W H I T E P A P E R

Understanding the ESB

What it is, why it matters, and how to choose one

Contents

Introduction

The Enterprise Service Bus (ESB) is no longer a new technology with 'potential'; it is already delivering return on investment and providing the foundations for the integration architectures of the future. Perhaps because of the ESB's obvious benefits, it already enjoys multiple definitions – so many, in fact, that potential adopters need to approach the technology with caution if they are not to be sold old technology in new clothing.

To this end, this whitepaper attempts to outline the capabilities required in an ESB in order to handle the most complex integration challenges in an open, scalable and flexible manner.

The Need to Integrate

Since the business implications of information technology were first understood, getting diverse systems to 'talk' to each other has been seen as a key obstacle in the way of delivering genuine efficiencies through the use of technology. The scale of investment in getting past this obstacle is reflected in Gartner research estimating that 35 percent of IT activity in a typical enterprise is dedicated to application integration – a figure that includes development, maintenance and operational cost.

Unfortunately, and despite this awareness, in many cases IT solutions have developed in a way that ensures their isolation from other systems, and erects barriers that can be costly and time-consuming to break down.

At the same time the industry has witnessed, and been disappointed by, any number of 'middleware' technologies that "remove all integration difficulties". At this point CIOs and architects are entitled to roll their eyes at whatever the latest market hype is focused upon. But despite this justified cynicism, there remains the understanding that *something* must be done. At the most basic level, there remains a need to integrate.

In order to deliver compelling solutions to customers, or improve operational efficiency, sooner or later an organization is faced with an integration challenge. At some point, information in *System A* must be able to interact with, feed into, or migrate to *System B*. Considering the issues facing organizations today, it becomes apparent that the correct *integration strategy* will separate tomorrow's winners and losers.

As an example, consider service provision in the public sector. The two key drivers in this area are:

- The creation of 'joined up government' whereby multiple agencies and departments are able to work together towards common targets and share information in real-time to improve results.
- The provision of eGovernment services to Citizens, changing the nature of interaction with Government and delivering a single 'one stop shop' for access to multiple services, both in the public and private sector.

Both of these initiatives require some form of integration solution in order to connect and deliver previously isolated information and business logic to citizens, departments, agencies and third parties.

Similarly, in financial services, organizations are under pressure to meet regulatory requirements (which typically involve the aggregation and delivery of data from multiple sources), whilst simultaneously maintaining competitive advantage by improving efficiencies and offering new services to customers. Both these challenges depend on the effective delivery of integration solutions.

Approaches to Integration

Until now, most organizations have had two choices when it came to an integration strategy – adopting one of the 'Enterprise Application Integration' products on the market, or attempting to build their own integration framework, often based on J2EE Application Server products. While either can be successful in certain circumstances, each approach suffers from a (different) significant flaw that makes it inappropriate for solving general integration problems.

1. EAI and 'Integration Latency'

EAI vendors viewed integration as a strategic initiative. With the promise of a single product set, organizations could 'standardize' their integration efforts and deliver a cohesive infrastructure that would potentially integrate the entire enterprise.

However, to be cost-effective this approach requires large-scale projects involving significant investment in skills and processes and the re-engineering of entire organizations. They rely on hub-and-spoke architectures that limit scalability and flexibility. First developed in the days before standard data formats and integration technologies such as XML, JCA and JMS, they provided their own proprietary data formats and adapters. This made EAI products complex, heavy and disruptive products to use and deploy.

For many, EAI became a Frankenstein's monster, lumbering out of control through the organization and delivering ever escalating project sizes and budgets. Even when projects were completed on time and budget (a rare occurrence - according to Gartner, 50% of such IT projects come in over budget), they had often taken so long to implement they were no longer relevant to the business challenges, having been designed to meet these needs when they were last identified – during 'requirements gathering' some time in the dim and distant past. This effect is sometimes referred to as "integration latency", and clearly in these circumstances ROI is hard to come by.

Even aside from these considerations, EAI failed to remove many of the issues that have always been associated with proprietary technologies; lock-in, reliance on expensive skills (or even the vendor themselves) to maintain solutions, and an inability to adapt to changing requirements.

2. Home grown frameworks and the 'Accidental Architecture'

Most organizations correctly view integration as a means to an end, rather than an end in itself. This view has become more common since the 'EAI boom' ended in 2000.

There are good reasons to take such a pragmatic approach to integration. It helps an organization remain focused on resolving actual business issues rather than the technology or 'plumbing' itself. By addressing only the business problem at hand, integration solutions are tightly focused, and as a consequence project timescales are shorter, costs are lower, and ROI is usually clear.

Within the scope of each point problem, custom coding can appear to be the best solution: It leverages existing skills within the organization and requires no additional investment in software products or skills. Development teams already understand the business and are able to start addressing the problem quickly. Familiar tools such as J2EE application servers can be used as the basis of the solution, taking advantage of the wizard-based code generation tools to reduce the amount of manual coding.

