporates use of open standards (XML, J2EE based messaging).

In the future, availability of new product offerings — presumably in the shape of some form of specific 'services integration bus' will deliver similar functionality but implemented on standards-based technologies. Yet, deployments already built on the earlier solutions will still provide a large portion of the candidate ESB functionality presented earlier and work with any dedicated ESB offering that appears.

Meanwhile there are a number of vendors providing ESB solutions. A representative list includes IBM, Sonic and Cape Clear (as mentioned above), along with vendors such as:

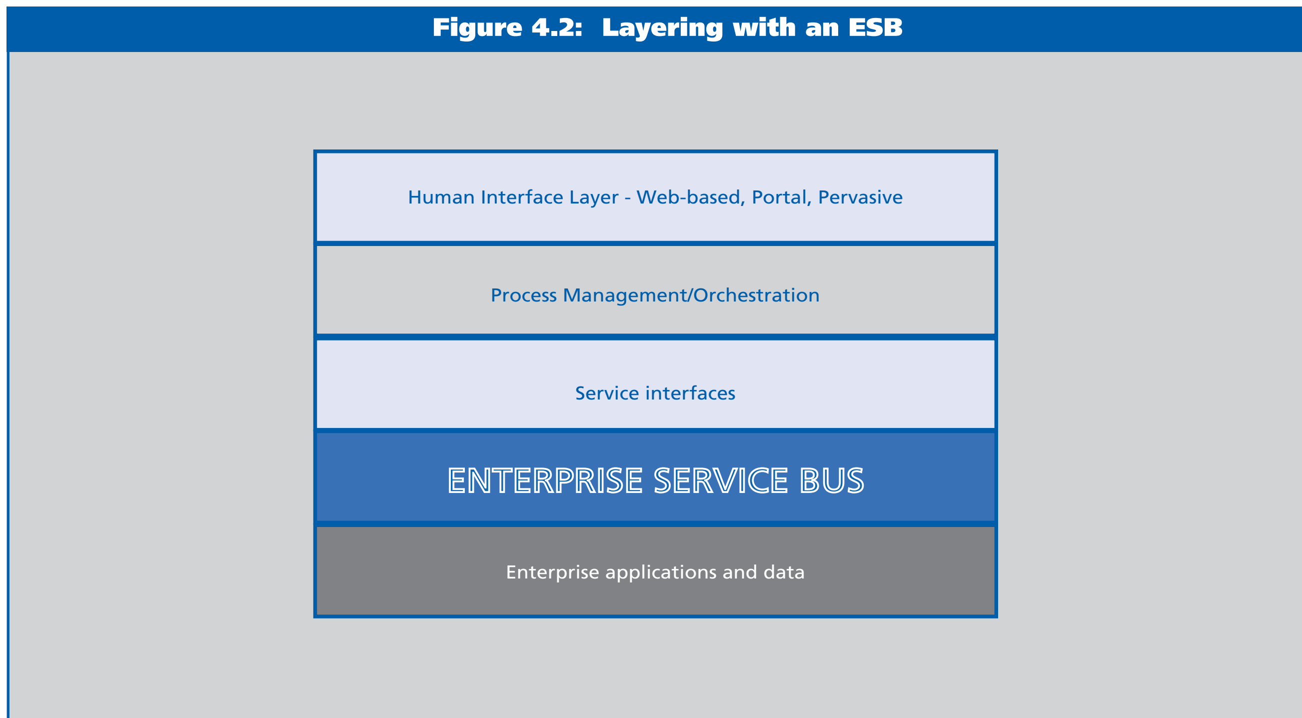
- **PolarLake (JIntegrator)**
- **Fiorano (Fiorano ESB)**
- **WebMethods (Fabric)**
- **in addition to many other vendors, large and small, with products or plans to address the ESB.**

Many of these have grown from a Web Services only perspective. These ESB products can be expected to develop and extend their capabilities beyond the existing reach to support more complex communications models and richer functionality in general.

ESB concerns

My concern about ESBs is not if they are one product, or

Figure 4.2: Layering with an ESB



even a collection of products. Much like any middleware, the value is found not so much in how they are packaged, but instead in how they can be used. For this reason it is better to approach an ESB much as one might other middleware technologies. This view supports the notion of the ESB as more of a design pattern, which is deployed in part by one or more products.

In contrast, integration middleware has evolved to provide rich functionality that operates at the centre of an ESB implementation and this is extended with improved distributed management capabilities. The concept of an ESB should be integral element of broader architectural initiatives, particularly those aimed at delivering a coherent 'Service Orientation'.

In my view, an ESB should provide a foundation on which to develop a 'Service Oriented Architecture' (SOA) that can inject new agility and quality into provision of composite applications. Typically delivery of these applications would be accessed using portal technologies.

Thus the value of an ESB as a design pattern is only realized when it is part of an IT architecture which supports improved alignment between business strategy and IT provision. But this is a rare bird, indeed.

Management conclusion

The growing acceptance of the ESB design model is indicative of an improved understanding of how infrastructure services can alter the way applications are designed, developed, deployed and utilized. The advancement of the J2EE and .NET application frameworks, plus the widespread recognition of the opportunity provided by Web Services, has resulted in a new style of application. With a service oriented application portfolio, distributed applications can be constructed from granular, re-useable services. With the use of standards based interfaces — published in both public and private directories — applications can be more quickly assembled, or re-assembled, to meet ever changing requirements. All these capabilities are essential as organizations seek to establish business process efficiencies.

In this analysis, Mr Beeston has argued that an ESB is really a design pattern and that this pattern can, to various extents, be implemented through use of commercial product offerings and selection of open standards. Furthermore, many middleware vendors have accepted the ESB as the baseline for future development. The result is that open standards and increased functionality will increasingly filter into products, old and new, that address the Enterprise Service Bus market.

Today's ESBs are much more than the early initiatives to exploit MQM and deploy EAI solutions. But without those initiatives and the lessons learnt from them, the ESB design pattern would neither be as technically mature or widely appreciated.

As such, an ESB has a fundamental role to play in support of 'Service Orientation' and the associated application development disciplines. Just as significant, it assimilates the capabilities previously handled by application integration solutions. As a result, an ESB both facilitates new application solutions as well as integrating legacy applications.

Understanding infrastructure

Peter Bye
Consultant
Unisys Systems & Technology

Management introduction

The IT industry has a depressingly poor record when it comes to the reliable delivery of properly working systems. If other industries that choose to call themselves branches of engineering consistently performed in the same way, the result would be widespread outrage (of course, sometimes other engineering projects do perform poorly as well as make headlines).

We can all think of examples. The public sector seems to lead the way in project failure, judging by the publicity its failures receive, but that may be partly because the private sector is better at hiding its disasters and partly the fact that taxpayers' money is used in the public sector which means that it is more open to scrutiny and hence exposure of failures. After all, it is all too easy to recall the number of private sector failures that have failed to make any headlines but would have been front-page news if they had occurred in the public sector.

There are many reasons for project failure. In this analysis, Peter Bye explores one cause — or more precisely one set of causes — of difficulty, which may be summed up as problems associated with not understanding infrastructure. He starts by taking a brief look at some aspects of project failure, then shows why this matters before processing to discuss — with illustrative examples — how a disciplined architecture with an understanding of the infrastructure is integral to avoiding failure.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

Complexity and failure to understand

The environments common in many current IT projects are complex, involving middleware architectures (such as J2EE and .NET), older application architectures, extensive networking and more. If the technical ramifications of these environments are not sufficiently understood, the resulting implementations are likely to fail, mainly in meeting non-functional requirements — but also in some cases failing functionally. An inability to understand the infrastructure is a frequent cause of failures to meet functional and non-functional requirements (where non-functional requirements include resilience, performance and scalability, systems management and security).

Often, success depends on the correct use of an infrastructure. Failing to take these requirements into account at an early stage, or not understanding their importance and how the infrastructure affects them, easily leads to systems that do not perform as expected. Furthermore, these failures are usually hard to fix in the advanced stages of implementation.

More about IT project failure

A successful IT project is usually defined as one which:

- **satisfies all its requirements**
- **is implemented on or ahead of schedule**
- **is delivered within the costs originally planned.**

I would argue that one additional criterion should be added: the resulting system should deliver a positive benefit to its sponsors. It is, unfortunately, possible to meet the first three criteria but still have a system that does not provide any benefit, because the requirements as stated were not right for the particular business objective. This would be a perfect implementation of the wrong objective, or at least an incomplete one.

In similar fashion, a failed project may be defined as one that does not satisfy all the four criteria above. Failure may be partial, for example a time and cost over-run or complete, where the project is eventually terminated without any system going live.

Because of the costs involved — some estimates show that IT failure costs around $400-500 billion per annum — there has been, and continues to be, a high-level of interest in measuring the scope of failure and the reasons for it. Some studies have shown that only 15-20% of projects are a success and as many as 30% fail completely. Whatever the figures, it is undeniable that IT project failure is a serious business as well as technical problem.

