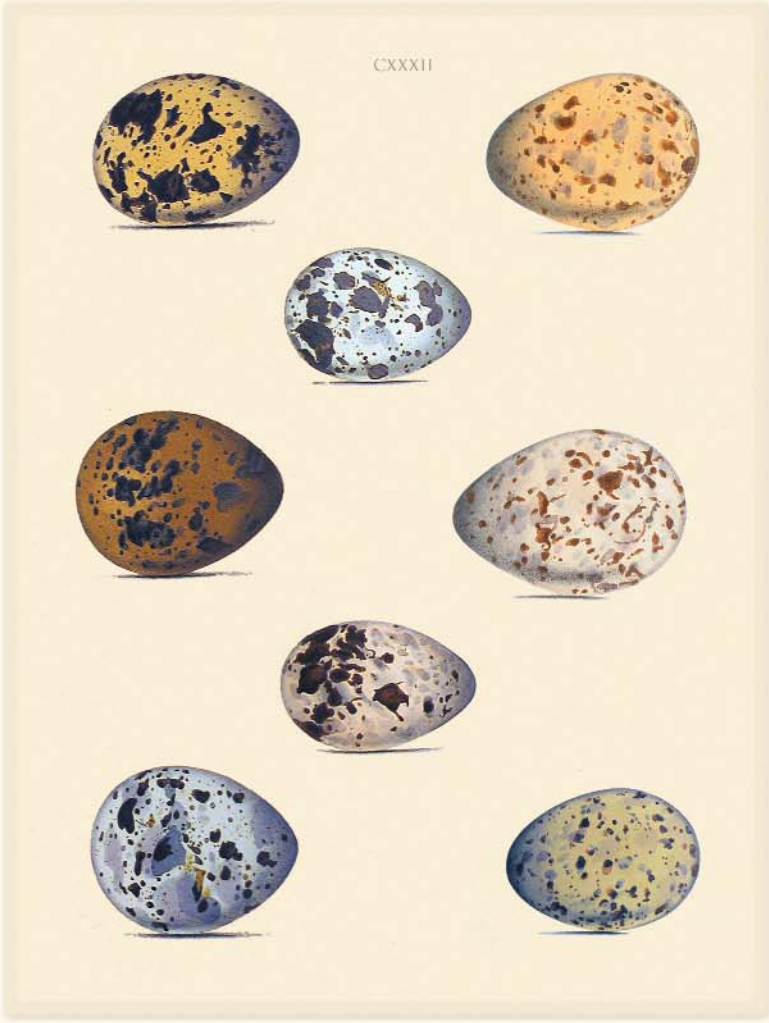
THEORY IN PRACTICE

Enterprise Service Bus



DAVID A. CHAPPELL

Necessity Is the Mother of Invention

3

The ESB is a new architecture for integration that is flourishing in corporations around the world. To many casual observers, the ESB as a technology category seems to have come out of nowhere. In reality, though, the ESB has not just "happened." Over time, many catalysts helped it develop and evolve, and lessons were learned from past technology approaches that extend back more than a decade.

This chapter will examine some key concepts of the ESB, including the many requirements, technology drivers, and forces in the IT climate that led to the creation of the ESB concept. All of this will be discussed in the context of the recent history and evolution of the ESB. This discussion will illustrate the point that an ESB is not merely an academic exercise; it was born out of necessity, based on real requirements arising from difficult integration problems that couldn't be solved by any preexisting integration technology. The discussion will conclude with a study of an ESB deployment, with a manufacturer exposing inventory management and supply chain optimization functionality to its remote distributors as shared services through an ESB.

Sometimes solving a problem requires looking at previous attempts at solutions and learning from their drawbacks. Entire trends had to come about as predecessors to ESB for the IT and vendor communities to have something to point at and say, "I like that," "I don't like that," or "That's what I've been trying to build on my own." The Greek philosopher Plato is credited for the phrase "Necessity is the mother of invention." The ESB is a shining example of invention fostered by necessity. In this chapter we will explore those necessities, and how an ESB addresses them.

The ESB concept is the next generation of integration middleware, capable of being applied to a much broader range of integration projects than what could be handled by specialized integration brokers. However, it should be stated that ESB is not just EAI plus web services, nor is it MOM plus web services. A number of recognized trends, both technology-driven and business-driven, have had an equal share of influence on the evolution of the ESB.

Enterprise Application Integration (EAI). A number of lessons, both good and not so good, have been learned from EAI. As we have seen, there are various downsides and painful lessons.

Much of the goodness inherited from EAI is in the "best practices" in data transformation and manipulation that can be carried forward into XML technologies and brought forward into an ESB architecture.

e-Marketplaces and vertical trading hubs. During the Internet bubble, this technology category was destined to change the model of how companies do business. The expectation was that e-Marketplaces would be universally adopted, inevitably replacing EDI with something more efficient and more accessible, and allowing companies of all sizes to participate in a supply chain. And while the e-Marketplace trend didn't garner the world's lasting attention as much as its early proponents had hoped, its existence on the hype curve caused businesses and IT culture to evaluate new ways of doing business electronically with other business partners.

