Enterprise Service Bus Scenarios and Solution Patterns in Service Oriented Architecture

Implementing an ESB in Today's Technology

Rick Robinson

IT Architect

Architecture Services, EMEA WebSphere Lab Services, IBM UK

rick_robinson@uk.ibm.com

10th May 2004

Contents

1 Introduction

The current state of the art in IT integration is the implementation of Service Oriented Architectures (SOA) using Web Services technologies, and many excellent descriptions of their benefits and practise are available (see References 1 to 4). More recently, the concept of an "Enterprise Service Bus" (ESB) has been expressed as a key component of the SOA infrastructure (see References 4 to 6). However it is important to clarify whether the ESB is a product, a technology, a standard or something else. In particular, is the Enterprise Service Bus something that can be built today, and if so, how?

This paper describes the Enterprise Service Bus as providing a set of infrastructure *capabilities*, implemented by middleware technology, that enable Service Oriented Architecture. The ESB supports service, message and event-based interactions in a heterogeneous environment with appropriate service levels and manageability. A variety of ESB capabilities required to achieve this are summarised and categorised; however, not all of them are required in every situation in which some form of ESB can deliver value.

This paper therefore identifies a set of minimum capabilities that fulfil the most basic needs for an Enterprise Service Bus consistent with the principles of Service Oriented Architecture. Identifying these minimum capabilities allows us to identify which existing technologies can be used to implement an ESB to support a Service Oriented Architecture. By then considering how the requirements of a specific situation indicate the need for additional capabilities, we can choose the most appropriate implementation technology for that situation.

In order to assist organisations wishing to take this approach we have defined a set of ESB Scenarios in Service Oriented Architecture capturing common starting points for ESB or SOA implementation. The scenarios can be analysed using A Capability Model for the Enterprise Service Bus and some common Issues Driving ESB Architecture and Design Decisions in order to select suitable Solution Patterns. The solution patterns in turn provide guidance for choosing appropriate implementation technologies.

It is likely that the capabilities required of the ESB in any particular solution will evolve incrementally over time as both the solution and the uses to which it is put evolve and mature. Similarly, the availability and capability of explicit ESB products will also mature. We therefore consider A Roadmap for SOA and ESB Adoption to guide the initial use of ESB capabilities and technologies, and illustrate options for such an incremental approach.

Throughout this paper, references are made to various IBM software products and technologies; links to further information can be found in References below.

1.1 Acknowledgements

This paper would not exist without the discussions the author has been involved in with the following people, all of whom contributed to the ideas and thinking behind it:

Beth Hutchison, Rachel Reinitz, Olaf Zimmerman, Helen Wylie, Kyle Brown, Mark Colan, Jonathan Adams, Paul Fremantle, Keith Jones, Paul Verschueren, Daniel Sturman, Scott Cosby, Dave Clarke, Ben Mann, Louisa Gillies, Eric Herness, Bill Hassell, Guru Vasudeva, Kareem Yusuf, Ken Wilson, Mark Endrei, Norbert Bieberstein, Chris Nott, Alan Hopkins and Yaroslav Dunchych.

1.2 The Role of the Enterprise Service Bus Within a Service Oriented Architecture

Whilst we will not discuss the definition of Service Oriented Architecture in depth, as that has been covered elsewhere (see References 1 to 4), it is useful to summarise here the principles that most descriptions of SOA agree with:

- The use of explicit implementation-independent interfaces to define services.
- The use of communication protocols that stress location transparency and interoperability.
- The definition of services that encapsulate reusable business function.

These principles are illustrated in **figure 1** below; and note that whilst the Web Services standards are an excellent match to these principles, they are not the only technology consistent with them.

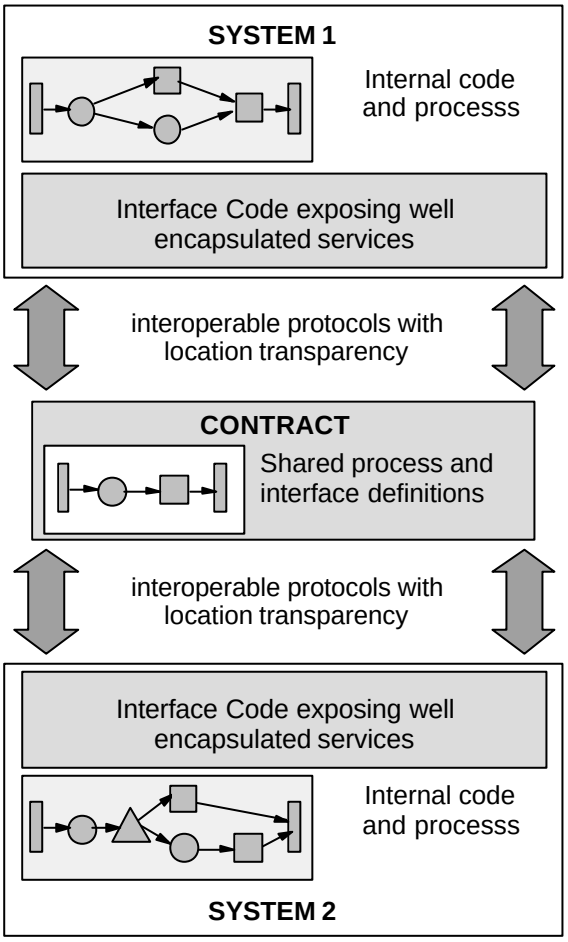

Figure 1: The principles of Service Oriented Architecture

In order to implement an SOA, both applications and infrastructure must support the SOA principles. Enabling applications involves the creation of service interfaces to existing, or new, functions, either directly or through the use of adaptors. Enabling the infrastructure at the most basic level involves the provision of capability to route and transport service requests to the correct service provider. However, it is also vital that the infrastructure support the *substitution* of one service implementation by another with no effect to the clients of that service. This requires not only the service

interfaces specified by SOA, but also that the infrastructure allows client code to invoke services in a manner independent of the service location and communication protocol involved. Such service routing and substitution are amongst the many capabilities of the Enterprise Service Bus (ESB).

The Enterprise Service Bus supports these service interaction capabilities and provides the integrated communication, messaging and event infrastructure to enable them, thus combining the major enterprise integration patterns in use today. The ESB provides an infrastructure for SOA consistent with the needs of the enterprise to provide suitable service levels and manageability, and to operate in a heterogeneous environment.

The remainder of this paper will discuss this role of the ESB in SOA, including the capabilities it provides beyond basic routing and transport, as described in A Capability Model for the Enterprise Service Bus below. The roles of the ESB in supporting message and event driven architectures are subjects in themselves, and beyond the scope of this paper.

The ESB is sometimes described as a *distributed* infrastructure, and contrasted as such against solutions, such as message broking technologies, which are commonly described as “hub and spoke”. However, this is a false distinction: two different issues are being addressed: the *centralisation of control* and the *distribution of infrastructure*. Both ESB and “hub and spoke” solutions centralise control of configuration such as routing information, service naming etc. Similarly, both may be deployed in a simple centralised infrastructure, or a more sophisticated distributed manner. **Figure 2** below illustrates this point.

Of course, different technologies will have different constraints on the physical deployment patterns they support – some might be suited to very widespread distribution to support integration over large geographical areas, whilst some might be more suited to deployment in localised clusters to provide availability and scalability. Matching the requirements for physical distribution to the capabilities of candidate technologies is an important aspect of ESB design. Also important is the ability to incrementally extend the initial deployment to reflect evolving requirements to integrate additional systems or extend the geographical reach of the infrastructure.

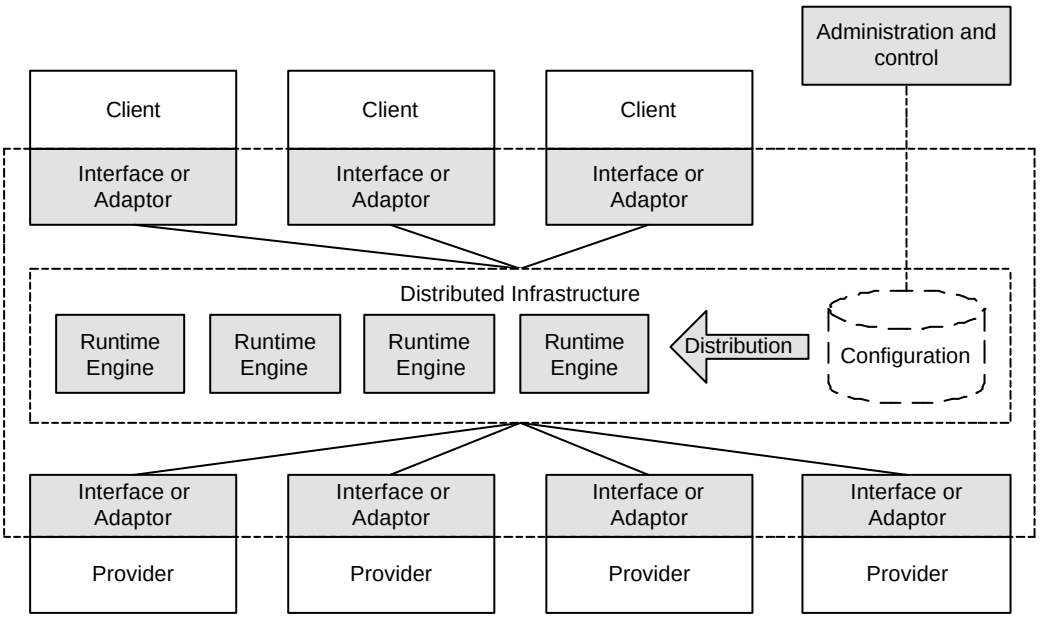

Figure 2: Centralised Control over Distributed “Bus” Infrastructure

We should also position the ESB in relation to other components in the SOA infrastructure, in particular the Service Directory, Business Service Choreography and Business to Business (B2B) Gateway components. Since these components are not strictly required by the SOA principles above, we will treat them as optional components. **Figure 3** below shows a Service Oriented Architecture indicating the relationship of these components to the ESB.