However, as this approach grows across multiple projects or departments, the downside becomes apparent. New business and regulatory requirements emerge and evolve all of the time. Each new integration project requires additions to the first point solution. As the complexity of the solution increases, issues such as manageability and performance begin to become a problem.

In some cases, integration products are introduced to address these issues, but these in turn must be integrated with the framework. In the worst case, completely independent solutions are implemented to solve the problem – increasing complexity still further. The end result of this process is the 'accidental architecture' – a complex spaghetti containing a home grown integration framework, duplicate technologies and 'islands of integration'.

This in turn leads to:

- High maintenance costs, due to the large and often complex code base, which also relies on secondary technologies, and places great demands on the scarcest skills in the workforce.
- An inability to adapt and evolve solutions without extensive further integration work.
- Typically, extensive use of 'hand-built' workarounds that in turn are difficult and costly to migrate or extend into new environments.

In other words, over time these accidental architectures become both costly and unwieldy.

Extending the Application Server – A New Alternative?

In recent years Application Server vendors have begun to add integration capabilities to their products. Using the same development environments currently used to build applications, this approach, superficially at least, offers the potential for quick and easy integration. Developers are able to create applications combining components from diverse and remote systems using the GUI tools they are already familiar with.

The reality is somewhat different. These solutions remain, primarily, development tools. Despite claims to the contrary, they remain code-centric, and code must be written or generated to perform common integration tasks, and particularly those associated with the mediation between multiple information models. In effect this means that the maintenance, migration and evolution of such systems are likely to be costly and involve extensive re-engineering.

To many organizations, simply 'extending' their current development effort may be tempting. But in a fundamental sense this approach is no different to the creation of accidental architectures. In just the same way, the end result is complex and inter-dependent integration solutions that are almost impossible to evolve and extend in a cost-effective way.

The Third Way – Incremental Integration

Fortunately, recent developments have made a third way possible - the *incremental* approach to integration. 'Incremental integration' delivers solutions that start small, focusing on specific business problems (as with the home grown approach), but that over time can be adapted and extended to cover the entire organization, as promised by EAI. Crucially, even small initial projects can deliver ROI, whilst the same approach is also ideal for enterprise-wide integration projects.

In this way organizations can maximize the use of existing assets without affecting the pre-existing IT infrastructure, improving the likelihood of delivering targeted projects on-time and under budget, and experiencing genuine ROI in the short term.

As mentioned above, the traditional objection to the incremental approach was the resultant "spaghetti" effect and the inability to develop a cohesive integration vision for the enterprise as a whole. New technologies have changed that reality. XML can be used as a common means of exchanging data between applications and databases. JCA, JMS and JDBC make it easier to plug-in to applications and infrastructure to get at that data. Now it is possible to build technologies that make incremental integration a viable proposition.

A product capable of delivering this type of incremental integration must be usable on small business focused projects – but also scale to the largest projects. To do this it must be

- Easy to learn – leveraging commonly available skills.
- Productive – delivering superior productivity over 'home-grown' approaches for even the simpler point solutions, and enjoying a low initial cost of adoption.
- Small-footprint, but scalable – stripped down for use in simpler projects, with additional capabilities where required for larger projects.
- Distributed – allowing solutions in different business areas to be joined together when required.

'SOA' and the Enterprise Service Bus

XML in itself does not meet all of the sometimes-complex challenges associated with integration in 'real world' environments, but it does provide the basis for this integration. If it is accepted that XML is an alphabet rather than a language, it is clear that genuine integration will require additional technologies to orchestrate business processes, manage transformations within XML and validate and route XML-based messages throughout the organization.

These features are at the heart of the Enterprise Service Bus (ESB).

As with any new product category, there is still some confusion around the exact definition of an ESB. As the industry puts its weight behind the *general* concept of service-oriented architectures as the basis of integration, so the arguments begin concerning the *specific* nature of the ESB itself. This is the usual pattern with any new type of product.

This debate is currently focused on two issues:

- Whether the ESB is an architecture (and one that does not even have to be standards-based), or 'way of doing things', or a product in itself. Whilst it is in the interest of established IT vendors without a product offering in this space to present the ESB as an architecture, the reality is that at present product is required to perform this function. The industry can expect to see more products defined by vendors as ESBs over the next two years.

- The nature of the ESB product; specifically whether an ESB is primarily a smarter message queuing system, which provides simple XML translation, plus routing and messaging capabilities, or whether it will effectively replace EAI functionality by providing application adapters and business process modelling and automation.

Our view is that the ESB must provide a distributed, message-oriented architecture, supporting traditional EAI features such as message routing and transformation, within the context of integration based upon 'business services' and XML messages. It is the technology that adds 'intelligence' to the integration infrastructure, and thus enables the creation of applications that automate business processes and interactions between multiple systems and organizations. As such the ESB can offer support for the type of 'incremental integration' discussed above, and can be seen as a 'disruptive' technology in the marketplace.