This trend also helped foster the recognition of a need for open standards for protocols and service discovery mechanisms. e-Marketplaces were the first to introduce the model of a loosely coupled, distributed federation of individual companies operating autonomously, but still working together in a supply chain in a collaborative fashion. The e-Marketplace showed the IT sector that supply chains can be improved.

Java Message Service. JMS is a standard for APIs and behavioral semantics of MOM. The popularity of JMS as a part of the J2EE platform has brought messaging into the mainstream, and has created a marketplace for competing vendors building new messaging systems from the ground up based on today's requirement of communicating reliably and securely across the Internet.

Application servers. Application servers are important to an enterprise as a means for hosting business logic. They are not a key foundation component of the ESB per se, but they can be integrated using an ESB network. They are being listed here as a catalyst of the ESB concept in that they have nurtured the evolution of some important standards, such as the servlet environment for dynamic processing of requests, JDBC for database connectivity, and the J2EE Connector Architecture (JCA) for a standardized interface to application adapters.

Y2K, and post-Internet-bubble economics. Y2K readiness caused an increase in IT spending, with a significant shift toward the purchase of packaged Y2K-ready applications in favor of applications developed in-house. All the hype and excitement around emerging technologies during the Internet bubble led to continued IT spending. Nowadays, the post-Internet-bubble period has caused a major corporate reevaluation of big IT spending, and a shift toward trying to make things work with the existing applications and with a much smaller budget—even smaller and more highly scrutinized than in pre-Y2K times. The ESB is well suited for the new economics of integration, both in a monetary sense and in practical application.

BUILD AN ESB USING AN APPLICATION SERVER

One common question I get is "Why isn't an ESB built on an application server platform?" In the ESB approach to integration, the integration broker functionality itself is not layered on top of the application server. Integration capabilities are deployed as independent services alongside of, or independently of, the application server. There are a number of reasons for this. One is to avoid the over-bloating of functionality. Application servers are not necessarily the best place for every new technology trend that comes along. An ESB requires a lighter-weight container for deploying integration services without having to install an entire application server stack everywhere. And there are other reasons too: the core architecture of an application server isn't designed for hosting loosely coupled services. The deployment model isn't optimized for dynamic reconfiguration and deployment. The code-centric model embeds things into application code that should be dynamically configured, either explicitly or implicitly. The deployment and management model isn't appropriate for distributed deployment of heterogeneous services. A more detailed discussion of this issue is found in Chapter 6.

Web services and SOA. Web services are an industry effort to provide platform-independent SOA using interoperable interface descriptions, protocols, and data communication. Web services are a key core concept of the ESB, and the ESB can be thought of as a middleware manifestation of SOA design principles as applied to integration.

Evolution and maturation of standards in support of integration and interoperability. In addition to web services, there are important standards for XML, security, and reliable messaging. The development and adoption of standards, along with communities of supporting vendor implementations, have matured enough to make the benefits of standards-based integration become a reality.

The accidental architecture. As we now know, the accidental architecture is something that nobody sets out to create, yet everybody has. We examined the accidental architecture in Chapter 2, and explored how the ESB can help a system migrate away from the accidental architecture in incremental chunks.

The Evolution of the ESB

The creation of the ESB has been an evolutionary process. As we just discussed, a number of events in the industry had their part in catalyzing the creation of the ESB (Figure 3-1). This does not mean to imply that the predecessors of ESB were bad or failed technologies. Each contributing technology in the ESB ancestry was the best available for its time and continues to have its "meritt"[sic]. The ESB draws positive influences from its predecessor approaches, and avoids the downsides.

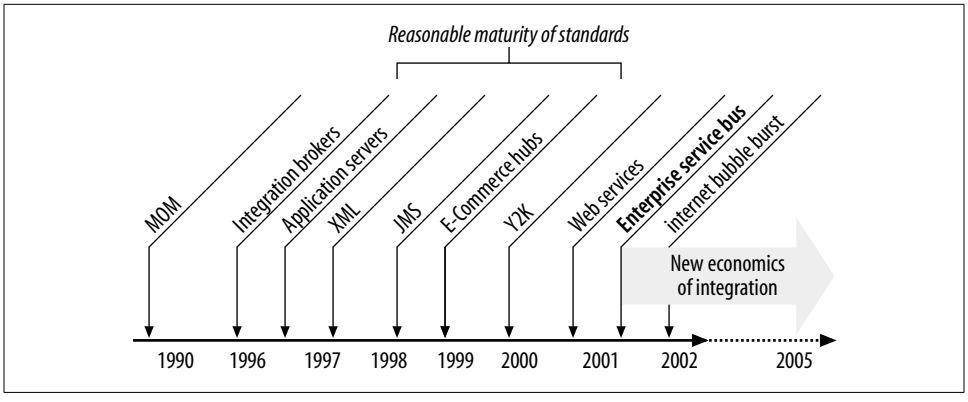


Figure 3-1. ESB catalysts: a timeline of technology and other events affecting the creation of the ESB

The invention of the ESB was not an accident. The ESB is a result of vendors working with forward-thinking customers who were trying to build a standards-based integration network using a foundation of SOA, messaging, and XML. These customers came from the end-user IT community in the manufacturing business, and from e-Marketplace infrastructure and trading exchange companies such as CommerceOne and GE Global Exchange Services (GE GXS).