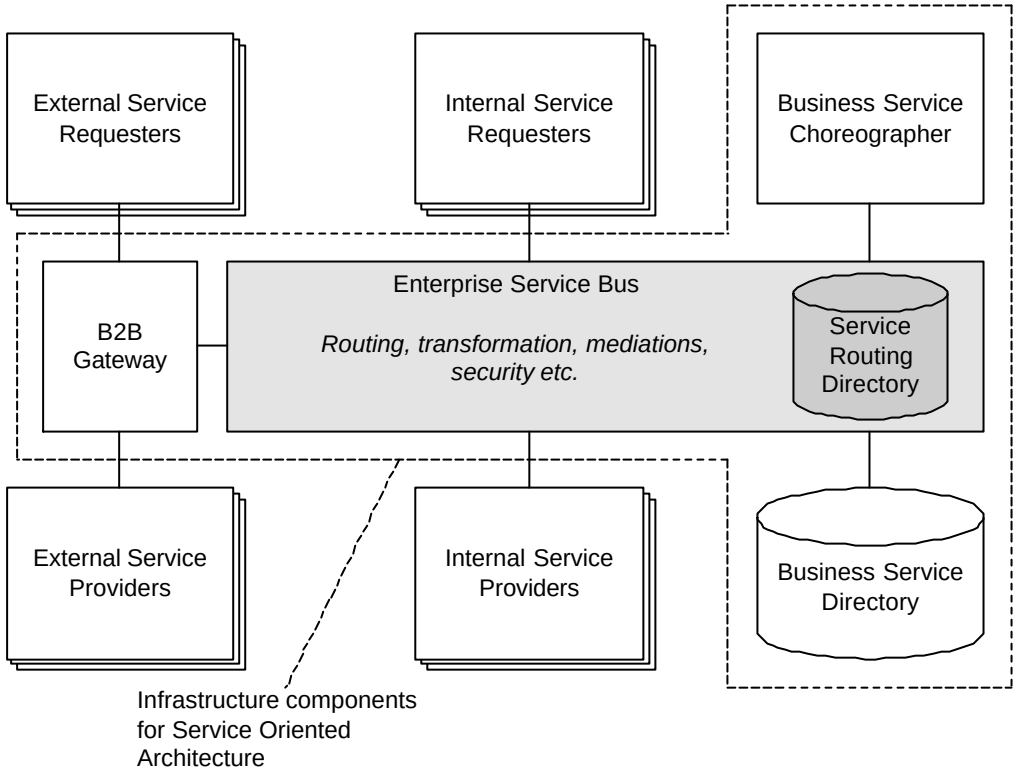

Figure 3: The Role of the Service Bus in a Service Oriented Architecture

Some form of service routing directory is required by the ESB in order to route service requests. However, an SOA may also have a separate "business service directory", which at its most basic might be a design-time service catalogue used to achieve re-use of services across the development activity of an organisation. The vision of Web Services places a UDDI directory in both the "business service directory" and "service routing directory" roles, enabling the dynamic discovery and invocation of services. Such a directory might be viewed as part of the ESB; however, until such solutions become common, the "business service directory" is likely to be separate from the ESB.

The role of the Business Service Choreographer is to choreograph *business services* into *business processes*; it will therefore invoke services through the ESB, and will itself expose business process as services available to clients, again through the ESB. However, its role in choreographing *business* processes and services identifies the Business Service Choreographer as separate from the ESB, which is an infrastructure technology.

Finally, the role of a B2B Gateway component is to make the services of two or more organisations available to each other in a controlled and secure manner. It is useful to view such components as integrated to the ESB, but not part of it. Whilst gateway *technologies* exist that provide capabilities suitable for implementing both "B2B Gateway" components and the ESB, the *purpose* of the "B2B Gateway" component

separates it from the ESB. Indeed this purpose might also require additional capabilities, such as partner relationship management, that would not be part of an ESB and might not be supported by ESB technologies.

1.2.1 A Capability Model for the Enterprise Service Bus

The table below summarises and categorises some of the Enterprise Service Bus capabilities identified in existing literature (see References 4 to 6). Whilst some are quite basic, some (such as autonomic or intelligent capabilities) represent significant steps towards an On Demand Operating Environment (see reference 8 in the References). It is important to recognise that most current ESB scenarios require only a subset of these capabilities from a subset of the categories; the *minimum* capabilities required for an ESB implementation are considered in The Minimum Capability Enterprise Service Bus Implementation below.

Communications	**Service Interaction**
• Routing • Addressing • Communication technologies, protocols and standards (e.g. WebSphere MQ, HTTP, HTTPS) • Publish / subscribe • Response / request • Fire & forget, events • Synchronous and asynchronous messaging	• Service interface definition (e.g. WSDL) • Support for substitution of service implementation • Service messaging models required for communication and integration (e.g. SOAP or Enterprise Application Integration middleware models) • Service directory and discovery
Integration	**Quality of Services**
• Database • Service aggregation • Legacy and application adaptors • Connectivity to enterprise application integration middleware • Service mapping • Protocol transformation • Application server environments (e.g. J2EE and .Net) • Language interfaces for service invocation (e.g. Java, C/C++/C#)	• Transactions (Atomic transactions, Compensation, WS-Transaction) • Various assured delivery paradigms (e.g. WS-ReliableMessaging or support for Enterprise Application Integration middleware)
Security	**Service Level**
• Authentication • Authorisation • Non-repudiation, • Confidentiality • Security standards (e.g. Kerberos, WS-Security)	• Performance • Throughput • Availability • Other continuous measures that might form the basis of contracts or agreements
Message Processing	**Management and Autonomic**
• Encoded logic • Content-based logic • Message and data transformations • Validation • Intermediaries • Object identity mapping	• Service provisioning and registration • Logging, metering and monitoring • Discovery • Integration to systems management and administration tooling • Self-monitoring and self-

• Data enrichment	management
Modelling	**Infrastructure Intelligence**
• Object modelling • Common business object models • Data format libraries • Public vs. private models for business-to-business integration • Development and deployment tooling	• Business rules • Policy driven behaviour particularly for service level, security and quality of service capabilities (e.g. WS-Policy) • Pattern recognition

Many of these capabilities can be implemented either using proprietary technologies or through the use of open standards. However, the various technology candidates for ESB implementation in a given case might vary considerably in their performance, scalability and availability characteristics, as well as in the ESB capabilities and open standards they support. Because of this, and the fact that some of the relevant standards are recent or still emerging, many critical decisions in implementing the Enterprise Service Bus today are concerned with the trade-off between more mature, proprietary technologies and less mature open-standard support.

We will not discuss each of these capability categories in detail in this paper; rather, we will focus on those that drive decisions between different approaches to adopting or implementing an ESB. In particular, in the next section, we will discuss what are the *minimum* capabilities that an ESB requires in order to support a Service Oriented Architecture.

1.2.2 The Minimum Capability Enterprise Service Bus Implementation for Service Oriented Architecture

If only a subset of the capabilities identified above are relevant to most SOA scenarios, we need to consider what is the *minimum set* of capabilities that are required in order to implement an Enterprise Service Bus. In order to do this, consider the most commonly agreed elements of the ESB definition:

- *The ESB is a logical architectural component that provides an integration infrastructure consistent with the principles of Service Oriented Architecture.*
- *SOA principles require the use of implementation-independent interfaces, communication protocols that stress location transparency and interoperability, and service definitions that are relatively large-grained and encapsulate reusable function.*
- *The ESB may be implemented as a distributed, heterogeneous infrastructure.*
- *The ESB provides the means to manage the service infrastructure and the capability to operate in today's distributed, heterogeneous environment.*

The minimum ESB capabilities can be defined from those principles, and are shown in the table below.

Communication	Integration
• Routing and addressing services providing location transparency. • An administration capability to control service addressing and naming. • At least one form of messaging paradigm (e.g. request / response, pub/sub, etc.). • Support for at least one transport protocol that is or can be made widely available.	• Support for multiple means of integration to service providers, such as Java 2 Connectors, Web Services, asynchronous messaging, adaptors etc.
Service Interaction	
• An open and implementation independent service messaging and interfacing model, that should isolate application code from the specifics of routing services and transport protocols, and allow service implementations to be substituted.	

Note that these minimum capabilities *do not require the use of particular technologies*, such as Enterprise Application Integration Middleware, Web Services, J2EE, or XML. The use of those technologies is very likely as they fit the requirements well, but it is not mandatory. Conversely, the minimum capabilities are nearly, but not wholly, fulfilled by the simple use of SOAP/HTTP and WSDL:

- URL addressing and the existing HTTP and DNS infrastructure provide a "bus" with routing services and location transparency.
- SOAP/HTTP supports the Request / Response messaging paradigm.
- The HTTP transport protocol is widely available.
- SOAP and WSDL are an open, implementation independent messaging and interfacing model.

However, this basic use of SOAP/HTTP and WSDL is really point-to-point integration, and does not fulfil some key capabilities required for an Enterprise Service Bus:

- There is no "administration capability" to control service addressing and naming; rather, service names are controlled individually by each adaptor, and service routing control is dispersed between the addresses invoked by service clients, the HTTP infrastructure, and the service names assigned to adaptors.
- Whilst dependent on implementation details, this approach does not tend to facilitate substitution of service implementations; service requester code (perhaps generated by development tools) will often bind directly to specific service provider implementations through specific protocols at specific addresses. Substituting one service implementation for another will hence require changes to and redeployment of application code.

Of course, further capabilities are required in many or even most scenarios, and will become increasingly common. In particular, the following types of requirement are likely to lead to the use of more sophisticated technologies, either now or over time:

- Quality of Service and Service Level capabilities.
- Higher level Service Oriented Architecture concepts, e.g. service choreography, directory, transformations etc.
- On Demand Operating Environment demands, such as Management and Autonomic capabilities and Infrastructure Intelligence capabilities.