The use of the word "service" acts as a reminder that the ESB is often seen as a central (indeed *the* central) component of the service-oriented architecture (SOA). Forrester research, for example, regards the ESB as "*a layer of middleware through which a set of core (reusable) business services are made widely available*". The SOA enables chunks of enterprise functionality to be presented as 'services' to the ESB, which routes, transforms and validates the XML inputs and outputs from these services. By developing in this way, once the Enterprise Service Bus is in place, following an initial point-to-point integration project, further projects merely involve 'socketing' new services onto this backbone or the re-use of existing services.

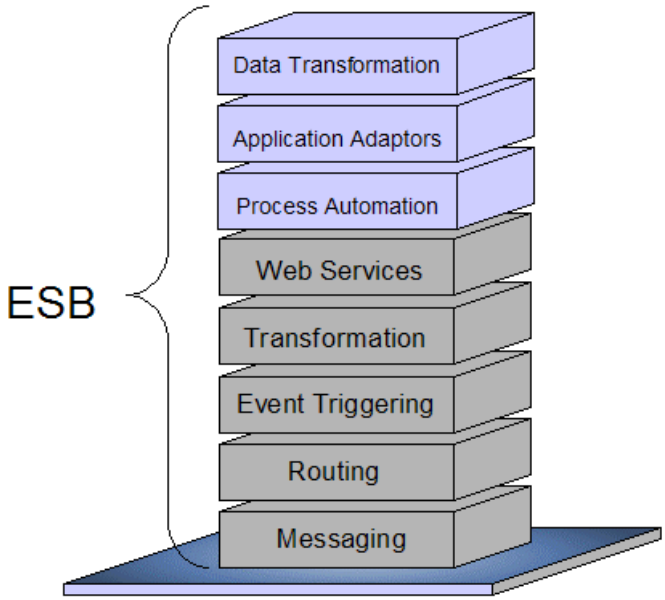

Figure 1 Overview of ESB Functionality.

The end result is the ability to undertake integration projects that can:

- Focus specifically on high-value business issues and deliver rapid ROI
- Offer a low-cost alternative to traditional EAI models
- Extend across the enterprise as more business operations are offered as services
- Re-use these assets by using a 'building block' approach to application development
- Integrate with third parties
- Help avoid reliance on high-cost skills associated with proprietary or custom-built solutions

Accidental architectures and integration latency can be things of the past. Organizations can deliver either tactical or strategic integration projects in confidence, knowing that whilst each delivers ROI on its own merits, they can also evolve to become part of a whole that is more than the sum of its parts.

How to Choose an Enterprise Service Bus

Based on the logic above many organizations are now convinced that the ESB provides a key enabling technology for future business integration. Gartner, for example, has noted that ""A majority of large enterprises will have an enterprise service bus running by 2005". IDC believe that "The ESB…will revolutionize IT and enable flexible and scalable distributed computing for generations to come". Clearly, as a technology it demands serious consideration.

The most complete definition of the ESB has been provided by Steve Craggs, of Saint Consulting, in the white paper "**Best Of Breed ESBs**", a defining document concerning the nature of the ESB and the features they can be expected to provide in order to solve real-world integration problems. According to Saint Consulting, and for the purposes of this whitepaper slightly simplified, these include:

- Basic bus services
- Basic connectivity
- Support for highly distributed environments
- Manageability
- Robustness, Fault avoidance and tolerance
- Scalability and performance
- Security
- Breadth of connectivity

In the remainder of this white paper we look at each feature in turn and suggest what potential ESB users should expect from their technology.

Basic Bus Services

At its heart the ESB acts as an information bus between the various services available within the organization. As such, it must support certain functions in order to perform this task. These include:

- Transformation – the ability to map one data format onto another (usually XML) in order to ensure inter-operability between the various systems plugged into the ESB.

- Routing – whereby the sometimes-complex 'flow' of messages from component to component can be specified.
- Communication – supporting the delivery of messages throughout the organization.

In the case of transformation, mapping between XML formats can be provided via XSLT, but in order to optimize performance and reduce the complexity associated with the mapping process, some ESBs may provide additional mapping functionality. This can improve performance significantly by reducing the processing overhead associated with mapping.

XSLT

Whilst XSLT is useful for simple/non-time critical transformations, more advanced mapping capabilities are also required.

Additionally, these enhanced mapping tools allow complex many-to-many maps to be created without recourse to the script or Java based extensions required in XSLT.

Although life would be simpler if all mapping took place on a one-to-one basis, real world integration problems demand that an ESB is also able to support one-to-many and many-to-one transformations of XML messages and formats. Those ESBs that are best able to model sophisticated and complex business processes in this way are most likely to find favor with customers.