The reasons for failure are various. One has already been mentioned: failure to specify requirements adequately, leading to incompleteness or frequent change. This appears to be the largest single cause of failure, but it by no means overwhelms the other causes.

Lack of user involvement is, according to some sources, the second most common cause. This, of course, could result in the first problem. If the expected users of the system are not involved, the requirements are likely to be incomplete or plain wrong; the system will not do what they want. Tony Morgan in his book *Business Rules and Information Systems* (Addison-Wesley, 2002) addresses in depth the question of defining requirements and business rules, and turning them into IT systems.

Lack of technological understanding is also on the list of reasons for project failure — and it is this cause of failure that is the concern of this analysis. I have called the analysis 'Understanding infrastructure' because it is how the infrastructure works that is frequently not understood.

But what exactly am I including in the word infrastructure? My next section aims to throw more light on the subject.

What is meant by infrastructure?

Applications and databases of various kinds provide the business value of IT. Infrastructure comprises the elements in which these applications run. It includes:

- **the hardware — servers, disk subsystems, etc.**
- **the networks, both local and wide area**
- **the system software — operating systems, database managers and so on**
- **the middleware, for connecting systems (and applications)**
- **supporting tools, like management systems.**

In this analysis, I concentrate particularly on middleware. But there are interdependencies, for example between clustering servers and how middleware products use it.

In this light, it is fair to say that today's typical enterprise IT environment will have grown over a long period and be complicated as well as heterogeneous.

There will be many servers — hundreds, perhaps even thousands — and different types of server, ranging from small systems running niche applications to large mainframes, and including both local and shared storage subsystems (for example, storage area networks and cartridge tape libraries). There will also be fixed and mobile laptop

PCs, with different kinds of client, as well as possibly some old terminals or terminal emulators.

In addition, there will be a variety of networks, both local and wide area which will include intranets, extranets and the Internet. Although most organizations have now standardized on TCP/IP, there remain pockets of older protocols, most obviously SNA. In some cases, the older protocols are used to communicate with third party organizations and may be difficult to change, as many people have to agree and then approve any new protocols. An obvious example is a shared automatic teller machine (ATM) network, or a network connecting point of sale devices in retail outlets to a credit card system.

Then there are the different operating systems:

- **from the mainframe ones (z/OS, OS2200, MCP, VSE, VM, etc.)**
- **to the many variety of flavors of UNIX (albeit increasingly Linux)**
- **to Windows**
- **to embedded operating systems.**

More than one middleware infrastructure is possible, just as there will be systems with no recognizable middleware. Middleware examples include CICS, Tuxedo, Open Group DTP, J2EE application servers and .NET. These are likely to be running applications as well as providing a framework for integration. Message orientated middleware (MOM) is also likely to be present, for example MQSeries and, increasingly, SOAP and other Web Services middleware. There may also be various portals and integration tools (EAI products) in use.

A broad variety of management tools will be in place. These will be integrated, in varying degrees, to provide a picture (but not necessarily an overall picture).

This is the starting point for most new IT projects. There are few green field sites. While some applications may be self-contained (in that they only have a set of end users and do not interact with other systems), many, if not most, new applications are composites, in that they contain new logic and integrate existing systems with which they have to co-operate. To add to the complexity, the existing systems may be both internal, as in owned by the same organization, or external (owned or operated by a third party).

An example

A simple example illustrates this. Suppose a bank — let us refer to it as the AZ Bank — offers a variety of current and savings accounts plus mortgages. It owns its own systems. Suppose further that it offers insurance products, but outsources the product development and management to an insurance company (we will call it HiRisk). Now suppose research by the AZ Bank into its customers' desires indicates that it should provide an Internet-based service, allowing its customers to obtain the current status of all products held, and a set of services for transferring funds between products.

A project to implement this set of requirements would require a customer relationship management (CRM) system to relate the customer to products held. An enquiry from the customer would then result in a query on each system holding product information, to obtain the current status. This will includes HiRisk's system — if the customer holds insurance products. Figure 5.1 shows a schematic of the environment.

As can be seen, the interaction between the various systems uses middleware. System architects would need to take the environments of the existing systems into account, as well as the new environment required by CRM system. They would also need to look at what HiRisk would support. A further complication could be whether or not the new project would be implemented on its own, or would be within the context of establishing a wider environment to support future, similar requirements. A number of organizations have set up an infrastructure to cope with projects requiring the integration of a number of systems. They have often used a specific application project to establish a general infrastructure, which allows similar projects to be completed more quickly in future.

The starting point is — or should be — the definition of an architecture, or high-level design. This provides a framework for discussing:

- **the business requirements of the system**
- **what is required to satisfy them**
- **any technology and product selection**
- **the non-functional as well as functional, to verify that both requirements are satisfied.**

In my experience, it is essential at this stage that the project architects:

- **either have an understanding of the technologies and products they are opting to use (whether existing or new or both)**
- **or hire people who do understand.**

Why does this matter so much? Can we not just assume

that the technology will do what we want, with minimal involvement apart from producing applications? The answer is no (explained below). Architects must work with those who understand the business as well as those who understand the technology. Or they need this knowledge themselves.

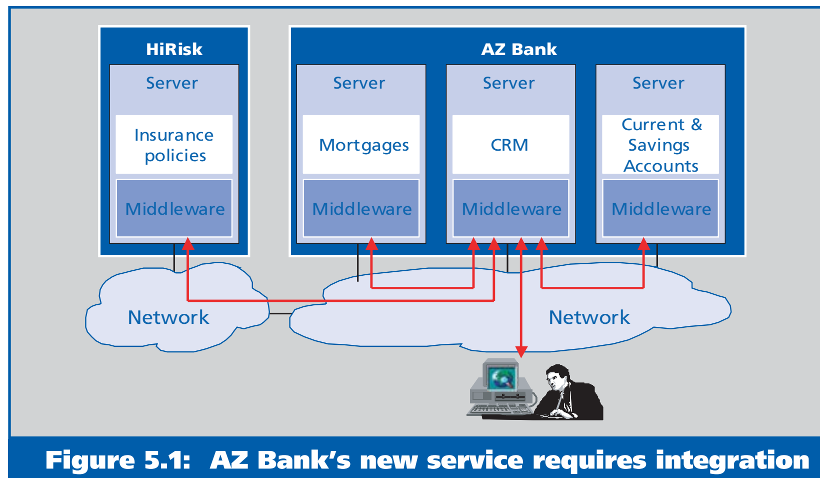


Figure 5.1: AZ Bank's new service requires integration

Architecture and technology

In working on architectures across the world, I often find that the person nominally in charge of the IT architecture is a technician through and through — and somewhat detached from the rest of the organization. Often that person, or group, is not really concerned with architecture, but rather is the organization's technology leader — the person (or group) who searches out new technologies and suggests to the organization what might be interesting for the future.

This role is important. But it is not architecture.

Architecture should not be described in a big document that is put on a shelf. An architecture should be the high-level design that is developed during a project's inception phase. It should then be used throughout the project. An architecture is valuable only when real problems are considered.

At the same time, beware trying to implement a design with knowledge only of the technology: this is fraught with danger. It is like making a decision about whether to go to work by train, car or bicycle without knowing how far you have to travel. If the technologists know little about the application, they may be forced to over-engineer, to build a Rolls Royce when a Mini will suffice, or the reverse, building a bicycle when a pick-up truck is needed.

Attempting to create an application without sufficient concern about the applicable (or available) technologies is also dangerous. Suppose someone programs an application and demonstrates it on a portable PC. It may work wonderfully, and impress potential users. But try to scale this to work with 1,000 users. All kinds of problems can arise.

Examples of what can go wrong

A couple of examples makes this clear. I recently attended a conference on architecture where one of the presentations was a case study of an application to process sales figures from a large number of retail outlets. A core part of the application was a data-collection process, followed by a 'daily' batch run to update the sales database. An architecture was defined and a fashionable runtime technology chosen.

The development progressed and a test was set up to execute a representative daily run against the database; it took more than the 24 hours available in a day. After a lot of work, this figure was reduced, but it was still not good. The matter was made worse by the fact that the technology selected for the implementation had minimal batch facilities. It was not bad technology; it just did not lend itself to batch processing. If sufficient attention had been paid to the technical aspects of the chosen environment at the architectural stage, many of the problems could have been avoided.

The second example concerns a relatively simple-minded transaction processing system. The application processed business transactions — each of which contained four short system transactions — with the complete business transaction (that is all four short transactions) finishing within about one minute. The application logic itself was relatively simple.

The architects designing the system decided to treat the business transaction as a business process and to use a product for business process management. This product was expensive and inappropriate for the job at hand.

The business transaction is indeed a process. But it is a very simple one, and hardly required a fancy process management tool which was more aimed at complex processes such as insurance claim management.

When the system was put together, it ran at about 1% of

the expected performance. Subsequent improvements were made because the tool was not being correctly used. But the real problem was the tool selected.