The ESB in Global Manufacturing

A global company in the manufacturing sector is an example of an end-user IT organization that helped to catalyze the ESB. This manufacturing company is made up of at least five different major business units located around the world. Their goal was to have a common integration backbone based on message-oriented middleware infrastructure and standards-based interfaces. This effort was started and restarted several times, and had a resurgence of interest a few years ago, when the Java Message Service was first introduced as a way of providing standard interfaces to a common messaging backbone. This manufacturing company had many "islands of integration" and departmental pockets of proprietary and third-party message buses, which had been installed and controlled at a departmental or business-unit level. The requirement was that all departments and business units be integrated with each other to form a more consistent environment in which to plug in applications.

Their IT organization was looking to JMS to provide a standardized interface and common behavioral semantics across all applications, across all departments, across all business units. While many of us in the JMS business were excited about this, the reality is that JMS alone can't meet that requirement. The company also needed integration broker functionality such as routing based on rules and data transformation, all based on an abstraction of loosely coupled shared service interfaces.

At the time, the manufacturing company was faced with a dilemma caused by several things. While JMS had been selected as part of the solution, not all of the messaging vendors supported it. The existing messaging vendors and integration broker hubs that were entrenched in the various business units couldn't support the highly distributed approach required to span across global business units in a reasonable and manageable way. Some of the newer messaging vendors had the right JMS support, as well as the right security, firewall traversal, and message broker clustering to support the distributed topologies required to bring the geographically dispersed business units of the company together. However, there was no model in place for a service abstraction layer upon which to build an SOA, nor did the rest of the integration-stack layers exist that were required to integrate all of those applications across the organization.

Some of the forward-thinking IT leaders in this manufacturing company started working with the forward-thinking JMS and MOM providers. Through the course of such work in this global company and in others like it, many of the details of what eventually grew into early blueprints for the ESB were fleshed out.

The need was identified for a loosely coupled design center of event-driven services that could be coordinated across a common middleware substrate. The solution was to define a stack of functionality that consisted of a standards-based integration backbone, making use of JMS, SOA, XML, transformation and routing, and adapters. This approach went a long way, but still couldn't solve the connectivity issues.

The IT architects at the company realized that they couldn't just make a mandate saying, "On this date, all applications will be required to support JMS." It just was not practical. The same can also be said for web services. IT staff is too busy just trying to keep up with their daily work to embark on a mission that takes every application across the organization and retrofits it into one particular connectivity style.

Common APIs and event-driven service interfaces are a core part of the design center of an ESB. However, diversity in connectivity options is critical to the adoption of an integration strategy. Another need that was identified was a way to bring the infrastructure to the applications, allowing the applications to plug into the infrastructure in whatever manner made sense for them and facilitating an incremental approach to adoption. This is how the requirement of multiple client types, connectivity protocols, application adapters, and MOM bridges became a core part of the definition of an ESB. The various IT departments across the business units needed to protect their investments in existing messaging and integration broker installations, and be able to reach the other corporate applications as shared services in a nonintrusive way.

Putting It to the Test

In an effort to provide customers, suppliers, and vendors with one unified view of their diversified business units, the manufacturing company embarked on a major strategic operations initiative. The goal of the initiative was to provide one-stop shopping for

customer service, invoice and payment tracking, inventory management and ordering, and so on, as opposed to forcing constituents to deal with five different product units (Figure 3-2).

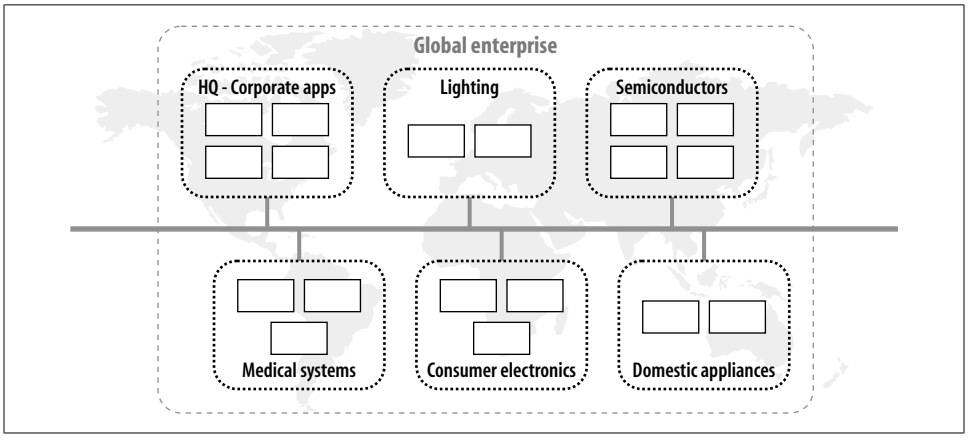


Figure 3-2. Loosely coupled, autonomous business units sharing a common integration backbone based on messaging and standard interfaces

The overall goal of the project was the creation of a single corporate IT backbone for integrating the payment processing applications across all the various product divisions worldwide.

The manufacturing company built its "payment automation" application on top of an early rendition of an ESB. This was the first test of the vision of a shared services network. The payment automation is based on an ESB-like implementation that serves as an enterprise-scale, centralized payment processing clearinghouse for all product divisions.