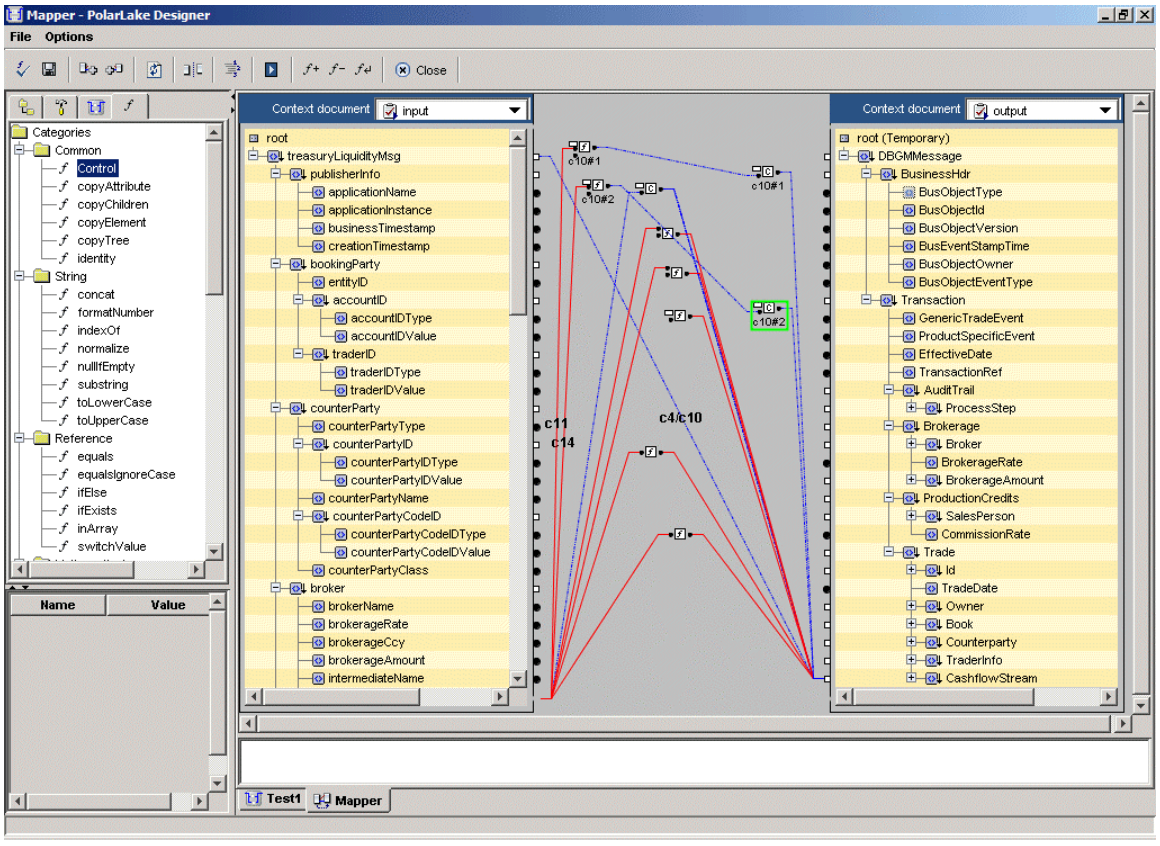

Figure 2 Data Model Mapping in PolarLake Integration Suite

As mentioned above, 'routing' is used to define the flow of messages. However, in many projects this flow must be dynamically altered based on the content of the message, or the results of a previous step in the process. An ESB must provide this ability if it is to be of any use in addressing complex integration challenges.

The communications function of an ESB is a more subtle case. Clearly there must be support for asynchronous messaging, publish and subscribe, store and forward and similar messaging models. However, for many organizations these functions will already be contained within existing messaging products (WebSphere MQ, for example), and it is reasonable to ask whether these functions should be replicated by the ESB, inevitably adding cost and complexity to the finished solution.

Messaging Protocols

ESBs must be able to natively support multiple messaging protocols - as well as Internet protocols such as HTTP.

Furthermore, analysts such as Gartner have recently identified this as the key problem in migrating to an ESB architecture: the need to integrate across multiple ESBs – in the Gartner sense of a smart messaging layer rather than the fully functional Integration Suite that we refer to as an ESB.

This mandates the selection of a 'universal' ESB that can integrate with the messaging solution of choice, providing intelligence on top of that solution rather than replicating its functionality. This is likely to be the most cost-effective and flexible solution. Solutions that include bridges between built-in messaging layers and other messaging systems cannot be considered 'universal' as they still require a new messaging layer to be deployed and often provide limited functionality over the third party system.

Basic Connectivity

Any integration system that does not recognize and accommodate the multiple systems and applications that make up the existing IT infrastructure is almost certain to fail. 'Connectivity' in this context simply means the ability to add components to the 'bus' itself. The object is to support the integration of the diverse applications and environments that make up an IT landscape, and in doing so support seamless communication across the enterprise.

Mini-Hubs

Check that there are no mini-hubs (delivering common services such as itinerary management or transformation).