Both of the above problems could have been avoided if an appropriate technical understanding had been available early on. During the inception phase both technologists and application designers need to be involved — as are, for much of the time, representatives of the business function. Only by building up a picture of how applications and databases support the business processes can we derive or discern detailed performance and resiliency requirements. This in turn influences technology decisions, which also relate to distribution decisions.

For example, a particular configuration may need a high-speed network that is prohibitively expensive. Finding this out early on creates the opportunity to choose another architectural design and avoid spending time and money that cannot be recouped.

The starting point for the example (in Figure 5.1) is to define an architecture, and then ask a number of questions about it — including (in no particular order):

- **if the system is to perform updates (as opposed to enquiries only), what are the business requirements for maintaining database synchronization (for example if money is moved from one account to another), is there a need to make the changes immediately, in real time, or can they be deferred and done later, as long as they are made?**
- **what volumes of traffic are expected, what response times are required by the business, how many users are expected at any one time and what is the expected growth profile?**
- **what level of availability is required and does this apply to the time the application is online (for example 24x365) or just some part of the day, such as 06.00 to 23.00 and what availability is expected within the work time (99%, 99.9%, 99.99%, etc.)?**
- **is there a requirement to handle multiple access channels — for example browsers, fixed telephones, mobile telephones, ATMs and so on?**
- **what security level does the business expect — for example concerning authentication, secure transmission and attempted fraud detection?**
- **are there other considerations, for example in the way CRM works, or external constraints imposed by a third party like HiRisk?**

Using the intended architecture as a framework for discussing these questions encourages problems to be flushed out at an early stage. Note that the emphasis in this partial list is on what the business wants. Technology and then product selection can follow, with the technology chosen to ensure that it allows the business requirements to be met. Having the right skills available allows appropriate choices to be made, both for functional and for non-functional requirements, such as performance.

Choices may be affected by a variety of constraints. For example, a relevant technology for communicating with HiRisk might be Web Services. However, there may be a constraint that HiRisk's systems do not yet support Web Services — so an alternative may have to be selected, perhaps with provision to implement Web Services later. Those familiar with the technology can work with the architects and business experts to match the technology with the business need.

Also, note that, in the above, technology and product selection follow from a discussion of business requirements and the general characteristics of the environment in the context of an architecture. They do not precede it.

For example, we may need to ensure database integrity across systems. We need, therefore, to choose an approach and technology able to satisfy this requirement. We should not choose technology and then try to make the problem fit it.

The wrong place to start defining an architecture

The wrong place to start defining an architecture is at the bottom, by drawing up a list of preferred products and technologies and then working upward. This may sound obvious, but people do indeed do things the wrong way round. The extended batch run example may be a case in point. And I have seen just such an approach in a public sector project. Not only was the technology selected at the start, but specific products were identified. They were certainly not bad products; but they did not fit the requirements.

Starting at the bottom often results from quasi-religious preferences for a certain technology without regard to its appropriateness in the context in which it will be have to be used, and it ignores what is in place. Centralized government IT strategy organizations in the public sector often dictate technologies and products to be used in different departments or agencies. They may or may not prove to be suitable.

Let is now consider the non-functional requirements and the importance of quantifying requirements. The next section discusses these questions.

The importance of numbers

Let me start by making another obvious-sounding statement (it constantly amazes me how often the obvious is not done). A failure to quantify non-functional requirements leads to undesirable results.

In many ways, it is hard to conceive of many non-functional requirements without possessing some numbers like:

- **95% of responses should be within 2 seconds**
- **the availability during the operational day should be greater than or equal to 99.99%**
- **the average transaction rate is 35 per second, with twice that value in the peak hour**
- **the system must handle 100,000 users, of which 40,000 may be online at the same time.**

And so on. Yet I have encountered too many cases where the requirements quantified above have been stated as something like:

- **"the system has to respond quickly" and "we assumed you knew it must perform well" (what do 'well' and 'quickly' mean?)**
- **"it has to be reliable because it is an important application"**
- **"the system has to handle a high volume of transactions"**
- **"there is a large user base so the system must scale well".**

Sometimes partially quantified information is available, for example the number of users and the transaction volumes but not the response times or availability figures. And sometimes it is difficult: for example, estimating the number of users of Internet applications may be very hard to do, as the total population of possible users is enormous.

Not having quantified information leads to problems. First, it is not possible to know whether the system is working properly, because there is no way to measure it. How can 'well' or 'quickly' be measured? Experienced users may be able to state whether they think the system is working well or quickly, but for contractual reasons, such as acceptance tests, it is less than satisfactory.

A second, and more dangerous, set of problems is that a lack of clear requirements can lead to over- or under-engineering. Take availability, for example. Having no identified target leaves architects and designers with no guidance on what structures to put in place to ensure resilience. Do they at least duplicate everything, using clustering and other technologies? These solutions can be startlingly expensive. They can also be complicated and difficult to implement, in some cases I suspect leading to less stability than would be the case with a more simple-minded approach.

Without measurement, it is difficult or impossible to predict what may happen, where potential problems lie and how these can be fixed. It is, therefore, essential to identify and quantify performance metrics early on and to measure the system to determine the values of the metrics as soon as possible. Tuning and refinement follow the measurement, followed by more tests. This is an iterative, not a once-off, process. The results of this activity can then guide the remaining development, for example, in guidelines for efficient use of applications servers and databases. Test drivers are an essential component of this activity — to manage costs and to ensure repeatability of tests.

Management conclusion

This analysis has looked at one particular set of problems out of the many that cause IT projects to go astray: understanding infrastructure, or rather the lack of understanding. To deliver successful projects, it is essential to have a combination of knowledge combining architecture, business understanding and technology.

The skills, which may require several different people, are all required at the start of a project when the architecture is defined. The architecture documents themselves should be maintained throughout the life of the project. They form the connection between business requirements and suitable technology.

Technical skills are needed to ensure that selected technologies are appropriate to the task at hand and are correctly used. The business skills are necessary to ensure that the real purpose of the system is kept in view: solving a business problem or problems, not indulging in technology for the sake of it.

Finally, the importance of numbers cannot be overstressed. Whenever a requirement is really a quantifiable entity, it must be quantified. Throughput, response time and availability are examples of quantifiable entities. Without working to numbers, over or under-engineering can result, lead to unacceptable costs, or instability and lack of performance — in other words failure as Mr. Bye defined it at the start.

Enterprise service integration

Dr. Keith Jones
IBM Software Solutions Worldwide

Management introduction

The term 'Service Oriented Architecture' (SOA) has become increasingly popular in recent months as Web Services standards have matured and middleware implementations have been made available by a number of leading software vendors. The promise that inherently comes with SOA is that core business functions can be identified, characterized and encapsulated as services, using open standard technologies, for use in strategic business processes and re-use as new projects are funded over time.

The identification and re-use of service assets within an SOA should lead to better alignment between business and IT organizations within an enterprise and to greater responsiveness to change. All this depends upon the deployment of a middleware infrastructure that supports services as they are built into key business processes.

But what is the nature of this middleware infrastructure? What are the pragmatics of adoption as an SOA is achieved over time? In this analysis, Keith Jones reviews:

- ***the most popular approaches being taken by enterprises to building service infrastructures***
- ***the principle components needed in such infrastructures to cater for growth from a simple service configuration to some of the most sophisticated involving an Enterprise Service Bus (ESB).***

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

Migration to SOA

Many enterprises are now planning migration to a Service Oriented Architecture for some of their IT systems. A significant number of large enterprises have already started deployment projects. Some enterprises claim to have been using SOA principles for at least a decade. Others are just becoming aware of their potential value.

There is already a broad spectrum of thinking about architecture for business systems amongst the enterprise architect community. Most are convinced that if only software could be deployed as well-encapsulated, re-usable components with strongly cohesive, stable interfaces, then there might be a break-through in IT productivity, responsiveness and cost-effectiveness.

One reality for most enterprises is that deployed business systems have already been constructed using a wide variety of different and earlier architectures plus technologies. These have been built and/or bought over an extended period of time. Whilst each separate system has probably served its purpose well (and may still being doing so), a map of current business systems unfortunately often resembles a map of the London Underground network (which does not necessarily link readily to other transport networks like buses or planes, never mind other forms of train network).

A parallel reality is that the cost and complexity of understanding and maintaining those systems is increasing geometrically as enhancements are integrated to meet new and evolving business goals. The pragmatics of buy-[or build]-extend-and-integrate are failing because of that resulting geometric complexity.

An SOA may, therefore, present an attractive alternative for many enterprises because it offers a different discipline for constructing IT systems as they relate directly to business goals. Figure 6.1 illustrates the mappings behind this discipline:

- **from business model to required business processes**
- **from business process to required services**
- **from service to underlying components.**

As businesses begin an SOA journey they discover that there is:

- **a top-down approach to determining these mappings**
- **an alternative bottom-up approach to discovering them.**

For inherently practical reasons, the starting point for many enterprises is to identify certain legacy components in the bottom layer as potential services and then to expose them using standard (WSDL) interface descriptions. These service interfaces are then deployed by projects, often focusing initially on point to point integration between applications. This frequently results in an ad-hoc service layer that may or may not deliver real SOA value in the long term.