The payment automation backbone is a standard corporate payment service used by all divisions that allows the company to centrally manage, track, and clear orders for payment to vendors and suppliers worldwide. Because this project used an early ESB prototype, a corporate-wide payment service could be shared by all the business units, as a service plugged into the bus. Payment processing could now be routed through one shared business function, versus having to be cleared through a variety of geographically dispersed banks and banking systems.

Finding the Edge of the Extended Enterprise

For the past decade, through the era of EAI and the evolution of the Internet and application server technology, a clear dividing line has developed between the communications and application integration infrastructure within the four walls of a corporation, and the "external" communication with business partners, vendors, and customers.

This separation has been driven largely by the capabilities and limitations of the technology. To date, technology such as application server infrastructure has been specifically designed to make clear distinctions between what's inside the firewall and what's outside the firewall. This distinction is evidenced by completely separate architectural approaches, with different programming models required for building applications. In the J2EE application server architecture, for example, this is manifested as a web container versus an EJB container.

Hub-and-spoke EAI brokers could get as far as the corporate boundaries, but were not really built for scaling beyond that. Various bridging technologies were designed to bridge the gap at the "edge" of the network. In many legacy cases, this is "bolted on" as an afterthought. The majority of the work being done in the area of web services has also been focused on this "edge" of the network.

But just where exactly is this "edge" of the network anyhow? Before we get to that, let's explore another ancestor of the ESB, the e-Marketplace, also known as the e-Commerce Trading Hub.

e-Commerce Trading Hubs

In a trading network of business partners, there is the desire to move away from expensive EDI Value Added Networks (VANs) and use the public Internet as a means of communications wherever possible, and to lower the barrier of entry such that small to medium enterprises can afford to participate. This was the impetus behind the creation of e-Marketplaces and trading exchanges such as those powered by CommerceOne and GE Global eXchange Services (GXS).

A trading exchange acts as an intermediary, or semiprivate business portal, that facilitates electronic commerce between buyers and suppliers in a supply chain. The majority of the interactions within the "portal" are not browser-based—they are performed directly between specialized applications that require little or no human interaction. The interactions occur between applications residing in a trading partner, and backend applications residing in the trading hub. These backend trading hub applications provide value-added functions such as the dispersion of Requests For Quote (RFQs) between a buyer and multiple suppliers, or Availability To Promise (ATP) data from suppliers to buyers (Figure 3-3).

This environment introduced some rather challenging requirements, some of which seemed at odds with each other. For example, e-Marketplace deployment topology depicted in Figure 3-3 requires secure access between the applications at the partner sites and the applications in the trading partner hub, using the public Internet as the vehicle for communication. This scenario also requires reliable messaging, but the traditional MOM vendors did not have a MOM infrastructure that was capable of spanning across the public Internet in a scalable and secure fashion.

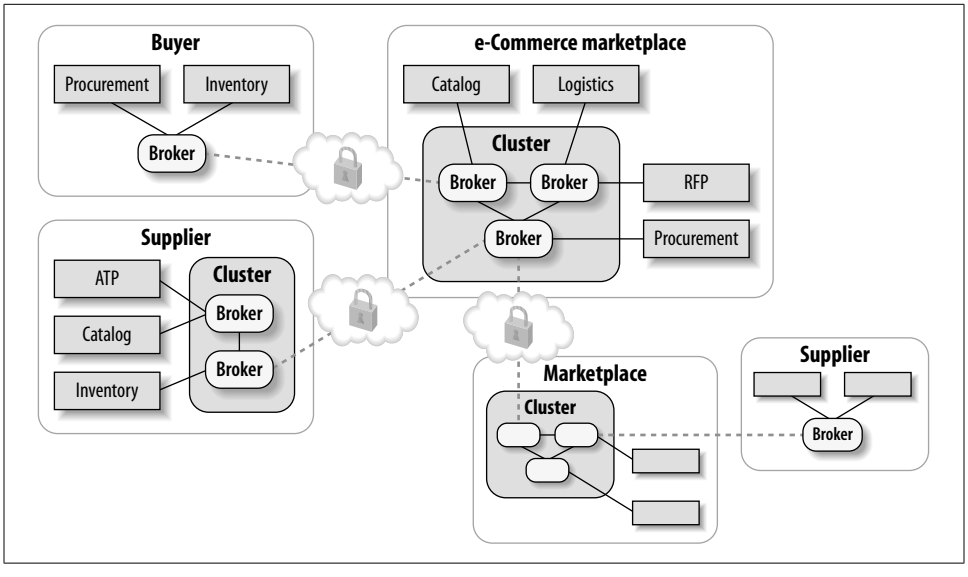


Figure 3-3. Deployment topologies of e-Commerce trading hubs encountered some interesting technical challenges

The suppliers communicating through the same trading hub are often fierce competitors with each other, so it is imperative that they not be able to see each other's data. This requires full authentication and access control between the trading partners and the trading hub. Functionality in the trading hub must be exposed to trading partners through a service interface. A supplier must also be able to freely share its data with the trading hub, and the trading hub applications must be able to selectively pass that data along to buyers, but not to other suppliers. A supplier must never be able to masquerade as another supplier and get access to its sensitive data.