- Truly heterogeneous operation across multiple networks, multiple protocols, multiple domains of disparate ownership.

1.3 Security Issues Affecting the Enterprise Service Bus

In this paper, we won't directly address security requirements; however, they are likely to be important to the choice of ESB technology. For example, if *no* authentication or authorisation of service requests is required, the choice of technology will be very broad. If, as is more likely, some level of security is required, it is important to assess what style of security will be acceptable, e.g.:

1. Is security in the communications infrastructure acceptable, e.g. the use of SSL mutual authentication between Enterprise Application Integration middleware servers, or through the use of the HTTPS protocol?
2. Is individual, point-to-point security acceptable between participating systems, or is an end-to-end model required? For example, is there a need to propagate client identity through intermediate systems such as brokers to the end-providers of service implementations?
3. Is security in the application layer acceptable, e.g. such that client code performs basic HTTP authentication with a userid and password, or that it passes such information to the service as application data?
4. Is some form of standard security required such as Kerberos or WS-Security?

Whilst all of these approaches are possible, the industry direction is towards standards-compliant (i.e. WS-Security) security features supported by infrastructure and middleware. However, these standards are relatively recent, and product support for them is emerging rather than established, particularly where interoperability is concerned. A priority of any ESB architecture should hence be to establish the security requirements as early as possible so that they can be included in the choice of implementation technology.

2 ESB Scenarios and Analysis

The starting points for many SOA and ESB implementations are described in the section ESB Scenarios in Service Oriented Architecture below. Each scenario indicates specific Issues Driving ESB Architecture and Design Decisions that drive the selection of Solution Patterns, described in the next section of this paper. The solution patterns represent an evolution from a basic use of service technology, through simple ESB implementations to a sophisticated Service Oriented Architecture infrastructure.

The scenarios are not meant to represent the totality of an organisation's requirements for SOA or ESB, either at a moment in time or as they evolve. For example, whereas one of the scenarios such as Basic Integration of Two Systems might seem a good match to a particular current requirement, that requirement might evolve over time into something more sophisticated, such as Enable Wider Connectivity to One or More Applications. Alternatively, there may be many separate requirements for SOA or ESB infrastructure, which individually match simple scenarios but overall represent something more complex.

There is a balance to be struck here between implementing a solution that meets the immediately apparent needs, attempting to anticipate future needs, and defining a consistent solution across an Enterprise. It may be appropriate to recognise the organisation's needs as a whole as a relatively sophisticated scenario such as Implement a Service Oriented Architecture Infrastructure with High Quality of Service and Web Services Standards Support; an alternative would be to treat individual situations separately as simple scenarios, but define a roadmap for evolving the resulting solutions over time into a common infrastructure.

2.1 ESB Scenarios in Service Oriented Architecture

The scenarios characterised in the sections below are:

- Basic Integration of Two Systems
- Enable Wider Connectivity to One or More Applications
- Enable Wider Connectivity to Legacy Systems
- Enable Wider Connectivity to an Enterprise Application Integration Infrastructure
- Implement Controlled Integration of Services or Systems Between Organisations
- Automate Processes by Choreographing Services
- Implement a Service Oriented Architecture Infrastructure with High Quality of Service and Web Services Standards Support

2.1.1 Basic Integration of Two Systems

Scenario	
There is a business need to provide basic integration between two systems implemented in different technologies such as J2EE, .Net or CICS etc. The Web Services SOAP standard or messaging middleware might be candidate integration technologies. Whichever technology is chosen will need to be supported to some extent in the environments of both applications. An important question in this scenario is whether the need to integrate additional systems is likely to arise in the future – the use of an extensible solution in the first place will facilitate support for future requirements, but this needs to be balanced against the initial need to solve a simple problem.	
Most Relevant Issues	**Relevant Solution Patterns**
1,3,4,6,10,13	• Implement basic integration using wrappers or adaptors - Basic Adaptors • Or, with future expansion in mind, either: o add a controlling Service Gateway o or implement a sophisticated infrastructure - Web Services Compliant Broker or Enterprise Application Integration Infrastructure for Service Oriented Architecture

2.1.2 Enable Wider Connectivity to One or More Applications

Scenario	
Existing packaged applications (e.g. CRM, ERP etc.) or bespoke applications, perhaps implemented in J2EE or other application server environments, provide functions that are useful beyond the applications themselves. There is value in exposing these functions as services either to enable the applications to interoperate with each other, or to provide access to new channels or clients. The use of interoperable or open standard communication and service protocols seems the best way forward.	
Most Relevant Issues	**Relevant Solution Patterns**
1,2,3,4,6,8,9,10, 11,12,13,14	• Implement basic integration using wrappers or adaptors - Basic Adaptors • Or add a controlling Service Gateway • If more sophisticated capabilities are required implement - Web Services Compliant Broker or Enterprise Application Integration Infrastructure for Service Oriented Architecture • If process choreography is also required, implement Service Choreographer or Full Service Oriented Architecture Infrastructure

2.1.3 Enable Wider Connectivity to Legacy Systems

Scenario	
An organisation has a large investment in “legacy” technologies (such as CICS, IMS etc.) supporting applications that provide their core business transactions and data access. There is significant value in providing interoperable or open standard, service-based access to those transactions (e.g. transactions that query account balance, create orders, schedule or track deliveries, query stock levels etc.).	
Most Relevant Issues	**Relevant Solution Patterns**
1,2,3,4,7,8,9, 10,11,13,14	• Implement basic integration using wrappers or adaptors - Basic Adaptors • Or, with future expansion in mind, either: o add a controlling Service Gateway o or implement a sophisticated infrastructure - Web Services Compliant Broker or Enterprise Application Integration Infrastructure for Service Oriented Architecture

2.1.4 Enable Wider Connectivity to an Enterprise Application Integration Infrastructure

Scenario	
There is an existing Enterprise Application Integration infrastructure, such as WebSphere Business Integration Server, to which extended access based on interoperable protocols or open standards is required. Whilst the exposure of service interfaces defined in terms of XML business data through a highly interoperable protocol such as HTTP or WebSphere MQ might provide an appropriate level of interoperability in some scenarios, support for the WSDL and SOAP Web Services standards might be required if neither a bespoke nor proprietary extension to the existing scope of integration are acceptable.	
Most Relevant Issues	**Relevant Solution Patterns**
3,4,5,8,9,11,13, 14	• Extend the EAI infrastructure using open data formats - Enterprise Application Integration Infrastructure for Service Oriented Architecture • Add a Service Gateway • Or add open standard support to the infrastructure - Web Services Compliant Broker

2.1.5 Implement Controlled Integration of Services or Systems Between Organisations

Scenario	
An organisation wishes to enable its customers, suppliers or other partners to integrate directly with functions provided by one or more applications, legacy or otherwise. A point of control is required to provide secure, manageable access from external parties to those applications. The use of open standards is preferred as the organisation has no direct control over the technologies used by its partners. This scenario might apply either between separate organisations, or between units of a larger distributed organisation.	
Most Relevant Issues	**Relevant Solution Patterns**
1,2,3,4,6,8,9,10, 11,13,14	• Add a Service Gateway • Or if more sophisticated capabilities are required - Web Services Compliant Broker

2.1.6 Automate Processes by Choreographing Services

(Note: this scenario can be considered an evolution of the Enable Wider Connectivity to One or More Applications scenario).

Scenario	
Existing packaged applications (e.g. CRM, ERP etc.) or bespoke applications, perhaps implemented in J2EE or other application server environments, provide functions that are useful beyond the applications themselves. These functions can be exposed as services using interoperable or open standard communication and service protocols so that the applications can interact. The interactions combine to form business processes at some level, and these processes should be explicitly modelled and executed using appropriate modelling and process execution technology, possibly in compliance with appropriate open standards.	
Most Relevant Issues	**Relevant Solution Patterns**
1,2,3,4,6,10,11, 12,13,14	• If the direct connection of services is possible, implement Service Choreographer • If more sophisticated integration or control are required, implement a Full Service Oriented Architecture Infrastructure

2.1.7 Implement a Service Oriented Architecture Infrastructure with High Quality of Service and Web Services Standards Support

Scenario	
This scenario is a composite of the preceding scenarios. It represents the need to enable widespread internal or external access to services provided by multiple applications, legacy or otherwise. Various security, aggregation, transformation, routing and service choreography capabilities are required. It is worth noting that this scenario is often driven by an IT organisation in response to increasing demands from across the business it supports to enable more widespread, more flexible integration between business systems.	
Most Relevant Issues	**Relevant Solution Patterns**
All	• Implement a Full Service Oriented Architecture Infrastructure

2.2 Issues Driving ESB Architecture and Design Decisions

In order to identify a suitable solution pattern and implementation technology for an ESB, requirements for specific ESB capabilities will need to be analysed in more detail than is encapsulated in the basic ESB Scenarios in Service Oriented Architecture above. The following questions are intended to aid this process, and the specific questions relevant to each scenario are indicated in the preceding section.

1. Are the existing functions and their data interfaces good matches to the services you want to provide, or can appropriate modification or aggregation be performed in the applications?
    - If not, transformation or aggregation capability will be required either in adaptors or the ESB infrastructure, or will have to be performed by service clients.
2. Should the services be exposed in the form of some common business data model, and if so, do the systems implementing those services already support that model, or can they be made to do so?
    - If not, transformation or aggregation capability will be required either in adaptors or the ESB infrastructure.
3. Are open standards required, or can appropriate interoperability be achieved through Enterprise Application Integration middleware? If open standards are required, which ones are appropriate?
    - Whilst the use of open standards is one way to achieve interoperability, proprietary Enterprise Application Integration middleware is also highly interoperable, and often significantly more mature. Many organisations also have extensive existing infrastructures which can, in some scenarios, minimise the benefits of open standards.
    - In scenarios where open standards are required, Web Services are perhaps the most obvious choice in this context. However, Java Messaging Service

(JMS), JDBC, basic XML or several other technologies such as EDI or industry XML formats can also be applied.