The alternative starting point for some enterprises is first to identify an important business domain in the top layer and then to break it down into business processes during re-engineering projects. By analyzing each process it is possible to determine:

- **which services are required**
- **how to implement them using new or existing underlying components.**

This produces a relatively well defined service layer for new business processes. On the other hand it is notoriously difficult to achieve for existing business processes.

In practice, most enterprises will find that that some combination of bottom-up and top-down approaches will be used as new projects are funded. In this evolutionary scenario there is often a mismatch between the services defined by top-down analysis and those exposed by the bottom-up projects. Some degree of adaptation, transformation, composition and re-engineering will inevitably be required over time within the process and service layers — as evolution toward an SOA is achieved.

If the principle value of migration to SOA is measured by re-use of core business functions (Figure 6.2) then a successful strategy will seek to maximize re-use as the number of services deployed increases over time. This should be true because higher re-use generally indicates greater productivity (as measured by lower IT costs for new or enhanced business functions) and shorter time-to-market (producing higher business revenues).

What is ‘Service Infrastructure’

To achieve SOA value there must be an IT infrastructure which is able to support deployment and operation of services. This must be available from the beginning of any implementation strategy and roadmap.

This infrastructure may be limited in both functionality and

scale at the start. But it must be capable of enrichment and extension over time, to satisfy growing SOA needs. The types of middleware component needed to build such an infrastructure are many and will vary — as requesters and providers are added to an SOA deployment.

Legacy application and data connectivity

To support the exposure of legacy applications and data sources, many different adapters will be needed. Each adapter will likely appear to be another client to the legacy system. But it will appear also to be a service provider, with a standard interface to requesters connected to the service infrastructure.

Many different adapters are already available from a variety of vendors for legacy systems — such as SAP applications, CICS transactions, Oracle databases and MOM networks. In the J2EE 1.4 arena there is a one standard (the J2EE Connector Architecture) for building and deploying adapters and another (Web Services for J2EE) for exposing Java components as services.

Some adapters will be deployed in J2EE containers and others run as standalone processes in front of legacy systems. Others, such as the SOAP adapter for CICS and batch COBOL adapters, will be integrated into the legacy system's infrastructure for performance and manageability.

Network connectivity

Additional service providers will be external to an SOA infrastructure but also connected using adapter technology (Figure 6.3). In some cases these providers will be reached via MOM networks. In others they will be reached using existing technologies, such as:

- **TCP/IP**
- **IIOP connections**
- **via SMTP servers.**

J2EE providers

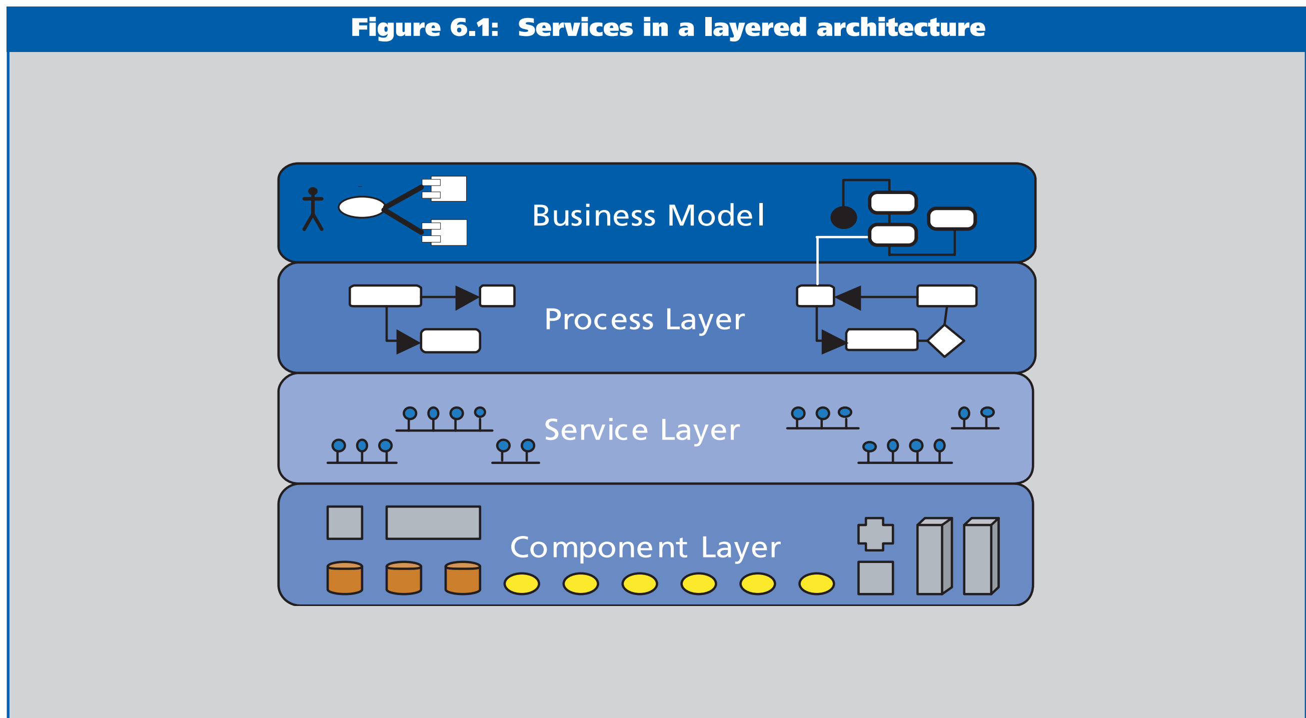
Some service providers will be more tightly integrated into the SOA infrastructure than others. For example, J2EE application servers will be used to contain familiar Java bean, servlet, session-EJB and message-driven EJB components as new services are deployed.

Such providers will be targets for communication using the J2EE JAX-RPC or JMS service APIs. A growing number of vendor application packages already deliver service provider implementations using these technologies and deployment patterns.

Message handling

Whilst it is true that many SOA deployments will not include examples of all these providers and their adapters, they will undoubtedly include support for handling incom-

Figure 6.1: Services in a layered architecture



ing SOAP standard service messages and invoking WSDL standard interfaces. The Apache AXIS model — which is based on the pipes and filters pattern — is the most likely model to be deployed for this basic infrastructure functionality.

An obvious foundation for service infrastructure is existing MOM technology, because this can be adapted to handle service messages and provide the necessary connectivity to application servers, adapters and legacy systems. WebSphere MQ and WebSphere Business Integration products provide this capability, as does SonicMQ and several other vendor products. However, it should be noted that it is not necessary to start with a MOM infrastructure for the simplest of SOA deployments.

Requester connectivity

A decent service infrastructure must allow for many different types of requester to be able to connect to available service providers. Many requesters will connect to infrastructure listeners that monitor TCP/IP ports and use HTTP sessions for incoming XML and SOAP messages. .Net and other clients will connect to legacy applications in this way within many common enterprise scenarios.

In some cases SOAP / HTTP requesters will be external to a particular service domain, such as those in affiliated enterprise divisions or in business partner organizations. These requesters will often connect using a gateway (Figure 6.3) to the service infrastructure — to provide a firewall to protect, monitor and secure enterprise services from misuse.