An e-Marketplace community could potentially consist of thousands of trading partners, all communicating through the same trading hub. Trading partners need to be able to asynchronously communicate with the trading hub in a "fire-and-forget" mode using reliable messaging.

Some very important ESB capabilities were born out of these requirements. For example:

- Strict authentication and access control between the entities connecting to each other
- Scalable clustering of message servers (Figure 3-3) to handle large volumes of message traffic from potentially thousands of concurrently connected trading partners
- Complete segregation of data channels
- Selective control over which channels can be opened up between application endpoints, across intermediary hubs

- Selective access to shared service interfaces and endpoint destinations
- Coordination of the business-level message exchange between applications that are separated by physical network domains and geographical location
- Secure MOM communication through all the firewalls that exist between the trading hub and the suppliers and buyers

Another contributor to the vision of the ESB architecture was the requirement for a trading hub to do business with other industry-related vertical trading hubs. This means that segregated groups of applications (the trading hub backend apps) need to selectively expose and share their interfaces and data with groups of applications residing in other trading hubs. Each trading hub and each trading partner needed to be able to maintain their own autonomy and local integration environments while communicating in a larger e-Commerce network. This network of trading hubs and trading partners could potentially fan out ad infinitum.

The evolution of e-Marketplace infrastructure has significantly contributed to the emergence of ESB providers offering the underpinnings to support the requirements of e-Marketplaces. The vendors building trading exchanges looked at a variety of EAI brokers and application server technology, and turned to the messaging vendors for help. Some of the newer messaging vendors were already beginning to provide the required underpinnings for the routing, segregation, and fan-out deployment topology. This process of defining requirements, talking to vendors, and designing this infrastructure took more than a year. During this time, a number of messaging vendors and EAI broker vendors put a great deal of effort into ensuring that their next-generation products would be able to support the requirements of e-Marketplace vendors, and this helped to contribute to the emergence of the ESB concept.

The Extended Enterprise: The Ever-Changing Edge

Fast-forward a couple of years, and it turns out that the technology requirements of large-scale trading exchanges are the same requirements of corporations building out their own integration networks. While the e-Marketplace never really took hold as a business model, there remained a need to provide common shared services across departmental and corporate boundaries. This need expands well beyond supply chain scenarios.

Due to mergers and acquisitions, collaborating business partners, and globally distributed business units, varying modes of communication are based on the degree of trust between the business entities. This model represents the "extended enterprise." In the extended enterprise, the "edge" of the network is always changing—or perhaps there was never really a single outer edge to begin with. For example, in a global organization of semi-independent business units, there are many firewalls, but also a need to have a distributed integration backbone that transcends the underlying topology (Figure 3-4).

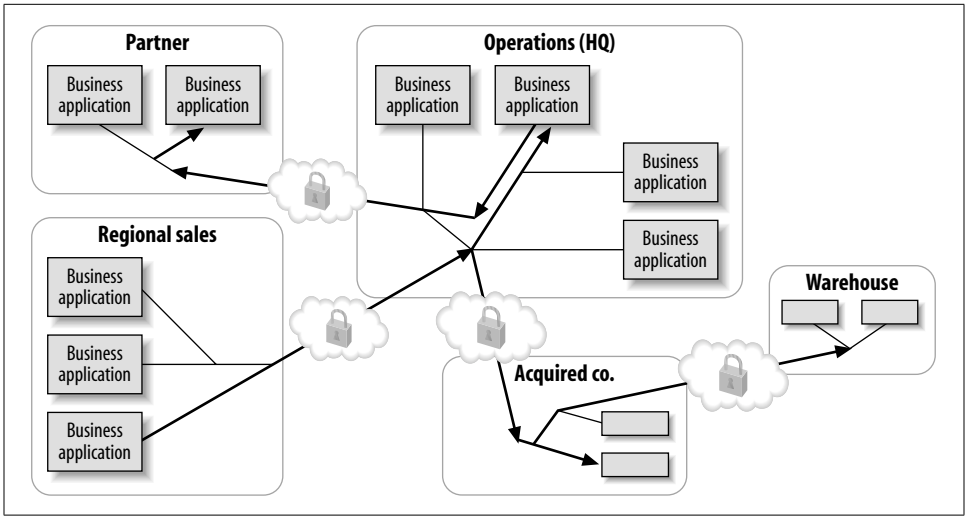


Figure 3-4. Corporate IT domains: intra-corporate requirements have technology challenges very similar to e-Marketplaces

There are different levels of trust when dealing with business partners. Reliable messaging requires that a piece of MOM software be installed on either side of the communication link. Sometimes it is acceptable business practice to tell a business partner, "If you want to do business with me, you must install this software at your site." In such a case, it's perfectly acceptable, from a technology point of view, to have an efficient, reliable MOM link between the two organizations. Chapter 5 will provide more information about MOM wire protocols, and new reliable SOAP protocols that can help (but not completely alleviate) the requirement to have MOM software installed on both sides of the wire. When your relationship with a business partner is not as close, you may instead need to supply clear instructions on how to send business documents over a secure HTTP link, and manage business message exchanges using a layered service that specializes in partner links.