More specifically, an ESB should be capable of integrating 'web services' developed within any commonly used application server environment. Although this should theoretically be a simple task, slight variance in the way in which services have been implemented can mean it is more complex than might be imagined. ESB 'functionality' that has been added onto existing application development environments should be approached with caution, and

customers must determine that ESBs of this type will enable integration of services regardless of implementation details.

Support for Highly Distributed Environments

In order to support integration across widely distributed deployments, the ESB itself must be a genuinely distributed system when deployed. Specifically, there must be no requirement to route all messages through a central hub in order to apply routing and transformation rules to them. To do this would have severe performance implications for even small deployments.

Distributed Management

ESBs should provide both distributed monitoring and management and plug-in capabilities to existing system management tools.

This can be seen as a central requirement for the ESB. It marks a significant move away from traditional 'hub and spoke' approaches to integration and enables solutions to be implemented in a truly incremental fashion and gradually connected across the enterprise. One trend is to provide micro-hubs to which key services such as transformation are delegated. If these are not distributed, this will cause new bottlenecks as multiple calls to a service will introduce latency.

In order to support this type of deployment, customers should look for light footprint ESB implementations that provide maximum flexibility in deployment and can thus support multiple nodes without difficulty. As a further benefit, light footprint ESBs are also likely to have lower adoption costs than more traditional implementations, helping to deliver productivity and ROI on initial projects.

Manageability

The likely distributed nature of the ESB has obvious implications when it comes to management of deployed systems. Whilst the product is distributed, support is best provided from a single point of control, from which system definitions can be maintained, adapted and then distributed across the full ESB deployment.

When the system is running, it is obviously important to monitor the state and behavior of the system, and to this end some form of ESB management must be provided that is able to monitor and identify potential problems before they become critical issues. Crucially, in B2B or inter-departmental deployments this information must of necessity be available across these boundaries. In order to provide this information to corporate management frameworks, the ESB must be capable of collecting comprehensive system statistics, of both a technical and business nature, and providing notifications relating to these to third-party management applications that can assist in Business Process Monitoring.

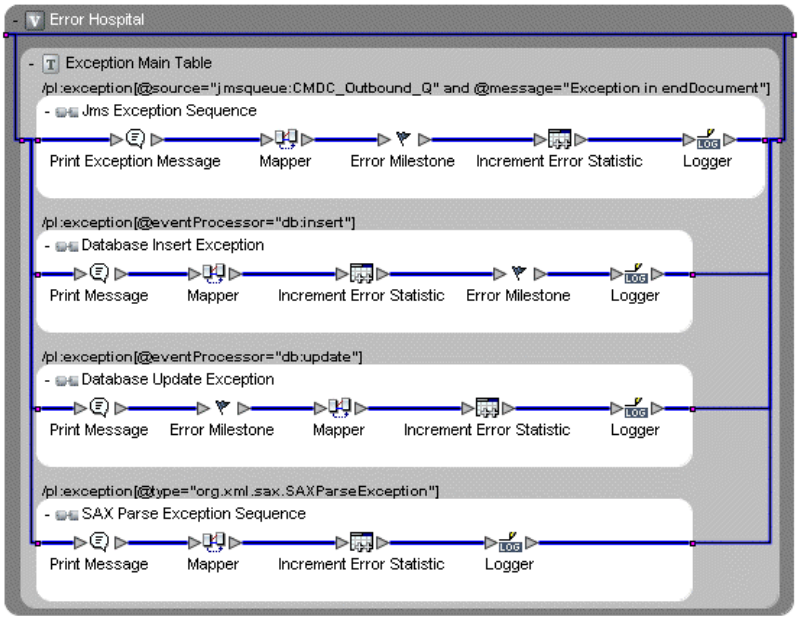

Figure 3 Exception Handling in PolarLake Integration Suite

When problems are identified, it is also necessary to support any actions taken to resolve these, and enable (from a single point of control) the operational activities that might be used to do this. These include starting and stopping specific processes, re-routing operations, and also problem determination functions such as application tracing and message editing. These tools enable the ESB to take a proactive role in ensuring systems do not break down.

Robustness, Fault Avoidance and Tolerance

Give the pervasive nature of the ESB, and its position at the heart of so many operations and business processes, the need for a truly robust solution is a given.

This 'robustness' manifests itself in two ways:

- Fault avoidance – by which the ESB aims to prevent problems occurring.
- Fault tolerance – by which problems, when they do occur, have little impact on levels of service.

Non-standard Extensions

Vendor defined extensions to standards, such as envelopes defining 'itineraries', increase chance of faults as well as vendor lock-in.

Fault Avoidance

One obvious way to reduce the occurrence of problems is to ensure that the ESB software itself is mature and extensively tested. Of course this may be difficult to measure in practice, although the history of the product and the organization behind it may offer some insight. Products that are deployed on a large scale within mission-critical environments may be assumed to be robust in nature. In many cases this can be checked with reference to their customers thereby getting real insight into product quality – useful in a field in which style often over-rides substance.