Service gateways

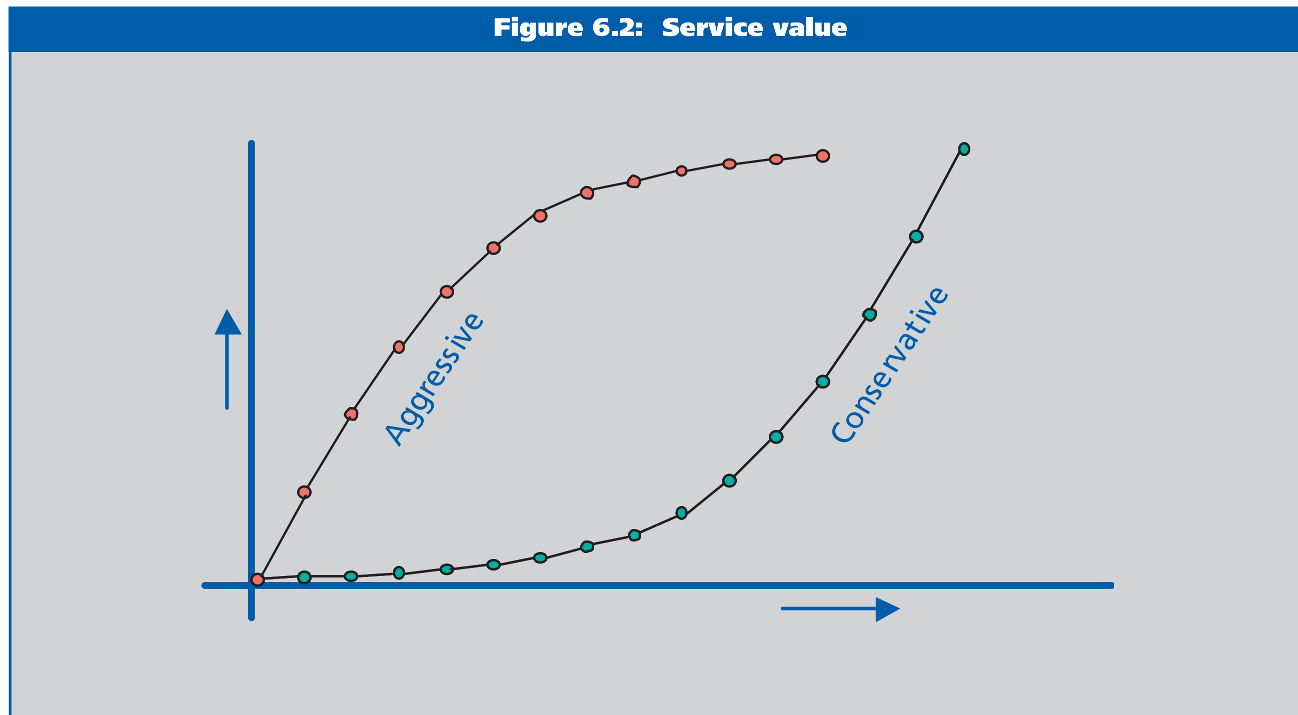
The gateway presents a service (or proxy) interface to external requesters. In turn it acts as requester to internal service providers.

A service gateway may also provide a control point for applying enterprise management policies — such as audit logging, billing and service load metering — in more sophisticated scenarios. Several service gateway and firewall products are available from software vendors for deployment in these scenarios.

J2EE requesters

An increasing number of service requesters will be deployed within J2EE containers attached to, and in many cases tightly integrated into, the service infrastructure. These requesters will use the J2EE JAX-RPC and JMS standard APIs to send and receive service messages as they interact with enterprise providers. A growing number of vendor supplied enterprise applications already include J2EE requesters that are ready to be connected into a service infrastructure.

Figure 6.2: Service value



In keeping with the recursive notion — that service providers may also be service requesters — the tight integration of application servers within a service infrastructure is especially important. Optimization of message handling paths between requesters and providers in the same application server (or server cluster) provides one of the greater benefits that can be achieved by tight integration. Other benefits include better autonomics and better management of the qualities of service. Furthermore, these can be achieved transparently for the business logic deployed.

Service flows

A special case for inclusion in a service infrastructure is the service flow engine. This implements standard (BPEL or BPELJ) flows that present a WSDL interface to requesters and make service requests to other providers. This is a special case because a flow may be connected to the service infrastructure as a:

- **provider (with access possibly to non-service-oriented components) in the component layer**
- **requester where it implements business processes in the process layer.**

The most common middleware technology that provides run-time flow capability is currently built into J2EE application server products. For example, IBM's WebSphere BI Server Foundation and Oracle's BPEL Process Manager offer implementations of such technology that are available with companion graphical tools for composing service flows.

Building an Enterprise Service Bus

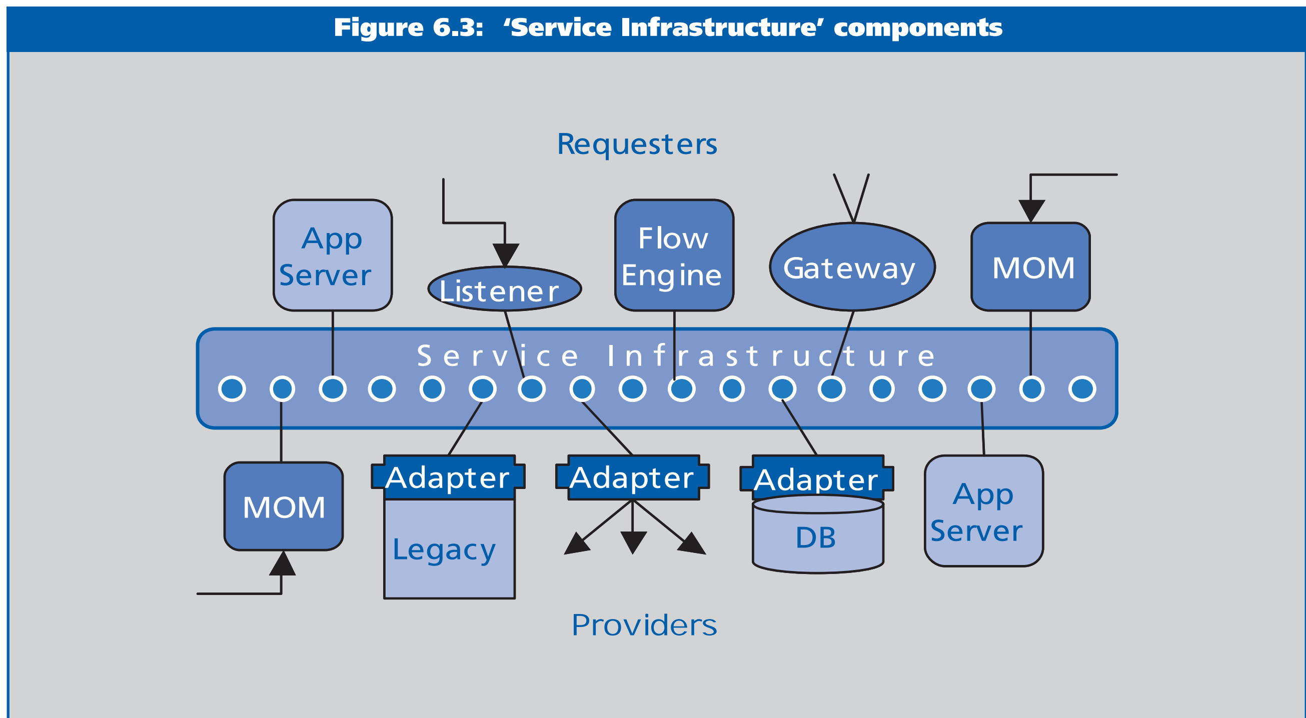
This analysis, thus far, has illustrated that service infrastructures may be relatively simple — particularly in the early days of an implementation strategy — and involve a small number of participating components acting as either requesters or providers or both. The simplest case is where one or more requesters interact with one or more providers in the same application server.

When the number of points of integration is small there is little need for the sophistication that caters for widely different service endpoints or large volumes of service messages. However, as the number of requesters and providers increases and their requirements become more varied, additional infrastructure is needed to:

- **facilitate growth (and, therefore, SOA value — Figure 6.2)**
- **provide a robust platform for well-managed enterprise-wide services.**

Such extended service infrastructure is commonly called an 'Enterprise Service Bus' (or ESB), see Figure 6.4 — in recognition of its capability to support plug-and-play attachment

Figure 6.3: 'Service Infrastructure' components



of service requesters and providers without any need to adapt the business logic included. As such, an ESB is both an architectural pattern and a run-time configuration of middleware components needed to deploy a service infrastructure.

In addition to the basic connectivity described for a variety of different types of requester and provider and possibly for large numbers of each in high volume enterprise scenarios, an ESB should provide virtualization of service resources so that neither providers nor requesters need to know:

- **where their counterparts are located**
- **what technology has been used for their implementation**
- **what transformation or adaptation is needed to comply with their interface requirements.**

In order to achieve this virtualization, ESB infrastructures will often include:

- **a service directory for WSDL interface descriptions**
- **dynamic allocation of service transport bindings**
- **dynamic assignment of service endpoints (ports) based on availability, reliability and other criteria associated with particular provider deployment specifications.**

A degenerate ESB infrastructure of this sort could be likened to a simple hub or gateway configuration into which a number of requesters and providers are connected.