Corralling the Ever-Changing Edge of the Network

The ESB provides the architecture that separates the higher-level SOA and integration fiber, which includes the management of physical destinations as abstract endpoints and the transformation and routing, from the details of the underlying protocols. This means that the network boundaries can change over time, and an ESB can support the changing of the underlying protocol (i.e., from MOM to SOAP-over-HTTP to WS-Rel*) without affecting the higher-level integration functions that sit above all of that. In the trading hub example, SOA, data transformation, and routing of messages based on context was the job of the value-added services that resided inside the hub. As we take the lessons learned and the concepts of trading hubs and apply them toward in-house IT integration, we can make those same concepts available as independently deployed services.

Standards-Based Integration

The maturity and adoption of relevant standards for integration have helped to foster the emergence of the ESB as a technology trend. The ESB makes full use of standards for integration wherever possible. The use of such standards can have significant effects on a business, as follows:

- Allows you to leverage existing IT staff, rather than specialist consultants. The amount of information available on XML and web services standards such as SOAP, WSDL, XSLT, XPath, and XQuery is expanding at an ever-increasing rate. There is an educational ecosystem in which information about standards (and budding specifications) becomes available as the standards evolve. The introduction of a popular specification creates a fertile ground for industry experts to write articles, tutorials, and O'Reilly books on the subject, which in turn allows IT professionals to learn more and stay current. This means that the average IT professional can readily attain the expertise he or she needs to become the in-house integration architect.
- Reduces proprietary vendor lock-in. With a proprietary integration stack, an organization can't simply say, "Well, vendor A wasn't what we expected, so let's try vendor B." This would result in an expensive restart and relearning. Adopting standards as part of an integration infrastructure means the ability to pick and choose best-of-breed implementations from different vendors and have them work together.
- Java standards. While the ESB concept can be Java-free, Java provides a set of specifications that don't exist elsewhere and that can standardize components and interfaces between those components. These standards define application interfaces, behavioral semantics of middleware and application adapters, and deployment models. Standards such as JMS, MessageDrivenBeans, and JCA can significantly increase the ability to interoperate between enterprise applications and J2EE application servers. J2EE application servers from various vendors have found their way into most, or perhaps even all, enterprise organizations.
- Increases the ability to integrate with business partners using standard interfaces and standard protocols.
- A JMS interface to an ESB is currently the only way that application servers from different vendors can talk asynchronously using reliable messaging in a common environment with an SOA. RMI-over-IIOP is (or was) the "blessed" method of application server interoperability, but it doesn't provide a model for loosely coupled interfaces or SOA, nor does it work outside of J2EE. While in theory any J2EE server can interoperate with any other J2EE server using JMS, in practice this doesn't really happen out of the box, nor is there any incentive to do so. An ESB can provide a neutral ground where application servers from multiple vendors can communicate with each other.

- J2EE Connector Architecture (JCA) can also provide the standard contract for application adapters, which can add to your arsenal of integration with any packaged applications that support Java interfaces. Entire suites of application adapters, available from multiple vendors, use JCA as the unified way of connecting between the adapter and the middleware infrastructure. Once an application is introduced into the ESB through a JCA adapter, its functionality is exposed as a standards-based, event-driven service. This gives you much greater flexibility and reuse than if it were plugged into a proprietary broker through a proprietary adapter.

WHAT ARE "STANDARDS," EXACTLY?

When talking about the use and adoption of standards, it is hard to tell just exactly what that word means. We live in a world of multiple overlapping efforts from different standards bodies to define standard specifications. Vendor alliances are producing web services specifications outside the domain of any standards body or consortium.

Every once in a while I am challenged on the use of a Java specification as a "standard." The argument is that because the Java Community Process (JCP) is owned by one vendor (Sun Microsystems), the specifications that come out of that process are not really *standards*—they are just specifications.

I tend to use the words "standards" and "specifications" interchangeably. As far as I'm concerned, any specification that has been through the JCP should be considered standard enough. That is, it has gone through a formal process in which it was jointly defined by a group of independent companies and/or individuals, and was posted for public review prior to ratification.

There are also "de facto standards" such as open source implementations, which either invented their own ways of doing things or conform to industry-accepted "standard" specifications. The Apache SOAP and Apache Axis toolkits are examples of such standards.

The evolving WS-* stack of specs from ad hoc vendor collaborations can also be considered de facto standards in that they represent the work and statement of direction for a meaningful constituency of the largest platform vendors. Some of these WS-* specifications have already been submitted to an official standards body, and the rest have been part of a program that involves public feedback sessions.

In short, the word "standard," in terms of standards-based integration, refers to either a specification or an implementation that has gained enough traction in the industry to have long-lasting staying power, and is open enough for multiple vendors to implement or repackage it.

An SOA built upon the combination of enterprise messaging with certain key technologies such as SOAP, WSDL, XPath, and XSLT, with interfaces to Java, .NET, and C++, collectively defines the means for a platform that allows cleaner solutions based on

standards. The concept of standards-based integration allows developers to learn and build a valuable skill set that can be used across a variety of integration projects. Integration based on standards also provides IT managers with a larger talent pool of developer resources, and allows for repeatable success patterns that can be carried from project to project.