The adoption and support for industry standards (XML, JMS, Web Services, JDBC) can also assist in fault avoidance. As these standards have themselves been tested and proven they can give some confidence in those products that support them. On that basis customers may be advised to avoid those ESB implementations that rely on proprietary extensions to these common standards – such as vendor-defined envelopes which include product specific routing or control information - for reasons of product quality, never mind the more familiar difficulties that are associated with proprietary technology.

Lastly, ease-of-use can be a factor in fault avoidance. For obvious and well-documented reasons it is desirable that the developers who create a solution are also those that maintain it going forward, or at least have some hand in this function. In order to make this possible, the chosen ESB should be as easy-to-use as possible, requiring little training before existing teams can use it to create new applications.

Fault Tolerance

Of course problems do occasionally occur, which is when 'fault tolerance' becomes an issue. How the ESB responds in this instance will have a significant impact on to what extent these problems impact on running systems and consequently affect the business.

Fault tolerance can be provided in a number of ways. An ESB should support:

- Intelligent routing – ensuring that information is routed around problem areas in the network and thus the target can still be reached and the business process unaffected.
- Exception handling – generating fully configurable exception architectures that are able to catch exceptions, generate compensating transactions, or deliver exception reports.

Exception Handling

Exception handling is key to fault tolerance – the Hurwitz Group estimates that 80% of time spent building business processes is spent in exception handling.

- Redundancy – a common concept, and as important here as elsewhere in the IT infrastructure. ESBs should provide clustering at critical nodes and ideally mirror services in use in case of failure.

- Recovery – when transactions fail, some form of roll-back is required to ensure that they fail on an 'atomic' basis – that is, they are either all or nothing, as in the classic example of the money transfer. In order to do this, the ESB must support some from of transaction management or compensatory transactions in the event of such a failure.

Scalability and Performance

As a key element of the IT infrastructure and a platform upon which multiple solutions will be based, the scalability and performance of an ESB are of critical importance. After the initial adoption of an ESB it is common to 'discover' new requirements and candidates for integration solutions, usually due to the initial success of the first integration projects. Thus adoption can accelerate rapidly across the organization. The ESB that may be initially chosen for a small project must be equally capable of supporting this increased workload.

A number of features and technologies can assist in this area and should be considered when selecting an ESB:

- Support for asynchronous messaging and multi-threading, thus enabling business processes to occur in parallel and consequently with greater efficiency.

- Intelligent load-balancing capabilities to deal with spikes in demand and unusually large document or data volumes within the system.

- Ability to deliver performance benefits through optimized path selection and intelligent mapping capabilities – such as breaking up and recombining documents in order to focus only on those areas requiring transformation.

- Prioritization of services, based on either document type or content-based rules, in order to support business-critical processes when processing resources are limited.

- Support for transparent resource addition – the ability to add new components or modify existing set-ups without disruption to business operations.

High-performance Processing

Poor performance in the processing of XML is the major obstacle to widespread adoption. Any viable ESB must be able to deliver high performance in this area.

Security

Clearly any solution that spans multiple departments and organizations must implement or support security measures that protect the integrity of the information and business processes within the system. The issue of security is a central concern in the IT industry. There are a number of solutions in the marketplace which any ESB should be able to use for the purposes of, for example, user authentication – ensuring that only those with relevant privileges can access the information within the solution, and indeed control the deployment of the ESB itself through system management tools.

There is a further requirement - to address legitimate concerns around the possibility of malicious outsiders either reading information as it passes through the ESB or even altering this information in transit. In order to guard against this ESBs must clearly support some form of encryption technology. They must also be able to specify exactly which components and processes require this functionality as encryption 'across the board' can have significant performance implications. Integrity can also be guaranteed by supporting checksum algorithms that detect any unspecified changes in information routed between components – possibly due to malicious influence.

Breadth of Connectivity

Connectivity is the name of the game for the ESB and as discussed above a basic ability to connect multiple systems is a requirement for any serious product in this area. But it is also worth considering a number of areas in which there may be some difference between leading ESB implementations. Remember that system requirements change on a rapid basis (hence 'integration latency') and that an ESB should be able to connect as many technologies as possible, even if at present they may not be an issue.

Further connectivity issues include those relating to:

- Database integration – both in terms of data and stored procedures. The ability to call stored procedures in particular may be important, as many existing business processes may make use of them.
- Enterprise applications, such as SAP, Siebel, PeopleSoft, Oracle, IMS, JD Edwards etc.
- Enterprise data standards, such as binary data, common text formats, Excel, OAG BOD, SQL, ANSI X12 EDI, UN/EDIFACT EDI, FIX and HIPAA.
- Legacy systems. Although it would be nice to live in a world without existing or 'legacy' systems requiring support, this is unlikely to be the case any time soon. As such, it is imperative that an ESB solution can integrate with existing data and business logic stored within, for example, mainframe environments.

- EAI implementations, which may have been used by the organization in the past and are likely to automate significant numbers of business processes within 'islands of integration'. In order to bring these together, the ESB must support integration with existing EAI solutions as effectively as possible.