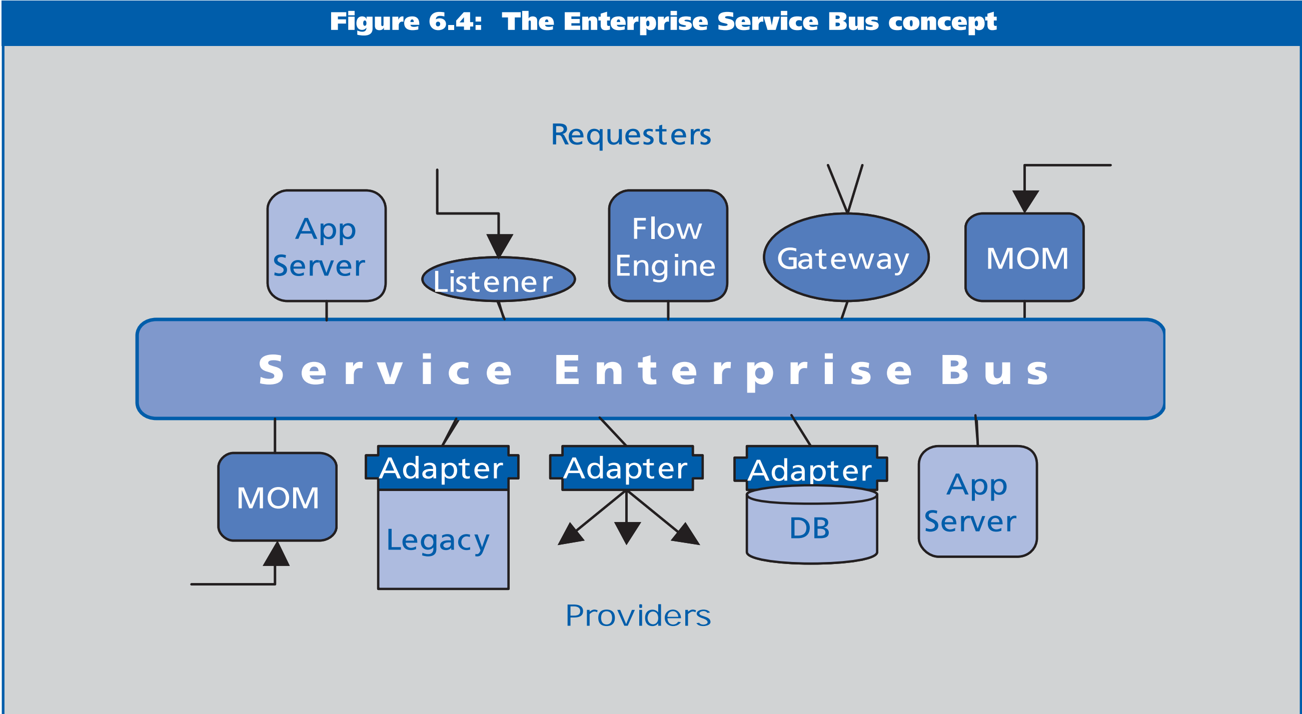
For large scale enterprise scenarios, ESB infrastructures should include the management capability to:

- **secure transactional rollback or compensation and**
- **provide automated service invocation,**
- **monitor compliance with service level agreements,**
- **provide failover and workload distribution**

for those services that are deployed across clusters of application servers.

Such sophistication goes far beyond the needs of most early SOA implementations. However, as projects deliver more services — and as service-oriented disciplines become engrained in both business and IT cultures at an enterprise level — the need for an ESB scales with success. A small number of enterprises have already reached this level of sophistication with leading software vendors making available most of the critical middleware components needed to build an ESB infrastructure.

Figure 6.4: The Enterprise Service Bus concept



First steps

For most enterprises, the first steps taken toward an SOA are those needed to grow basic skills in the use of available Web Services technologies. These skills are often grown on pilot projects that implement point to point integration of enterprise applications. However, without a set of guiding principles for reaching SOA goals and a roadmap for delivering valuable re-use of critical business functions, those first steps are probably a long way from success.

The next steps will probably be taken as business partners come to the table with integration proposals or there is a need to integrate application packages that now expose web service interfaces. These steps will likely be driven by the existence of previously defined WSDL interfaces and a business need (or needs) that must be satisfied before a top-down SOA strategy can be developed.

The first steps taken to implement an SOA that has been defined by top-down analysis of a business domain will frequently be to re-engineer a core business application system in order to expose re-usable business functions as services for future use. This scenario often involves decomposition of an existing application system (as shown in Figure 6.5). In such a scenario, the re-usable business functions (P1, P2, P3) are:

- **identified, isolated and characterized as services**
- **then deployed, using the available service infrastructure.**

The control logic in the original application will usually be re-engineered to become a service flow that invokes the newly exposed re-usable services. The same service flow may also be deployed as a provider attached to the same infrastructure for invocation by the entry logic re-engineered as the requester [R].

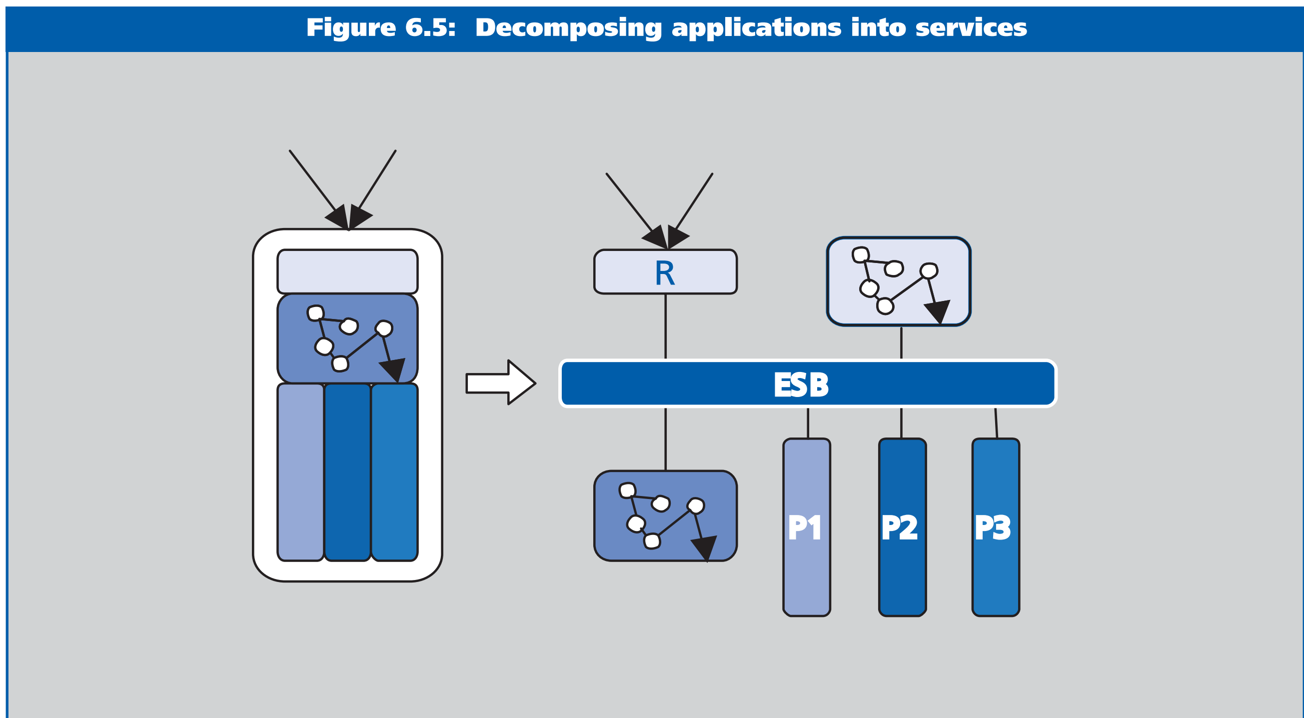
As successive service projects lead to an accumulation of re-usable services so the potential for re-use and SOA value increases over time:

- **re-engineering projects that focus on core business processes will benefit from this accumulation**
- **new business processes will become easier to model and implement using available re-usable services — if the service infrastructure is extended in concert with demand.**

SOA infrastructure evolution

There are many organizational and methodological issues to be resolved as an SOA strategy is established and business systems are re-engineered to become increasingly service-oriented. Large scale implementations founded on ESB middleware componentry will often evolve first within a

Figure 6.5: Decomposing applications into services



well defined domain, such as within a division of an enterprise organization, and then spread out to multiple domains federated by a common infrastructure (Figure 6.6).

In a fully developed scenario a number of different service domains may be linked to provide extensive use of shared service resources such as those shown in outline:

- **external partners will access selective enterprise business processes using service gateways established to enforce the appropriate management policies**
- **internal users will access enterprise business processes using service portals connected to the ESB infrastructure**
- **legacy and existing message-oriented applications will also be integrated using federated MOM and ESB infrastructures as re-engineering and re-use projects evolve.**