A major European food distribution network was an early adopter of an ESB, and successfully completed their first supply chain automation project in six weeks. One of their directors of IT strategy had this to say about the use of standards for integration in an ESB: "Now we no longer have to worry how long the next integration will take, or even if it is possible."

The New Economics of Integration

In the ESB, you deploy what you need, when and where you need it. The licensing model being put forth by vendors leading the charge in ESB technology reflects these physical deployment characteristics.

According to a Forrester Research report, license costs for integration brokers begin at $100,000 per project and have an average price of $400,000 to $750,000. In contrast, an ESB license can cost 10 to 15 times less than that. Does this mean that ESB vendors have an unrealistic licensing model that is incapable of sustaining a business? No. The ESB licensing trend is based on the philosophy that integration should be pervasive throughout the enterprise, and a high cost of licensing should not be a hindrance to adoption. This licensing philosophy reflects the technology model, which is to license only what you need on a per-project basis, while building toward the strategic goal of corporate-wide integration. So you can start wherever you need it most, and grow at your own pace without costly obstacles.

For integration brokers, you typically pay for consulting services that are four to five times the licensing costs. Because the ESB is based on standards, you can leverage in-house staff and avoid having to pay high fees for consultants who specialize in proprietary integration broker technology. By investing in the adoption of standards and educating your in-house IT staff on standards for integration, you are future-proofing your staff as well as your technology.

Driving Down the Cost of Technology

In his book *Loosely Coupled*, Doug Kaye illustrates his own version of a "technology adoption curve" in support of a discussion on how the cost of a particular technology decreases over time. As an example, he talks about the publishing of O'Reilly books on a particular subject as a significant event in the adoption of a technology, helping to drive down the cost of the technology by making knowledge about it more readily available. You are witnessing that now in terms of the ESB.

Case Study: Manufacturing

As an example of how ESB technology is changing the economics of integration in a real deployment scenario, let's take a look at how a building material manufacturer is using an ESB. We will pay special attention to how they are taking advantage of the selective deployment model of the ESB.

The manufacturer operates 50 plants in 15 countries. They distribute their products through large building supply retailers, such as Home Depot and Lowe's, as well as through a network of more than 60 independent distributors serving retail and wholesale customers in regional markets across the United States. Twenty-eight of their larger distributors are connected to their headquarters using an EDI VAN.

Connecting one supplier with 60 distributors is a simple challenge for an integration infrastructure that is capable of scaling out to thousands of diverse endpoints. However, the point of this case study is not the scale of the deployment, but the traits of the ESB that it utilizes. Its simplicity highlights yet another characteristic of the ESB—that it is capable of scaling up to large global integration networks, but is also well suited for small projects.

Building a Real-Time Business

To improve operations and optimize the distribution chain, this manufacturing company embarked on building an infrastructure that would enable direct distributor participation in inventory management and ATP as a means of achieving real-time order fulfillment.

Inventory management

The inventory management application allows the manufacturer to better anticipate the demand of its distributors by tracking each distributor's monthly inventory consumption and creating a recommended monthly order requisition for each distributor.

By deploying an ESB, the manufacturer can now analyze its distributors' order and sales histories, thus allowing inventory to be jointly managed. This was accomplished by implementing a message exchange that allows the manufacturer to anticipate a distributor's inventory requirements. In this scenario, distributors periodically send product activity data to the manufacturer, which uses that data to anticipate product consumption activity and determine when the distributors need to replenish the stock. They then generate a shipping schedule message indicating the products and quantities that the distributor should order to replenish its stock. The distributor will use this data to generate a purchase order back to the manufacturer.

This process allows the manufacturer to better manage inventory, reducing the quantity of on-hand product needed for the distributors and achieving more predictable production schedules and revenue forecasts.

Technical challenges

The manufacturer looked to integrate the 60 disparate ERP systems of their distributors with their own SAP R/3 inventory management and order processing systems. One of the challenges they faced was accommodating varying levels of technical sophistication among their broad range of distributors. They already had an EDI infrastructure for about half of the larger distributors, but that was a high-cost solution that would not be viable for the smaller distributors. Some of the larger distributors had the technical capability to communicate with them using SOAP and web services, but some of the smaller distributors didn't have the sophistication or technical know-how to do that. Most distributors had an ERP system locally, with the extent of their IT expertise limited to how to operate it. The manufacturer needed a simple solution that could be installed at each of the distributor sites, and configured and managed remotely by a central IT staff.

Availability To Promise (ATP)

The manufacturer has also deployed an ATP lookup for nonstock items. ATP is a common business function in supply chains that allows you to accurately predict the delivery of goods between multiple buyers and suppliers. This is particularly important when the immediate supplier does not have the item in on-hand inventory and needs to custom-build it, which could involve another level of special-order items from its own suppliers. Being able to plan and predict the timely delivery of nonstock items can significantly reduce the unnecessary cost overhead of carrying special inventory items to effectively meet the demands of customers and avoid lost business.

Flexible Partner Integration

The solution for both the inventory management and ATP issues was to deploy an ESB. For the inventory management function, they needed a remote integration solution that was easy to install, deploy, and integrate with the many disparate ERP systems at each distributor site. For the ATP solution, they decided to expose the ATP function as a service and use a direct web services link with the distributors.