- In practise, interoperability between different implementations of the same standards cannot always be assumed, particularly if the standards are recent or emerging. In the case of Web Services, the Web Services Interoperability Organisation has recently published the Basic Profile for interoperability using SOAP and WSDL, and other profiles for more advanced standards will follow (e.g. WS-Security, WS-Transaction etc.). Until such profiles are comprehensive, established and widely supported by products, the use of open standards will not guarantee, and may not always facilitate, interoperability.

4. Is support for basic communication protocols and standards (e.g. WebSphere MQ, SOAP, WSDL) required, or more capabilities such as WS-Security, WS-Transaction etc.)?

   - Requirements to support more sophisticated standards will impose more stringent constraints on the options for implementation technologies, and may imply the use of less mature technologies.

5. Where changes to the message formats and protocols used by an existing infrastructure are under consideration, including the adoption of open standards, are the changes required throughout the existing infrastructure, or can they be applied at the edges? If EAI technology is in use or under consideration, does that have it's own internal format, or can it process open standards as an internal format?

   - Any use of open standards is likely to be driven by needs to extend access, so it is usually more important that they are available at the interfaces to existing infrastructure than that they are used internally.
   - If internal use of specific formats, technologies or standards is required, this will place constraints on the choice of implementation technology.

6. Do the systems implementing functions which should be exposed as services support the required technologies or open standards such as SOAP, JMS or XML?

   - If not, either the ESB infrastructure or adaptors will need the capability to transform between the required open standards and the formats supported by the service providers.

7. Where access to legacy systems is required using more recent XML-based technologies (including SOAP, but also basic XML with Enterprise Application Integration middleware), is direct support (e.g. CICS SOAP support) available, or is a separate adaptor required? Does the legacy platform support XML processing, and is such processing a sensible use of the platform capabilities?

   - If for any of these reasons a required SOAP or XML capability will not be made available on a legacy platform, appropriate transformation capability will be required either in adaptors (such as JCA or WebSphere Business Integration Adaptors), in an integration tier, or in the ESB infrastructure.

8. If an EAI technology is already available, does it implement "services" as message flows with appropriate function and interface granularity, or can it be made to do so? What connectivity protocols does it support (e.g. JCA, SOAP, WebSphere MQ, RMI)?

- If existing message flows do not provide the required services, then additional flows will be needed to perform transformations. If the EAI technology does not directly support the required standards, a gateway component can be added.

9. What measure of protection should be afforded to the service provider systems from service client channels in the form of workload buffering, security, logging etc.?

   - Such buffering will often be a role of the ESB infrastructure, and define some of the capabilities it requires. If specific service provider systems (e.g. legacy transactional systems) have additional needs for protection, a dedicated integration tier could be used.

10. *How many* services should be enabled? What aspects of enablement should be consistent across the services, and how can consistency be enforced, perhaps across multiple platforms and applications?

    - If very few services are involved, a simple point-to-point integration model may be appropriate. However, if more are involved, or likely to become so over time, the addition of a control point such as that provided by an ESB becomes increasingly beneficial.

11. Are the service interactions contained within the organisation or are some external?

    - If external access is required, a gateway component can be used to provide additional control. This is often the case in addition to an ESB infrastructure implemented within a single organisation, as the requirements for security and service routing etc. may differ for services made available externally.

12. Are there requirements for service choreography, and do they involve short-lived or long-lived (i.e. stateful) processes, or both? Do they include manual activities?

    - Where these requirements constitute *business function*, the choreography should be implemented in a Service Choreographer component separate from the ESB. Requirements to support long-lived stateful processes or manual activities will place constraints on the choice of implementation technology.

13. What service level requirements should the infrastructure support, e.g. service response time, throughput, availability etc., and how is it required to scale over time?

    - Some of the candidate technologies for ESB implementation are relatively new and may only have been tested against limited service levels. Similarly, because the relevant open standards are either recent or emerging, support for them in more established products and technologies is also new.

    - For the foreseeable future, critical architectural decisions will be concerned with balancing the benefits of specific open standards supported by emerging or mature product technologies against service level requirements. These point-in-time decisions will need to recognise that some standards, and product support for them, are relatively mature (e.g. XML, SOAP etc.), others (e.g. WS-Security) are newer whilst others (e.g. WS-Transaction) are still emerging.

- The trade off between the benefits of standards and proven service level characteristics will often drive a mixed approach combining standards-compliant and proprietary or bespoke technologies in an ESB and SOA architecture.

14. Is a point to point or end-to-end security model required (e.g. should the ESB simply authorise service requests, or should it pass the requestor identity or other credentials through to the service provider)? Is there a need to integrate the service security model with application or legacy security systems?

    - If point to point security is acceptable, a number of existing solutions (e.g. SSL, J2EE security for database access, adaptor security models etc.) can be applied. If end to end security is required, the WS-Security standard is a possibility, providing it is supported by all the systems involved. Alternatively a bespoke model using custom message headers or passing security information as application data could be used.

3 Solution Patterns

Each of the following sections describes a “solution pattern” for one style of Enterprise Service Bus (except the Basic Adaptors pattern, which represents a simpler, point-to-point solution). Each pattern suggests various implementation options using current technologies, along with pros, cons and migration considerations.

Note that the diagrams in each solution pattern depict “Service Clients” as being separate from the systems that provide services; of course, in many situations the same systems or applications may be both clients and providers of services. The diagrams are not intended to rule out this possibility by separating clients and providers, but do recognise that there are two different *roles* that may be played by the same system in different interactions. This distinction is often important in determining the way a system selects, identifies and invokes services in its role as a client, and receives, handles and responds to service requests in its role as a provider.

The solution patterns characterised in this section are:

- Basic Adaptors
- Service Gateway
- Web Services Compliant Broker
- Enterprise Application Integration Infrastructure for Service Oriented Architecture
- Service Choreographer
- Full Service Oriented Architecture Infrastructure

3.1 Basic Adaptors

3.1.1 Description

This solution option represents simple point-to-point service integration using wrapper or adaptor technology, rather than a true Enterprise Service Bus. Such technology might enable integration through WSDL-defined SOAP access, or other interoperable technologies such as WebSphere MQ. In the case of technologies which do not provide a native model for service interface definition, such as WSDL, a bespoke model will be needed to fulfil the principles of Service Oriented Architecture.

Whilst simple, the benefits that can be obtained through this pattern should not be underestimated. For example, “direct” integration through WebSphere MQ or SOAP/HTTP can still be relatively loosely coupled, particularly if aspects of the interaction are declared using interfaces. At some point in the future, the integration could be “interrupted” by an ESB infrastructure that supports the integration technologies initially used. It is also possible to exert some level of control over service naming and addressing at a process level.

A wide range of adapters are available or can be created via development tooling or runtime technology. Support can be provided for Web Services standards and Enterprise Application Integration middleware, and for a variety of systems, including modern distributed application servers (i.e. J2EE Servers such as WebSphere, or .Net), legacy applications (e.g. CICS), and Commercial Off-the-Shelf (COTS) software packages (such as SAP or Siebel).

Figure 4 below illustrates the generic Basic Adaptors solution, including the use of existing HTTP and Enterprise Application Integration middleware infrastructure to support new integrations. Whilst the figure depicts an internal integration scenario, it could also apply to external scenarios, providing HTTP is used as the communication protocol, or some form of internet-compatible EAI technology is available, such as WebSphere MQ Internet pass-thru.

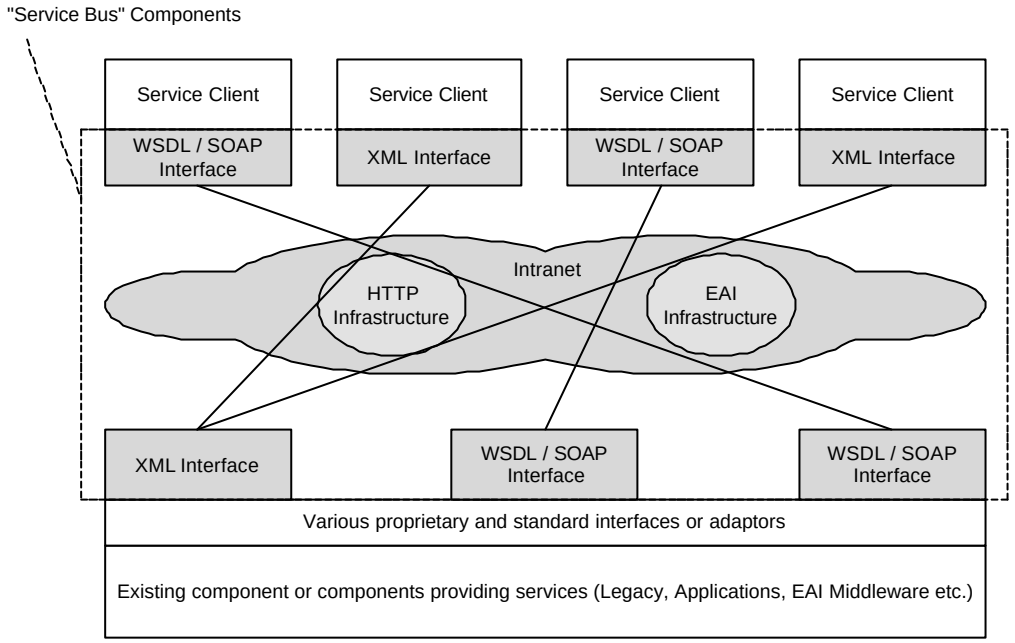

Figure 4: Basic Adaptor solution pattern depicting existing HTTP and unmodified EAI infrastructures as supporting some aspects of service bus capability.