Management conclusion

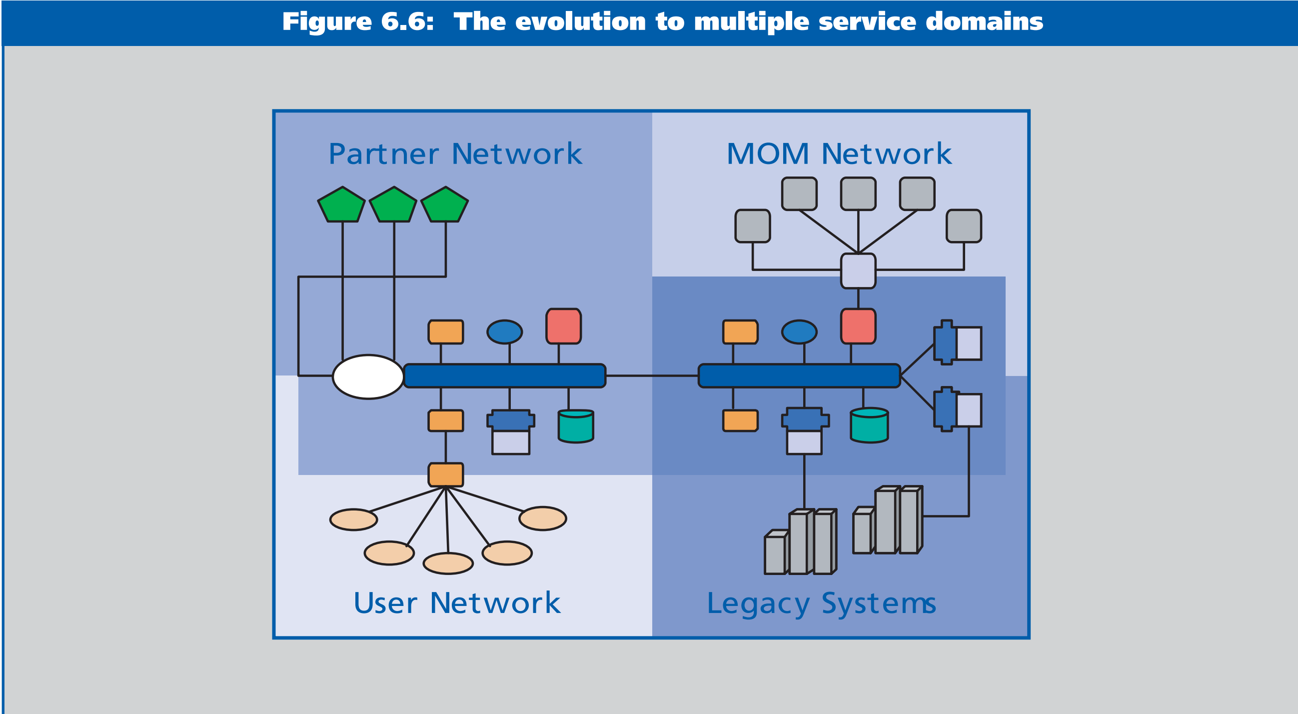
Today, many enterprises are evaluating the SOA concept and are ready to establish a roadmap for future projects, preferably one that will lead to re-use of core business functions as services. This re-use — at the coarse-grained level of business function — has a much better chance of success than earlier attempts at code re-use. An even greater chance of success attaches to re-use of the architectural patterns associated with the service oriented approach.

As services are identified and exposed within an enterprise scenario there is a growing imperative to establish an infrastructure that is capable of connecting widely different service requesters and providers using standard protocols and interfaces and carrying large volumes of standard service documents and messages. This infrastructure may:

- ***start as a small configuration of middleware components — such as a simple gateway***
- ***grow over time to become a fully populated multi-domain Enterprise Service Bus.***

The good news — as Dr. Jones articulates — is that many of the components needed are already available, and more are coming. In addition, positive experiences are already being delivered as leading enterprises that have already started on the road to SOA begin to reap the benefits from rationalizing middleware within a coherent approach to enterprise service integration.

Figure 6.6: The evolution to multiple service domains



Middleware, complexity and legacy application integration

Mark Lillycrop
Principal Analyst
Arcati

Management introduction

For a considerable number of years, we in IT have been trying to make IT more responsive to business needs by improving the level of integration between IT resources, applications and business processes. Integration has been at the top of the agenda in big companies for years, yet the sad reality is that most integration efforts seem unable to progress beyond a basic level of interoperability.

In the majority of cases, senior executives have a pretty clear idea what they want their IT systems to deliver. IT is now firmly accepted not just as a way to improve the delivery of information or expediting transactions, but as a critical way of building competitive advantage and keeping abreast of customer requirements.

With the growth of trends such as CRM (Customer Relationship Management), organizations have also come to realize that the value of information systems comes not from the systems themselves, nor even from the information held within them, but from the interaction between the processes and applications that enable this information to be manipulated in commercially advantageous ways — irrespective of where the data and applications are physically located. The retail store or financial services supplier that can analyze a customer's buying preferences and generate a specially customized offer on the fly greatly improves its chances of generating new revenue. Similarly, business-related events (generated by one application) may have implications for business processes running across other applications on other systems elsewhere in an organization. Indeed, these processes may influence major business decisions.

In both these cases, there is a real need for integration between applications and systems. In this analysis Mark Lillycrop considers not only how much integration is needed but at what level and cost. As he discusses, this depends on a whole range of business factors that need carefully to be considered.

All rights reserved; reproduction prohibited without prior written permission of the Publisher.
© 2004 Spectrum Reports Limited

The legacy issue

Integration is not only needed as a way of driving commercial advantage. In today's increasingly cautious and regulated business environment, integration is also key to ensuring tight security within the enterprise, tracking the movement and use of corporate assets and checking that all business processes adhere tightly to accounting and regulatory legislation.

With the level of complexity encountered in most large companies — with their many islands of computing developed at different times by different IT groups — the potential for IT resources to be used in an inappropriate way is enormous. End to end visibility of business processes is increasingly, and rapidly, becoming a legal necessity as well as a commercial imperative.

With the proliferation of widely-supported open standards — and the huge variety of integration tools and middleware that are now available — it may seem incredible that the task of achieving real, flexible integration of processes, applications and communications remains so difficult. The problem is that most companies and large organizations are faced with a whole range of 'legacies'. These continue to provide a growing obstacle to integration.

But what do we really mean by legacy? The term needs to be used with some caution.

For many senior IT managers, it means any technical architecture that has been in use within an organization for more than 10-15 years — and which has, consequently, slipped quietly from of boardroom consciousness. Quite often this means 'the mainframe', or whatever centralized server is sitting in the background quietly (and efficiently) managing the integrity of the company's transactional business and the security of its core data dssets.

This is, of course, far too simplistic a view of legacy systems. The sheer complexity of the modern IT environment is attributable to whole ranges of different legacies that have been introduced or grown up in layers over the years. All impact integration efforts in different ways. Four of the most obvious are organizational legacies, cultural legacies, mainframe legacies and legacies beyond the mainframe.

Organizational legacies

Organizational legacies arise in many and varied ways. Mergers and acquisitions are commonplace. Even in non-commercial areas, administrative departments and systems are regularly shuffled and recentralized depending on the financial and political expedients of the day.

The overall structure of an organization is often the best point from which to start considering legacy integration. When large businesses come together, there are often peripheral departments which remain poorly integrated with an organization's main operations. In consequence these can remain quite resistant to change or removal. Ensuring that IT processes become (and remain) consistent with those of other business areas can be one of any CIO's biggest headaches.

Cultural legacies

Cultural legacies are a second category. Even within a single company, there are often signs of cultural conflicts within the technical and business landscape.

For example, businesses often swing pendulously between centralized and distributed systems, or between insourced and outsourced IT solutions. These changes often depend on prevailing fashion or the personal preferences of individual finance and IT executives managers. Nevertheless, each swing leaves its mark. That mark contributes to the legacy of incompatible systems.

Mainframe legacies

Then there is the mainframe legacy. There are a whole range of technical legacy issues that plague large organizations amongst which the mainframe stands out. It is often regarded as the legacy system par excellence. To some extent, the accusation is justifiable, because many of the tools and applications that have built up within the mainframe software stack are highly proprietary in nature. Customers have been 'easily locked in' to products that have been in use for decades.

On the other hand, an architecture such as IBM's S/360 offers an unprecedented level of backward compatibility (mainframe COBOL applications written forty or more years ago can run, more or less undisturbed, on the latest zSeries processors) as well as extremely mature open interfaces for interacting and sharing data with other platforms.

That said, it is abundantly clear that relatively few mainframes are actually being replaced in large organizations. Even if new development is taking place on other platforms, and Web Services are replacing more traditional applications, it remains economically impractical to move the 70-80% of core operational data that is stored and managed within the existing mainframe environment.

For companies that value the scalability and resilience of the mainframe, and see it as a strategic platform for new

development, the core issues are how to integrate legacy and new applications within the mainframe architecture and simplify the whole infrastructure.

Legacies beyond the mainframe

Finally, in this list, there are the legacies beyond the mainframe. The analysts tell us that the composition of the data center environment is growing ever more complex, and will continue to do so for some years. Wintel, RISC/UNIX and Linux systems will grow rapidly, taking a larger proportion of the installed base. Mainframe MIPS will continue to expand, at a much slower rate in the background.

In this environment, the real problem will be with non-mainframe legacy systems, particularly the lack of compatibility between successive generations of Windows server and desktop products. For large corporate organizations, Microsoft's determination to incorporate extra function into its products, at the expense of compatibility, is arguably the greatest obstacle to integration and may yet prove to be the real reason why open Web Services become so popular.

Middleware and legacy integration: I

Every one of these areas can reduce the cohesion of the enterprise in general and make it increasingly difficult and expensive to bring IT applications and processes together. With the right direction at the highest level, it is possible to address both organizational and technical legacies. But it is rare for businesses to have the incentive or the financial resources to address them all at the same time.