The inventory management function is facilitated by an ESB-managed business process that controls the routing between the partners and the inventory management application, which is implemented in SAP. There are no integration brokers or application servers required at the remote sites. Still, the solution provides standards-based connectivity, reliable and secure messaging, and transformation and service orchestration between its own systems and the individual ERP systems utilized by the remote distributors. The "edge" of the ESB network is capable of supporting a variety of connectivity options as new distributors come on board. Even existing links with distributors can change their underlying protocol without affecting the higher-level business processes and XML message exchanges between the manufacturer and the distributors.

To manage the link between distributors and headquarters, the manufacturer chose to deploy an ESB service container at the remote distributor sites (Figure 3-5).

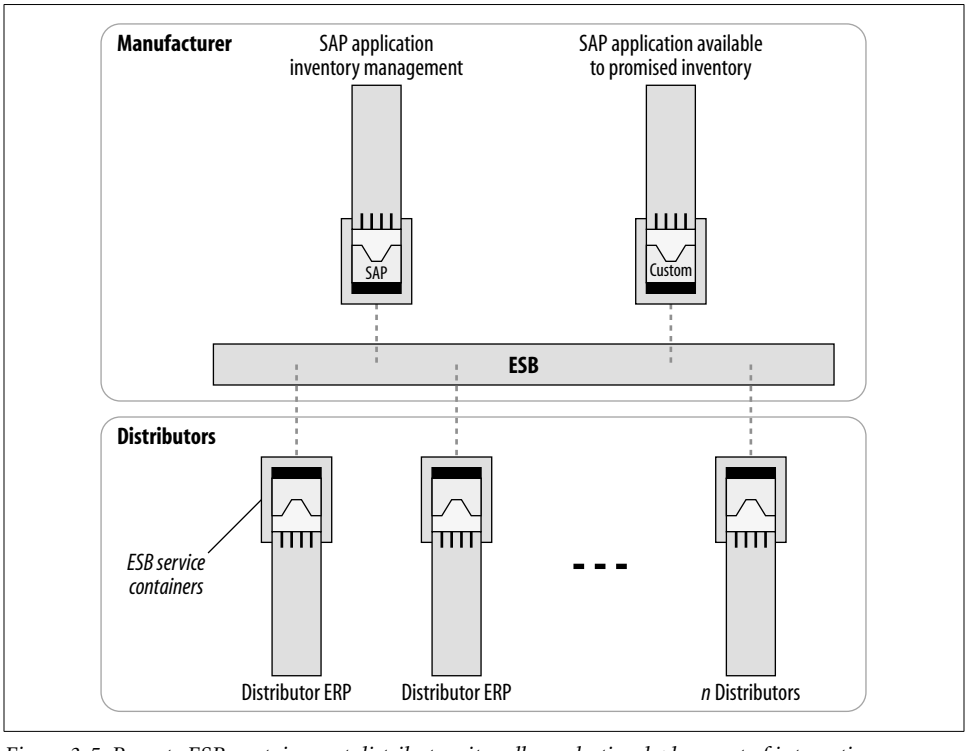


Figure 3-5. Remote ESB containers at distributor sites allow selective deployment of integration capabilities

Remote ESB containers located at the distributor sites allow selective deployment of integration capabilities when and where they are needed, with no integration brokers or application servers required. In addition, this approach provides both the manufacturer and their distributors the following advantages:

- A secure, authenticated channel between the distributor and headquarters
- Reliable delivery of messages, both synchronous and asynchronous, between the distributor and headquarters
- Simple integration at the partner site
- A managed environment, where deployment configuration and management can be handled remotely by the manufacturer's IT staff, without requiring any additional IT expertise at each partner site
- No need to license and install an integration broker, application server, or "partner server" at each distributor location

- A link to an EDI VAN, and transformation between EDI messages and XML messages
- Integration with SAP within the manufacturer's inventory management system

Within the ESB network at the manufacturer, data transformation between comma-separated ASCII (CSV) files, an internal canonical XML format based on Common Business Library (xCBL), and SAP's Intermediate Document (IDOC) format is accomplished as part of several orchestrated business processes facilitated by the ESB. More details on the virtues of a canonical XML format are discussed in Chapter 4. Chapter 8 will explore the more technical details of this deployment.

Summary

In this chapter, we saw that the ESB concept was developed out of necessity, and had many catalysts. We also explored the following:

- The ESB provides a pervasive, event-driven SOA, which is based on requirements of IT architects working together with vendors to build broad-scale integration networks using messaging, standard integration services, and standard interfaces.
- e-Marketplaces provided a fertile breeding ground for scalable and secure ESB infrastructure capable of supporting the needs of large trading hubs with potentially thousands of trading partners. Out of this environment, the requirements of sophistication routing across segregated data channels were identified.
- The proliferation and reasonable maturity of standards has provided the benefits of standards-based integration.
- Application servers have their place in IT infrastructure as containers for housing business logic. An ESB architecture can provide a loosely coupled integration fabric for integrating application servers with other application servers and cross-platform applications at large.
- Remote ESB service containers obviate the need to install integration brokers at every remote site to be integrated.