3.1.2 Implementation Technology Options

- Use SOAP or EAI capability directly provided by legacy systems or applications. For example, IBM CICS now provides direct SOAP support, and many systems and application packages can support WebSphere MQ or SOAP interfaces.
- If the applications you wish to provide access to are bespoke applications running in an application server environment, either the runtime or application development tooling for the application server can be used to add wrappers to the application. WebSphere Studio Application Developer can be used to add XML, SOAP or WebSphere MQ support to J2EE applications deployed in WebSphere Application Server.
- Where such support is not available or appropriate (e.g. if XML transformation is not an appropriate use of processing resources on the existing platform), an additional architecture layer may be required, as shown in **Figure 5** below. This might be an application server layer hosting adaptors integrated with application or legacy systems. For example, WSAD-IE provides Java 2 Connector Architecture connector tooling to access legacy systems such as CICS and provide both J2EE and Web Services interfaces to them through a WebSphere runtime environment.

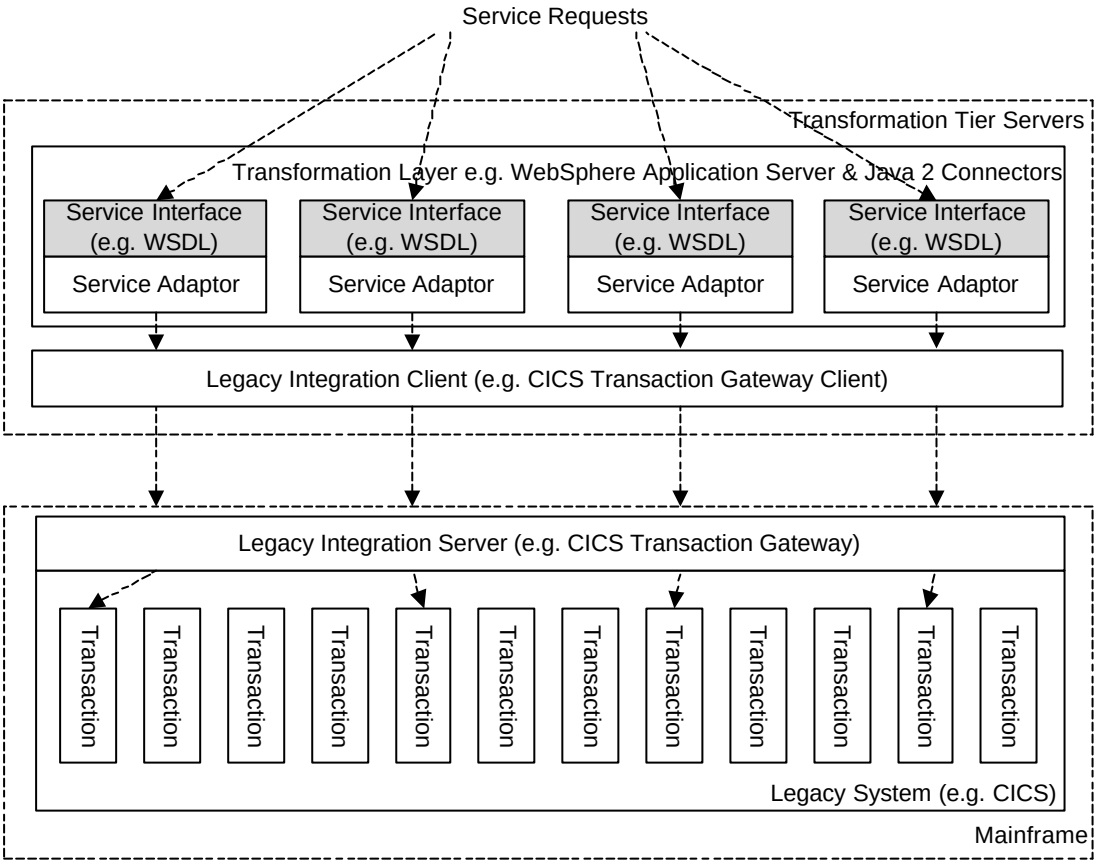

Figure 5: Additional Architecture Layer to Perform XML Transformation Processing

- Where development tooling is used to create wrappers it is possible to augment the function provided by the tooling, for example by creating a framework or set of utility classes to perform common tasks, such as security, logging etc. However, this approach can lead to "scope creep" and result in the framework becoming a de facto bespoke Service Gateway or Web Services Compliant Broker. Care is required when defining the capabilities of a proposed framework to verify that the benefits justify the development and maintenance cost, and that it would not be more appropriate to switch to a more sophisticated pattern.

More detailed implementation advice for this pattern can be found in reference 10 in the References.

3.1.3 Implications

On the positive side, this solution pattern requires no or minimal new infrastructure, and employs basic, widely supported standards and technologies. On the negative side, any security, management etc. capabilities are left to the applications or perhaps the implementation of individual wrappers.

Migration to a more sophisticated architecture should be relatively straightforward as this pattern is based on the use of interoperable technologies and open standards.

3.1.4 Alternative Patterns

Where integration requirements cannot be met by any of the options above, or where some additional capability or quality of service requirement exists, a "wrapper" approach might be insufficient. In this case a Service Gateway is the logical next

step. If more sophisticated ESB capabilities are required, then either the Web Services Compliant Broker or Enterprise Application Integration Infrastructure for Service Oriented Architecture patterns could be suitable.

3.2 Service Gateway

3.2.1 Description

This pattern represents a basic ESB implementation, close to The Minimum Capability Enterprise Service Bus Implementation described above. Service Gateways often support client connectivity through SOAP/HTTP, WebSphere MQ, JMS etc., but may support broader integration to service providers, e.g. through the Java 2 Connector Architecture or WebSphere Business Integration Adaptors. The gateway component acts as a central control point for service routing, protocol transformation and security.

A gateway can be used to provide clients with a consistent service namespace (e.g. in the form of URLs for SOAP/HTTP services) and authorisation model to services that are in fact provided by disparate systems through multiple protocols. This is obviously a requirement where there is a need to expose services to external partners such as clients or suppliers, but may also be useful within a single enterprise where there is a desire to simplify access from applications to functions implemented in a variety of systems and technologies.

A key gateway capability is the transformation of service protocols supported by clients to service protocols supported by providers. Protocols might include SOAP/HTTP, WebSphere MQ or SOAP/JMS, Java 2 Connector Architecture, RMI/IIOP etc. The capabilities of candidate implementation technologies will need to be assessed against the required client and provider protocols.

Figure 6 below depicts the Service Gateway solution pattern.

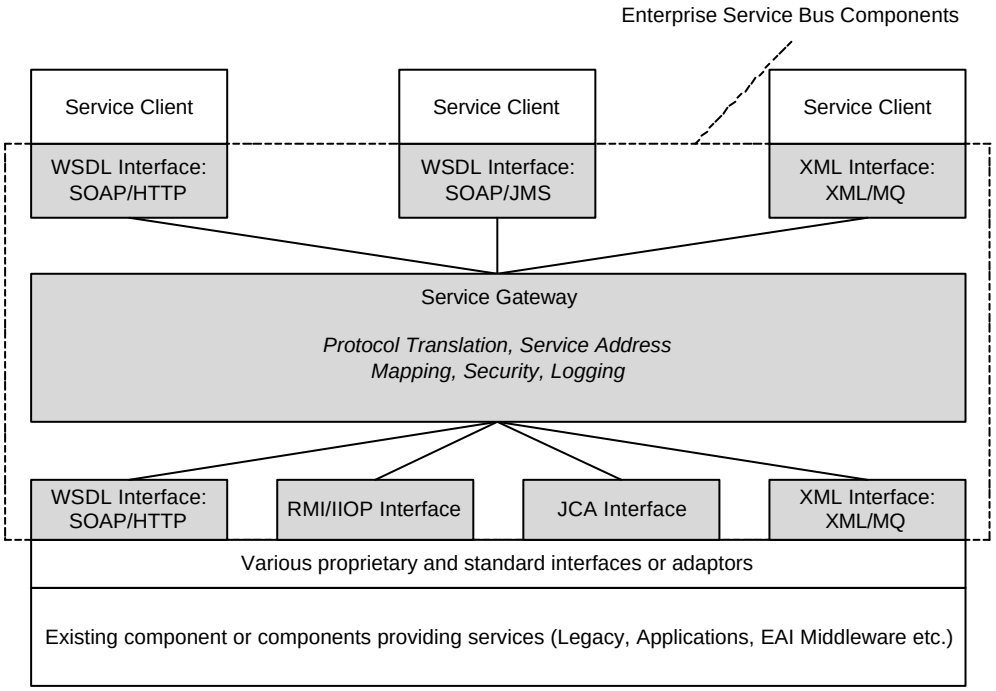

Figure 6: Implementation of an Enterprise Service Bus using a Service Gateway Pattern

3.2.2 Implementation Technology Options

- Use packaged Gateway technology such as the Web Services Gateway provided with WebSphere Application Server Network Deployment or WebSphere Business Integration Connection. Many gateway technologies support some form of “intermediary”, “filter” or “handler” programming model to enable bespoke enhancements to function. The Web Services Gateway provides some configurable intermediary function, and also supports the use of Web Services request and response handlers as defined in the JAX-RPC specification.
- Use the application development and runtime capabilities of a modern application server technology, such as the WebSphere Application Server, to implement a bespoke gateway. This might involve the same type of adaptors as described in the Basic Adaptors solution pattern above.
- If more sophisticated function is required consider more sophisticated Enterprise Application Integration middleware, such as WebSphere Business Integration Message Broker.
- A number of implementations of this pattern exist in legacy technology, usually without the use of Web Service technologies. Many organisations have, for example, constructed “router” transactions that offer a simple interface using a text-like data model to multiple legacy transactions. Such systems are effectively implementing the “gateway” pattern, using a bespoke data format with some of the portability benefits of XML.

3.2.3 Implications

On the positive side, this solution can involve minimal infrastructure, although some gateway technology must be deployed in an appropriately resilient manner. The emphasis on interoperable protocols and open standards also simplifies infrastructure concerns. The ability of most gateway technology to interact with a number of other interface types, such as RMI/IIOP and JCA, should minimise the deployment of additional connectivity technology.