- Previous 'standards', such as COM and CORBA, within which previous integration solutions may have been integrated, and from which information and logic may be required to support the integrated organization of the future.

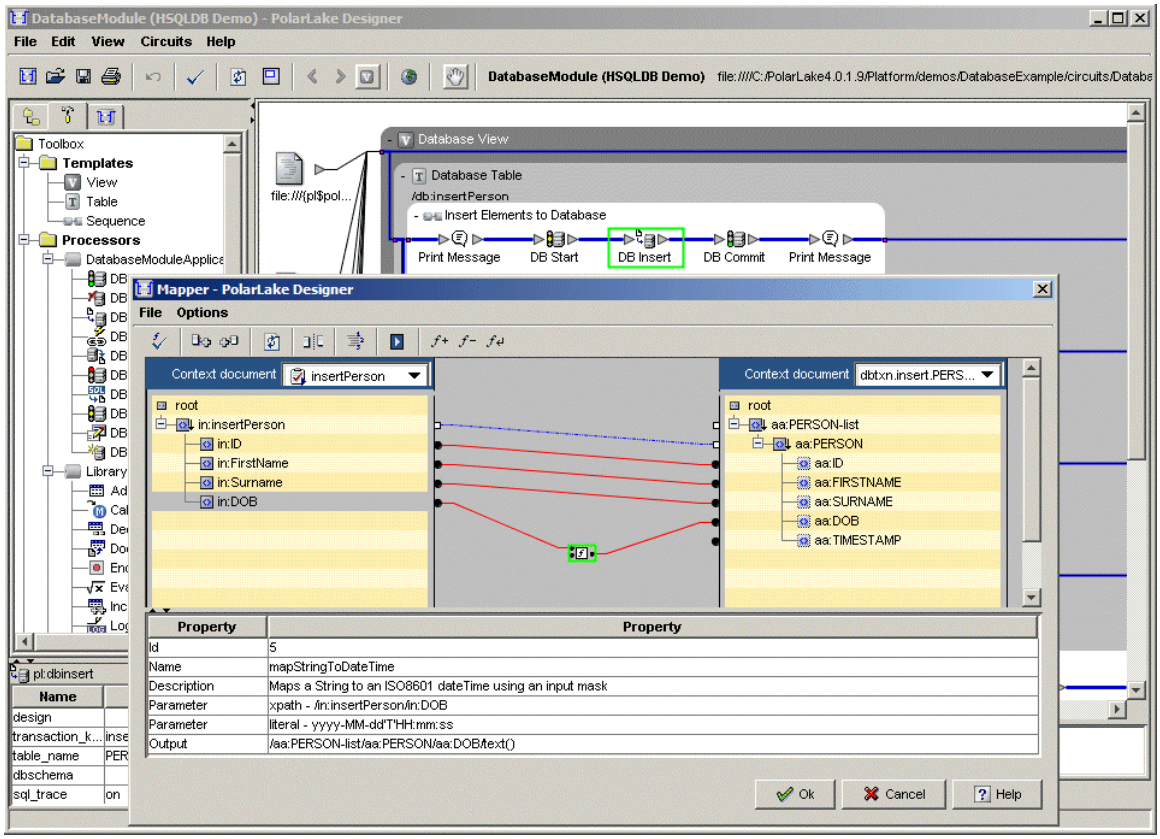

Figure 4 Database Integration in PolarLake Integration Suite

Development / Deployment Toolset

The quality and usability of the toolset provided by an ESB vendor is absolutely vital in terms of reducing the cost of implementation. Whilst more conventional EAI solutions required scarce skills to implement, ESB solutions can deliver significantly increased developer productivity by enabling existing staff, familiar with commonly used standards, to develop and deploy solutions themselves with minimum training. In turn this leads to a significant saving in terms of development and maintenance costs – but this relies on providing easy-to-use interfaces and toolkits that assist in getting started and developing with the product.

Lower Costs

The more an ESB can reduce the number of lines of code required the lower the cost of development and maintenance.

Clearly development and deployment tools must cover the basic requirements around configuration, connectivity, incremental deployment and life-cycle management, but they must also do this in a way that is as easy-to-use and efficient as possible. This may include support for GUI tools enabling drag-and-drop configuration of business processes and transformations, for example. The more an ESB can reduce the number of lines of code required the lower the cost of development and maintenance.