For well over a decade now, middleware has been the main tool for integrating applications and systems across the enterprise. Like many technologies, middleware was not initially identified as an enterprise-wide solution, when the first integration tools and techniques began to appear.

Indeed integration itself was for a long time seen as an unavoidable add-on to an IT investment rather than a key part of the overall solution. Because of the tendency for user departments to turn their back on centralized data center services during the 1980s, client/server applications and central IT have often developed in parallel, with integration tools (for database access, message queuing, etc.) being applied ad hoc — only when a need arose for a specific data source or interface.

Not surprisingly, then, integration middleware has evolved in a somewhat haphazard fashion. Often it has contributed further complexity in its own right. Early integration tools were platform specific — intended to capture a data stream from one platform and convert it to a form suitable for use by another. In many cases, little or no thought was given to the idea of extending this adaptation/conversion capability to other applications. The integration tool in itself offered no clear return on investment; it was usually funded by the business unit that needed access to that specific legacy data.

Gradually, however, middleware and integration technology have matured — with EAI solutions emerging that offered a range of adapters for major application packages or database platforms that could be plugged into a central integration server. Message oriented middleware also developed from a highly specialized solution into a general purpose vehicle for providing asynchronous communication between dissimilar platforms. Businesses started to see the merits of treating integration as an enterprise-wide issue, and began to put technologies in place that would allow more flexible use of middleware tools.

Middleware and legacy integration: II

As we have moved from traditional EAI towards Web Services and that ever elusive concept, the Service Oriented Architecture, integration has become a much higher-level concern within the IT planning process. Individual platforms are now seen as peripheral contributors to the integrated delivery of IT services.

On the technical side, message brokers are being complemented and replaced by Enterprise Service Bus (ESB) products. These may yet prove to be superior because they seek to provide something much closer to a plug-and-play solution to integration. Today the whole approach to integration is driven by business strategy rather than by immediate technical needs.

From a business perspective, this evolution and maturity of integration is extremely welcome. Standards such as J2EE, .NET and XML are bringing all much nearer to a lingua franca for interconnecting and re-using IT resources, while Web Service standards (such as SOAP and UDDI) offer a straightforward approach to integrating and sharing IT resources on the fly.

At a lower level, however, the picture is not always so pretty. We talk about integrating legacy systems with newer applications. But most corporate networks contain many integration components that are legacies in their own right. Tools that were put in place several years ago — to translate and transform data or to provide a gateway between two application environments — may still be per-

forming that same low-level function. And, like the servers and operating systems with which they interact, these tools have been maintained and fine-tuned by product specialists who possess a highly focused knowledge of the way that they work.

If we take an enterprise-wide view of where these tools are, what role they are playing and how they are managed both as technical elements and hardware/software assets, it becomes apparent that they are contributing to the complexity of the IT — rather than relieving it. With increased complexity, costs rise — particularly people costs. Legacy integration specialists (for want of a better name) are not only rare and expensive; they are also (in many cases) distributed unevenly across organizations and, unlike their counterparts in strategic areas such as XML and Web Services, they often possess a low profile (and valuation).

Integration middleware — like data center platforms — is proliferating. Much of it is being buried deeper in the infrastructure rather than being replaced. This will have a significant effect on organizations that are trying to streamline their business processes and simplify the interfaces between IT resources.

The mainframe microcosm

Legacy integration middleware afflicts every part of the enterprise. But the mainframe provides a microcosm of complexity which illustrates how the problem has developed.

Writing recently in the 'Enterprise Data Center' newsletter, NEON Systems president Mark Cresswell talks about the numerous enterprises where highly critical mainframe subsystems — such as CICS, IMS, DB2, Adabas, IDMS and VSAM — are held together within a very delicate 'web' of integration tools, from multiple vendors, with no easy way of reducing the complexity. To address this problem, Mr.Cresswell calls for a unified, single-vendor approach to mainframe integration: "[What is needed is] a single enterprise product to support all z/OS subsystems, implement event-driven and service-oriented architectures, and handle and expose transactional controls to distributed applications. As well as this a single, consistent security implementation would be needed to ensure that J2EE and .NET based applications did not introduce vulnerabilities when bridging the gap between the mainframe and distributed worlds.

"Along with the single solution for mainframe integration should come a single interface to manage the transactions from the point they touch the integration software to the point they are returned to the calling application. In addition an architectural awareness is required to enable systems management tools to monitor the J2EE and .NET application platform and to query the instrumentation inside the mainframe integration software. This provides the systems administrator with the opportunity to visualize transactions from an end to end perspective within a single tool."

Management conclusion

This view of the simplified mainframe environment is an admirable goal. Large organizations will, however, find it hard to achieve. Many of the middleware tools (and their associated vendors) represented in this scenario are linked to the enterprise via tortuous service contracts that are as hard to replace as the products to which they apply. Moreover, as Mr. Cresswell points out, we need to be acutely aware of the many security vulnerabilities which may appear as we try to simplify the set of middeware tools that bridge legacy and newer applications.

Nevertheless, since cost reduction is a major objective for IT departments today, there is a strong argument for examining and re-examining the way that integration is achieved at every level — not just on and around the mainframe but in every sphere of IT operations. Large organizations are notoriously poor at asset management — keeping track of which products are in place, who chose them for what purpose and who (if anyone) still uses them.

Furthermore, asset management is particularly poorly applied in many areas of integration middleware. It is essential, therefore, that cost-conscious enterprises address this problem before they become overwhelmed by the complexity of their IT infrastructure. Web Services, Service Oriented Architectures (and the widely supported products that adhere to them) are understandably popular as the building blocks for today's integration solutions. Ignoring the hotchpotch of underlying tools is not a sensible long-term option for any forward-thinking enterprise.

Ultimately, the rapid growth of regulation and legislation within the business world may provide the impetus required to drive this process of simplifying IT integration. As Mr. Lillycrop has argued in this analysis, effective integration is an essential element in tracking the correct use and security of corporate data. Large enterprises need to take a long hard look at the way that this is achieved, and make sure that they are applying 'best practices' to every part of their IT infrastructure.

Members of the International Advisory Board

Charles C.C. Brett
President, C3B Consulting Limited & President, Spectrum Reports

William Donner
Fenway Partners

Kathryn Dzubeck
Executive Vice President, Communications Network Architects, Inc.

Ellen M. Hancock

Paul Hessinger
Vision UnlimITed

Pierre Hessler
Deputy General Manager, Cap Gemini

Michael Killen
President, Killen & Associates, Inc.

Dale Kutnick
Chairman, Meta Group, Inc.

Thomas Curran
Consultant

Norris van den Berg
General Partner, JMI Equity Fund, LP

Fiona A. Winn
Managing Editor & Publisher
Spectrum Reports

Additional contributors include:

Jay H. Lang
Distributed Computing Professionals

Keith Jones
IBM

David McGoveran
Alternative Technologies

Anura Gurugé
Consultant

Amy Wohl
Wohl Associates

Martin Healey
Technology Concepts Limited

Mark Allcock
J.P. Morgan Asset management

Aurel Kleinerman
MITEM

Chris Cotton
Consultant

Nick Denning
Strategic Thought

Yefim Natis
Gartner Group

Mike Beeston
Maven Associates

Mark Lillycrop
Arcati

Eric Leach
ELM

Randy Rhodes & Troy Terrell
Black & Veatch

Roy Schulte
Gartner Group

Mark Whitney
Delta Technologies

Jim Johnson
Standish Group

Tom Curran
TC Management

Alfred Spector
IBM Corporation

Max Dolgicer
International Systems Group, Inc.

Peter Bye
Unisys Systems and Technology

Steve Ross-Talbot
Enigmatec

Peter Houston
Microsoft Corporation

Jeff Tash
Database Decisions

Ed Cobb
BEA Systems

Bernard Abramson
Consultant

Geoff. Norman
Xephon

Jim Gray
Microsoft Research

Wayne Duquaine
Grandview DB/DC Systems

Steve Craggs
Saint Consulting

Tom Welsh
Consultant

Gustavo Alonso
Swiss Federal Inst. of Technology

Mike Gilbert
Micro Focus

Tony Leigh
Sensima Technologies

MIDDLEWARE**SPECTRA**
is published and distributed worldwide by:

USA and Canada:
Spectrum Reports, Inc.

Subscription Center
PO Box 32510,
Fridley, MN 55432, USA
Telephone: 763 502 8819
Fax: 763 571 8292

UK and Rest of the World:
Spectrum Reports Limited

Research and Editorial Office
St Swithun's Gate, Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Subscription Centre
St Swithun's Gate
Kingsgate Road
Winchester SO23 9QQ
England
Telephone: +44 1962 878333
Fax: +44 1962 878334

Email and Internet

Email:
spectrum@ middlewarespectra.com

World Wide Web:
www.middlewarespectra.com

ISSN 1356-9570

[incorporating FINANCIAL MIDDLEWARESPECTRA ISSN 1460-7220]