However, gateway technologies will often limit service processing to simple one to one mapping of request / response and publish / subscribe services – any more sophisticated function such as message transformation, message correlation, message aggregation etc. might lie outside the capabilities of appropriate technology, or require inappropriate development effort in a bespoke scenario.

Most Enterprise Service Bus technologies recognise the gateway pattern and its associated capabilities; given this, the use of interoperable protocols and open standards, and the simplicity of gateway function, any migration issues to a more sophisticated ESB infrastructure should be kept reasonably low.

3.2.4 Alternative Patterns

The most obvious alternative patterns are Web Services Compliant Broker or Enterprise Application Integration Infrastructure for Service Oriented Architecture, suitable when the requirements indicate rather more capability than would comfortably be associated with a “Gateway”, or than is provided by packaged Gateway technologies. On the other hand, if very few services are in fact involved, a simple Basic Adaptors solution might be appropriate.

3.3 Web Services Compliant Broker

3.3.1 Description

This solution pattern represents a sophisticated enterprise service bus implementation, providing all the capabilities of a fully fledged EAI solution, and using an open standards model. The precise requirements of a specific situation will define what level of EAI capability is required, and hence which EAI technologies are appropriate. **Figure 8** below shows the implementation of an ESB using a Web Services compliant broker.

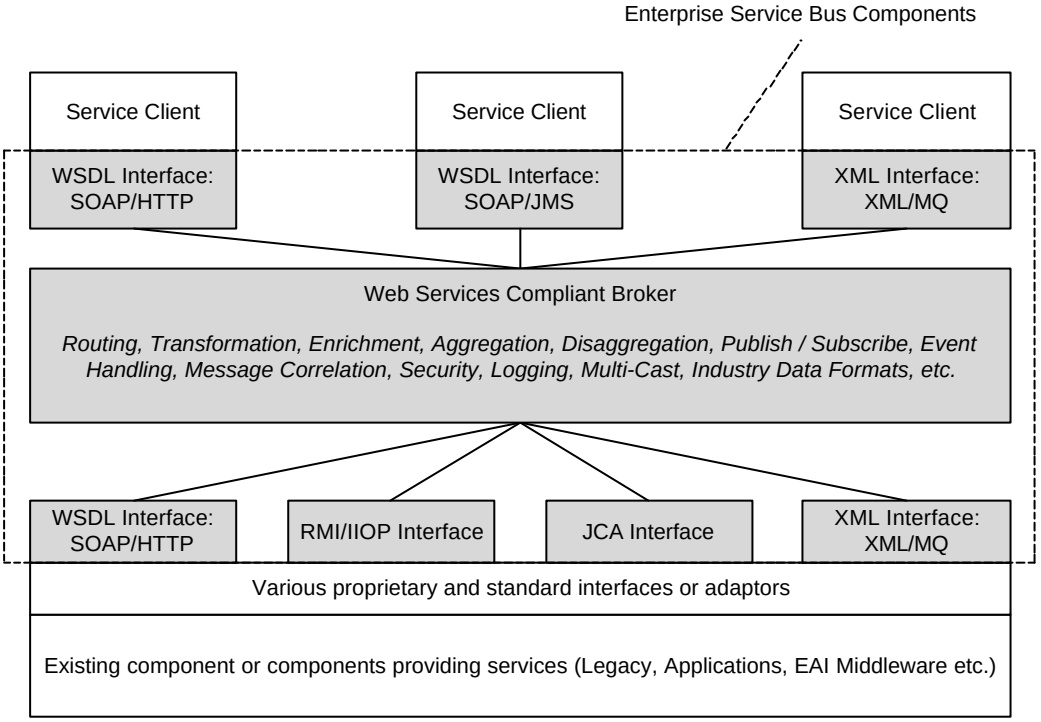

Figure 8: Implementation of a Rich-Featured Enterprise Service Bus Using a Web Services Compliant Broker

3.3.2 Implementation Technology Options

- The most likely implementation technology for this solution is Enterprise Application Integration middleware, such as WebSphere Business Integration Server, providing appropriate Web Services support.
- Optionally, where Web Services support is required primarily for external integration, the proprietary features of the EAI middleware can be used internally, combined with the use of a Service Gateway component to add Web Services support.

More detailed implementation advice for this pattern can be found in reference 9 in the References.

3.3.3 Implications

The advantages of this implementation are the provision of rich functionality within an open standard model. However, whilst EAI middleware is mature, its support for open standards, particularly the more advanced Web Services standards such as WS-

Policy and WS-Transaction, might not yet be so mature. So, the primary disadvantage of this scenario is that it simply may not be viable in all situations.

3.3.4 Alternative Patterns

If appropriate Web Services support cannot be provided, the requirements for a service bus can be fulfilled in a more proprietary or bespoke manner by the Enterprise Application Integration Infrastructure for Service Oriented Architecture pattern, perhaps in combination with a Service Gateway component to at least add Web Services interfaces. Alternatively, if open standards support is the most critical requirement and some of the EAI capabilities such as transformation and aggregation can be accomplished elsewhere, perhaps in applications or adaptors, the Service Gateway pattern might be appropriate.

3.4 Enterprise Application Integration Infrastructure for Service Oriented Architecture

3.4.1 Description

For reasons discussed throughout this paper, it may not always be appropriate to adopt the Web Services standards. However, the principles of Service Oriented Architecture (SOA) can still be applied to construct a solution based around either proprietary or bespoke technology, or alternative open standards.

An obvious approach, proven in many successful implementations, is to use Enterprise Application Integration (EAI) technology, often but not exclusively in combination with XML, to construct a bespoke SOA infrastructure. Providing service interfaces are explicitly defined and of appropriate granularity, EAI middleware can ensure the interoperability and location independence principles of SOA are met.

The potential benefits of this approach are significant, as the full functional and performance power of mature EAI technology is applied to the flexibility of SOA. These benefits apply both to the implementation of new, robust infrastructures for SOA, or to the application of SOA principles to an existing infrastructure.

An Enterprise Service Bus implemented in this way will use and benefit from important open and de facto standards, and may in fact be the means by which widespread introduction of these standards to the existing IT infrastructure takes place, providing a basis for further evolution:

- Many EAI technologies are so widespread, particularly within individual organisations, that they bring the same interoperability benefits as open standards.
- Where appropriate, XML data and message formats can be used to facilitate interoperability and platform independence – just as XML facilitates these benefits in the Web Services standards.
- It is likely that the EAI technology will support some form of Web Services, so open standard interfaces might be provided where appropriate, particularly using the Document / Literal SOAP model to expose any XML formats in use. Alternatively, such access could be provided by addition of a Service Gateway to the solution.
- In some cases, the use of Java as a platform-independent programming language can be used to provide a client API package, and this might be usable

not only from J2EE environments but from standalone Java environments, Database environments that support Java, and various others.

- The EAI middleware might support other open standards, such as Java Messaging Service, which whilst not perhaps quite as broadly applicable as Web Services, are nevertheless supported by multiple technologies.

This approach can represent a significant step towards a fully open standard SOA infrastructure: whilst migration to Web Services standards is likely to be at least a consideration at some point, the interim use of EAI and perhaps XML technologies does at least provide a means to address questions such as interface granularity, common data models and formats, etc., all of which are important steps along the way.

Finally, we should re-emphasise the benefits of this approach: mature EAI technology offers an incredible wealth of Enterprise Service Bus capability (process and data modelling, transformation, content-based routing, service aggregation and choreography etc., etc.) with proven performance, availability and scalability characteristics. Where these capabilities are the most significant requirements, the use of EAI technology to implement an Enterprise Service Bus without Web Services technologies at the core of the solution is entirely appropriate, particularly since there are a number of options for adding Web Services support where required.

Figure 7 below depicts the components involved in this solution pattern.

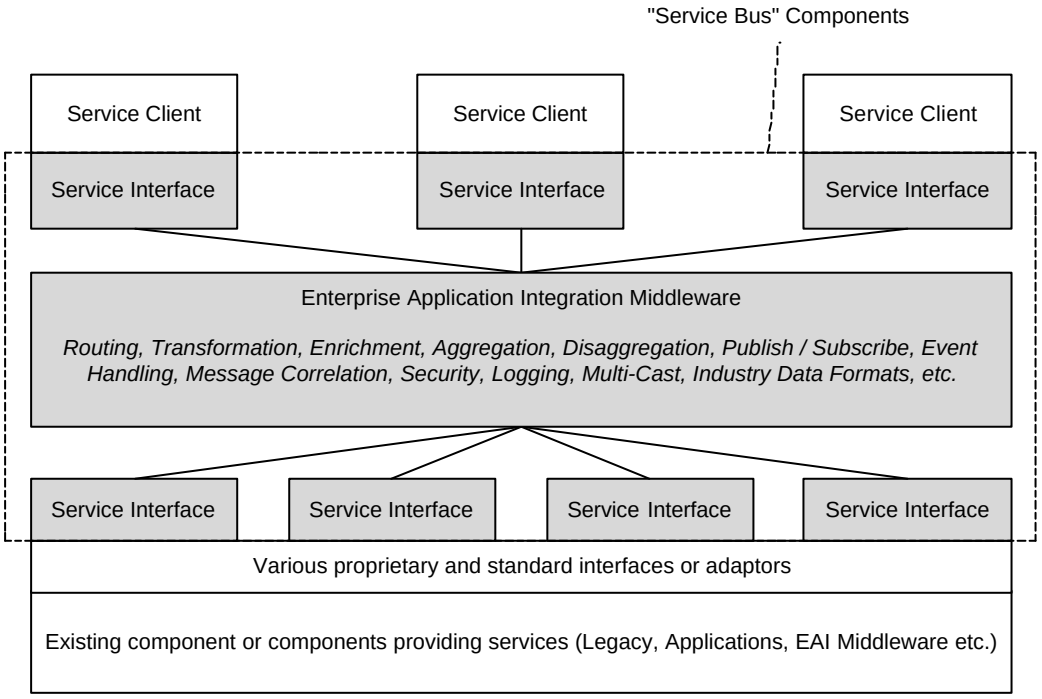

Figure 7: Implementation of a Rich-Featured Service Bus Using Enterprise Application Integration Middleware

3.4.2 Implementation Technology Options

The choice of Enterprise Application Integration middleware will be determined by matching the ESB capabilities required by a specific situation with the features of various EAI products such as the WebSphere Business Integration family.