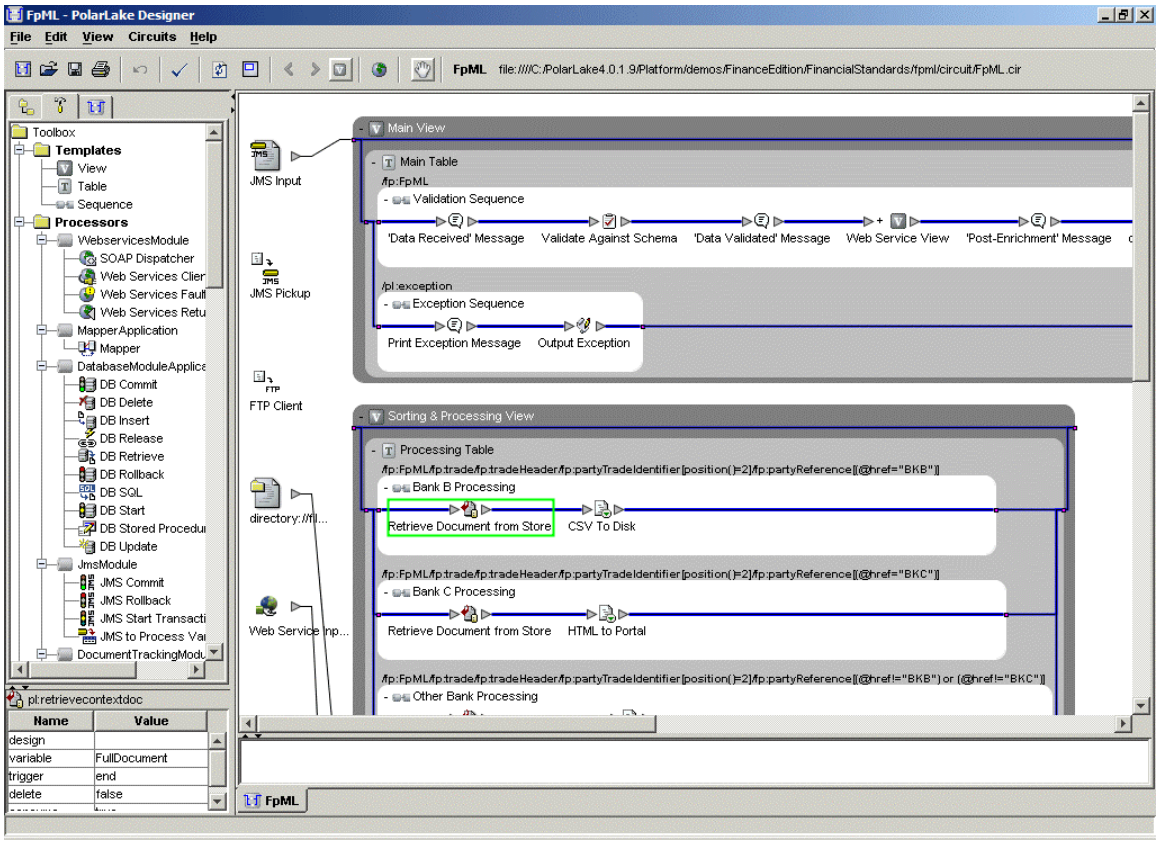

Figure 5 Development Environment in PolarLake Integration Suite

The more such a simple development environment can accommodate multiple technologies and enable a single user to create and manage complex integration solutions, the more efficient they will be in development and the more likely will be their extensive adoption.

PolarLake provides a complete suite of products for implementing integration solutions based on the Enterprise Service Bus. Our full-strength, productive solutions deliver code-free orchestration and mediation of software services, enabling integration solutions to be extended and re-used without extensive re-engineering. The result: real return on investment.

PolarLake has a proven track record in delivering the benefits of incremental integration with a technology that leverages existing IT investments in standards, skills and systems to reduce both initial investment and total cost of ownership. Deployed customers include leading corporations in financial services such as JP Morgan Chase, Pioneer Investments (Ireland), Man Financial Ltd (UK), and Nissay Dowa (Japan), in Government, such as CJIT (Criminal Justice IT, UK), and in telecommunications such as Midwest Wireless (USA) and KDDI (Japan).

PolarLake's solutions are provided by partners such as Hitachi Systems and Services and Sun Microsystems. PolarLake is a private company, headquartered in Dublin, Ireland, with offices in London, New York and Tokyo.

Leveraging its unique Dynamic XML Runtime™ technology and XML Circuits™ application assembly framework, PolarLake's products allow customers to deliver integration solutions at a fraction of the normal time and cost.

CONTACT DETAILS

PolarLake Ireland (HQ)	**PolarLake Japan**	**PolarLake USA**	**PolarLake UK**
Block F1 East Point Business Park Dublin 3 Ireland	13 F Ebisu Business Tower 1-19-19 Ebisu Shibuya-ku Tokyo, Japan	1001 Avenue of the Americas, Suite 1121 New York, NY 10018 USA	No. 78 Cannon Street London EC4N 6NQ UK
T: +353 (1) 449-1010	**T:** +81-3-4360-3965	**T:** +1 (212) 813 2965	**T:** +44 (0) 20 7618-6426
F: +353 (1) 449-1011	**F**: +81-90-1421-6486	**F:** +1 (212) 790 9072	**F:** +44 (0) 20 7618-8001
E: info@polarlake.com	**E:** japan@polarlake.com	**E:** usa@polarlake.com	**E:** uk@polarlake.com

www.polarlake.com