A key area of design activity is in the service interface definition model: in order to comply with the principles of Service Oriented Architecture, services should be defined using an explicit interface. Whilst some EAI technologies may offer such a model, in other cases a bespoke solution is required. In practise, this is often implemented using an XML schema combining service identification and addressing and business data. However, non-XML solutions are possible, such as the textual solutions used by some legacy implementations of the Service Gateway pattern.

The function of those aspects of the interface model that are *not* related to the data model is to declaratively define how the features of the EAI infrastructure should be used to mediate service requests and responses. Some mechanism is therefore required by which applications can interpret the interface definition and make the appropriate calls to the EAI infrastructure. Again, this may be provided by the EAI technology; alternatives include the enforcement of design and development standards, or the use of framework APIs.

The development and maintenance of a framework API is obviously not trivial, but more effective than enforcing standards across multiple applications. Such an approach is most beneficial where at least a majority of the applications connecting to the service bus support the same programming language, such as Java.

Choices also exist in the adoption of a business data model, whether XML-based, proprietary or bespoke. There are a large number of both general and industry specific XML data models, and there may be some advantage in adopting one of these. However, many are in the process of migrating to the Web Services standards; if this solution pattern is under consideration because the available Web Services technologies are not suitable for some reason, then those standards would not be an option.

Finally, if some form of Web Services or other standards based access is required to services implemented using this bespoke solution, then options exist either to use Web Services support provided by the EAI technology, or to add an explicit Service Gateway component if that provides a better match to the requirements.

3.4.3 Implications

As this solution pattern can represent significant development, implementation and maintenance effort, it demands careful consideration. The benefits are that the solution is entirely consistent with the principles of Service Oriented Architecture, has repeatedly been proven to deliver business benefit, and can be implemented in mature technologies with enterprise-class function, resilience and performance.

The costs lie primarily in two areas: firstly, in the initial implementation and ongoing maintenance of the solution, and secondly in the migration effort that is eventually likely to be required to adopt an open standards solution as Web Services technologies mature, and their use becomes increasingly compelling.

Adoption of this pattern is a point-in-time decision depending on whether near or medium term advantages justify the necessary investment. The investment required can depend on the existing level of use of EAI, and on the extent of any additional bespoke development. The definition of “near or medium” term depends on when an individual organisation believes that emerging Web Services standards will be sufficiently mature to meet its functional and non-functional requirements.

3.4.4 Alternative Patterns

The Web Services Compliant Broker pattern represents a similar implementation using open standards technology.

3.5 Service Choreographer

3.5.1 Description

This pattern consists of the implementation of a dedicated service choreography component. Such a component is not really an enterprise service bus but will support connectivity to services through various protocols such as SOAP/HTTP or WebSphere MQ that either require or imply the presence of an ESB. In some scenarios, such support might be sufficient to allow direct connectivity to service providers and server requestors; where that is not the case, an ESB could be provided through any of the other solution patterns described in this paper – this would constitute the Full Service Oriented Architecture Infrastructure solution pattern.

Figure 9 below depicts the implementation of a service choreographer.

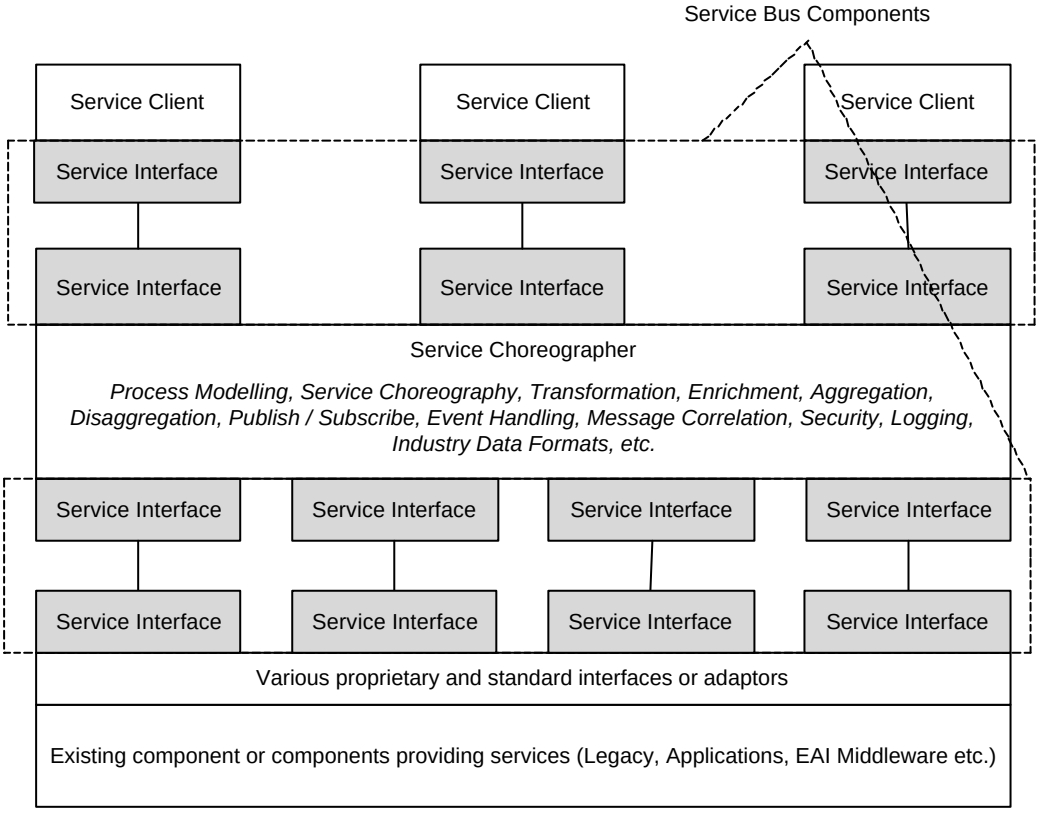

Figure 9: Implementation of a Service Choreographer

3.5.2 Implementation Technology Options

The most important choice to make in this solution pattern is the degree to which open standard support is required; roughly three scenarios exist:

- The wholesale adoption of Web Services standards for service interfaces and process modelling.

- The adoption of Web Services standards for service interfaces combined with the use of proprietary process modelling technology.
- The use of both proprietary interfaces and proprietary process modelling technology.

These questions are particularly relevant to this solution pattern as the Web Services standards relating to process modelling (primarily BPEL4WS) are amongst the most recent, and hence amongst those for which product support is least mature. Most vendors of service choreography technology will offer a mixture of proprietary and standards based technology, for example:

- WebSphere Enterprise Process Choreographer technology, which provides support for Web Services interfaces and process definitions.
- WebSphere MQ Workflow provides support for more mature but more proprietary service choreography technology, with either Web Services or proprietary interfaces.

If a proprietary technology is adopted, perhaps in order to address scalability or resilience requirements, a Service Gateway component could be added to provide Web Services connectivity. If the service choreography technology chosen cannot provide sufficient integration with service providers (e.g. legacy systems or application servers), then an ESB will be required following one of the other solution patterns.

3.5.3 Implications

The implications of this solution pattern depend largely on whether a standards-based or proprietary solution is implemented. Standards-based solutions are currently less mature but will eventually offer better interoperability. Proprietary solutions will likely offer scalability and resilience in better known models, and may well utilise highly interoperable communication technologies such as WebSphere MQ; however as open standard technologies mature and become pervasive, some migration effort may eventually be required.

3.6 Full Service Oriented Architecture Infrastructure

3.6.1 Description

This pattern represents the combination of a Service Choreographer component with a service bus implementation. As both these aspects are described in detail elsewhere in this paper, they will not be described further here, except to say that an entire spectrum of implementation is obviously possible, from an entirely proprietary solution using perhaps the Enterprise Application Integration Infrastructure for Service Oriented Architecture pattern for the enterprise service bus and a proprietary Service Choreographer technology, through to an entirely open-standard solution using the Web Services Compliant Broker pattern for the enterprise service bus and an open-standards compliant Service Choreographer technology.

4 A Roadmap for SOA and ESB Adoption

The patterns described in this paper are concerned with constructing an Enterprise Service Bus for immediate needs, and are likely to be simply the first step towards a more comprehensive Service Oriented Architecture implementation. This section discusses some of the options available to an organisation considering how to evolve in a controlled and incremental fashion. We do not propose that there is a single roadmap for all organisations; rather, we intend to discuss some of the issues that should be considered in the construction of an SOA or ESB roadmap.

4.1 Identifying the Immediate Scope of Concern

In a simplistic view, there are two major aspects to implementing a comprehensive Service Oriented Architecture: the implementation of a fully functional, resilient infrastructure, and the exposure of all relevant function across the business as services. Whilst not entirely independent, there is a degree of decoupling between these two aspects, allowing organisations some flexibility in how they choose to approach them.

In some ways, then, the first decision to take is: whether to prove an ESB technology, or to prove the SOA principles of functional architecture? This question leads to two extremes of approach:

- **Rich Infrastructure, Pilot Function** – here, the primary concern is proving technology capability. The infrastructure is likely to include sophisticated ESB capabilities (whether open-standard or proprietary), and perhaps constitute the Full Service Oriented Architecture Infrastructure solution pattern. However, such a technical solution incorporates a high degree of risk and may incorporate relatively immature technologies, so its implementation will not be linked to business critical projects or functions. The functions exposed as services may either be of low criticality, or remain delivered primarily through alternative channels. As the infrastructure capability is proven and matures, services will be migrated to it over time.
- **Basic Infrastructure, Rich Function** – here, the primary concern is the exposure of business function as services so that they can be accessed or combined in new ways to deliver business value. In this case the forecasted business benefit or other factors driving change are usually significant enough that a lower level of technical risk is mandated. Consequently, the infrastructure is implemented either using only the most basic and mature Web Services standards, or in more established EAI technologies. Once the infrastructure is in place and supporting service interactions, its capabilities can be upgraded or migrated over time as ESB technologies mature.

Of course, there are two other extreme cases of approach – to do nothing, or to do everything at once, but these are perhaps less interesting from the point of view of a roadmap!

Another approach is adoption “by stealth”: i.e., the incremental adoption of SOA and ESB principles, technologies and infrastructure by individual departments, projects or applications. Many organisations may in this way have progressed further with ESB or SOA adoption than might be immediately apparent. This “local” adoption of specific technologies or practises may often provide a more successful proof of them than a “big bang” approach. This is similar to the “rich infrastructure, pilot function” approach, but consists of many basic infrastructures rather than a single rich one.

There are two other aspects of an approach to SOA should be established early on: the provision of internal or external access to services, and an approach to service granularity.

The decision to support internal or external access will drive several factors, including what level of service security is required (See Security Issues Affecting the Enterprise Service Bus above), and whether an explicit Service Gateway component is required to control external access. External access may also drive the use of Web Services standards, whereas for internal access rather more flexibility is possible, such as WebSphere MQ, RMI/IIOP or proprietary XML, as discussed in the Enterprise Application Integration Infrastructure for Service Oriented Architecture solution pattern.

The issue of service granularity has been discussed widely in the industry (see, for example, References 2 and 11). A comprehensive SOA is likely to contain various granularities of service, perhaps from "technical functions" such as logging, billing etc., through "business functions" such as "query account balance" through to business processes such as "process stock order".

Services at each level of granularity will be composed from services, or other functions, of lower levels of granularity. There are hence several levels of service aggregation or choreography to consider, which may well be suited to more than one implementation technology. A practical way to deal with this issue in any specific situation is to identify, characterise and *name* the levels of service granularity which are applicable. Specific aggregation or choreography requirements between different granularity levels can then be defined, and appropriate implementation technologies selected.

Finally in this context, it is worth noting the relationship between the "top-down" and "bottom-up" models of service enablement. The "bottom-up" approach is focussed on enabling the functions of applications and legacy systems as services. In the general case, this involves the use of adaptors and development tooling to provide appropriate interfaces, and results in the enablement of relatively fine-grained "business functions".

The "top-down" approach is more concerned with the architectural process of analysing business systems and components to identify processes and services. This tends first to identify rather large-grained services which are likely to be composed from more granular services.

It is likely that many organisations will employ both these approaches to service identification and enablement: some sort of "meeting in the middle" is then required to combine them. That meeting is likely to be more straightforward to engineer if it is undertaken whilst explicitly recognising and categorising the various granularity levels as suggested here.

4.2 SOA Milestones

Whichever broad approach is taken to full SOA implementation, there are a number of milestones that will need to be passed along the way. This section identifies and discusses some of these milestones. The milestones are not presented in order, as they are largely independent, and the order in which they are achieved depends on many factors affecting individual organisations.

- **Standards Based Security Model -** Whilst simplified or proprietary models may be sufficient in the short term, a fully featured and open standard security model will be essential to a comprehensive SOA. Understanding the point at

which product support for the Web Services security standards meets an organisation's requirements should be a key component of the overall implementation plan.

- **Service Enable Legacy Systems and Applications** – In the same way that the evolution of modern application servers (e.g. J2EE servers such as WebSphere Application Server) led organisations to enable SQL and JDBC or ODBC interfaces to their databases, the evolution of Service Oriented Architecture will drive organisations to enable service-based access to their legacy transactions and application functions. Organisations should therefore plan to define and implement the most appropriate form of service enablement for each system – options include leveraging native XML or Web Services support, the use of adaptors such as JCA adaptors, or the use of EAI or Gateway technology providing legacy connectivity.
- **Implement a High Quality of Service Infrastructure** – By far the most mature Web Services support available is for an unreliable communication protocol, i.e. SOAP/HTTP; standards offering higher qualities of services, such as WS-ReliableMessaging or WS-Transaction, are not yet widely supported. Providing higher qualities of service for SOA currently requires the use of EAI technology. Longer term, organisations should keep abreast of support for emerging standards, and the convergence of EAI technologies with Web Services standards.
- **Identified service granularity levels** – as noted in Identifying the Immediate Scope of Concern above, it is critical to identify the levels of granularity relevant to an SOA, and the requirements for aggregation and choreography between them. Implementation of each level of granularity (e.g. technical function, business function, business process etc.) and the associated choreography should also form a key milestone.

4.3 Steps to SOA Implementation

Following the discussions in the preceding two sections, we are now in a position to compose a general roadmap for SOA and Service Bus implementation:

1. Decide which elements of *SOA technology* or *SOA function* are your priorities to implement (see Identifying the Immediate Scope of Concern).
2. Identify or define a suitable project to implement a first solution, either a technical pilot, a business pilot or perhaps a real business project with an acceptable risk profile.
3. Identify which one of the ESB Scenarios in Service Oriented Architecture is applicable to the project. Further analyse the requirements relative to the Issues Driving ESB Architecture and Design Decisions and A Capability Model for the Enterprise Service Bus. Select one of the Solution Patterns based on this analysis. Based on further analysis and security and non-functional requirements (see Security Issues Affecting the Enterprise Service Bus) select an appropriate implementation technology.
4. In parallel to this work, begin planning the roadmap to evolve this first implementation towards a fully comprehensive SOA. Depending on the focus of the initial pilot, this might involve various aspects of evolving the technical capability of the infrastructure or enabling additional functional services to take advantage of it. In either case, the roadmap should include the SOA Milestones identified above.

5. Beyond these initial projects, plan to evolve in several directions
    1. Evolve and improve data models and processes across the organisation.
    2. Implement phased service enablement of applications, and bring them into the infrastructure.
    3. Evolve the technical capability of the SOA infrastructure.

5 References

5.1 General References

1. "Web Service Oriented Architecture – The Best Solution to Business Integration", Annrai O'Toole, Cape Clear Software CEO at http://www.capeclear.com/clear_thinking1.shtml
2. "SOA – Save Our Assets", Lawrence Wilkes, CBDI Forum (subscription required) at http://www.cbdiforum.com/report_summary.php3?topic_id=2&report=623&start_rec=0
3. "Patterns: Service Oriented Architecture and Web Services", Mark Endrei et al, IBM Redbook SG246303 at

   http://publib-b.boulder.ibm.com/Redbooks.nsf/RedbookAbstracts/sg246303.html?Open
4. The IBM series of articles "Migrating to a Service Oriented Architecture" by Kishore Channabasavaiah, Kerrie Holley and Edward M. Tuggle Jr. at http://www-106.ibm.com/developerworks/library/ws-migratesoa/ (part one) and http://www-106.ibm.com/developerworks/webservices/library/ws-migratesoa2/ (part two).
5. LooselyCoupled.com has a list of Enterprise Service Bus links at http://www.looselycoupled.com/blog/2003_07_13_lc.htm - 105836683995371084
6. The original Gartner article defining the Enterprise Service Bus requires a subscription, but can be found by searching their site http://www.gartner.com , and is entitled "Predicts 2003: Enterprise Service Buses Emerge", published 9th December 2002 by Roy W. Schulte.
7. IBM Patterns for e-Business website at http://www.ibm.com/developerworks/patterns/
8. "The On Demand Operating Environment" at http://www-3.ibm.com/e-business/doc/content/evolvetech/operating_environment.html
9. "Using Web Services for Business Integration", Geert Van de Putte et al, IBM Redbook SG246583 at

   http://publib-b.boulder.ibm.com/Redbooks.nsf/RedpieceAbstracts/sg246583.html?Open
10. "WebSphere Version 5.1 Application Developer 5.1.1 Web Services Handbook", Ueli Wahli et al, IBM Redbook SG246891-01 at

    http://publib-b.boulder.ibm.com/Redbooks.nsf/9445fa5b416f6e32852569ae006bb65f/d336dbf7a0ae01c385256d5000578477?OpenDocument
11. "Coarse-Grained Interfaces Enable Service Composition in SOA", Jeff Hanson, Builder.com at

    http://builder.com.com/5100-6386-5064520.html

5.2 Web Services Standards References

1. SOAP:

   http://www.w3.org/TR/soap/

2. WSDL:

   http://www.w3.org/TR/wsdl

3. UDDI:

   http://www.uddi.org/specification.html

4. WS-Security:

   http://www-106.ibm.com/developerworks/webservices/library/ws-secmap/

5. BPEL4WS:

   http://www-106.ibm.com/developerworks/library/ws-bpel/

6. WS-Transaction:

   http://www-106.ibm.com/developerworks/webservices/library/ws-transpec/

7. WS-ReliableMessaging:

   http://www-106.ibm.com/developerworks/library/ws-rm/

5.3 IBM SOA References

1. Enterprise Integration Solutions, IBM's Service Oriented Architecture team

   http://www.ibm.com/webservices/eis

2. WebSphere Application Server:

   http://www.ibm.com/software/websphere/appserv

3. WebSphere Studio:
   http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/studio

4. WebSphere Business Integration (homepage):
   http://www.ibm.com/software/info1/websphere/index.jsp?tab=products/businessint

5. WebSphere Business Integration Message Broker:
   http://www.ibm.com/software/integration/wbimessagebroker/

6. WebSphere MQ Workflow:

   http://www-306.ibm.com/software/integration/wmqwf/

7. WebSphere MQ Internal pass-thru, support pac MS81:

   http://www-306.ibm.com/software/integration/support/supportpacs/product.html

8. WebSphere Business Integration Connect

   http://www-306.ibm.com/software/integration/wbiconnect/

9. WebSphere Web Services Gateway Introductory article:

   http://www-106.ibm.com/developerworks/webservices/library/ws-gateway/