Understanding the Enterprise Service Bus

Understanding ESB

The Enterprise Service Bus, or ESB, has recently become the subject of intense interest by enterprise customers and heated debate by technologists—a clear sign that the idea either carries significant merit or is compelling hype without substance. Certainly the noise level alone on ESB warrants an open-minded examination of the concept.

Defining the ESB

There are multiple definitions proposed for an ESB, which some critics point to as evidence that there is no such thing as an ESB. It's also true that some vendors use ESB definitions that are self-serving. We have no wish to be self-serving, nor do we want to add yet another definition to the mix. Instead we have assembled a synthesis of popular ESB definitions, and we have discovered that they are not as diverse as is often claimed. Most ESB definitions encompass a healthy subset of the following composite definition.

- An ESB is a backbone for connecting and integrating an enterprise's applications and services.
- An ESB provides the necessary infrastructure to create a service oriented architecture.
- An ESB is a convergence of EAI, MOM, and SOA concepts.
- An ESB is based on open standards such as XML, SOAP, and WS-*.
- An ESB provides intelligent routing, such as publish-subscribe, message brokering, and failover routing.
- An ESB provides mediation, overcoming data, communication, and security differences between endpoints.
- An ESB integrates with legacy systems using standards-based adapters.
- An ESB provides logical centralized management but is physically decentralized.
- An ESB is able to apply EAI concepts such as rules and orchestrations.
- An ESB is able to monitor and throttle activity as per a Service Level Agreement (SLA).

We would expect a true ESB to satisfy most of these points and an imposter ESB to score low. Of course, not all enterprises necessarily need all of these capabilities, so even a partial ESB solution could provide real benefit to some classes of customer.

Although it's not part of the composite definition, we would propose adding one more point that should help distinguish a true, well-intentioned ESB design from a deceptive offering:

- An ESB is modular and product-agnostic; its parts can be replaced with new implementations.

Let's round out the definition by exploring the above points in a bit more detail.

A Connection and Integration Backbone

In the ESB model, most or all applications and services in the enterprise connect to the ESB and communicate with each other over the ESB. Applications and services usually connect using SOA standards, whereas legacy systems require integration via EAI technologies such as adapters. The communication between endpoints is handled by message oriented middleware.

The ESB serves as a common messaging fabric for the enterprise. Programs connect to the ESB and send or receive messages. The ESB handles routing details, mediation of differences between endpoints, and the physical details of communication. It's far more sensible to put such matters in the hands of I.T. personnel who can make enterprise-level decisions than having them controlled by developers at the application level.

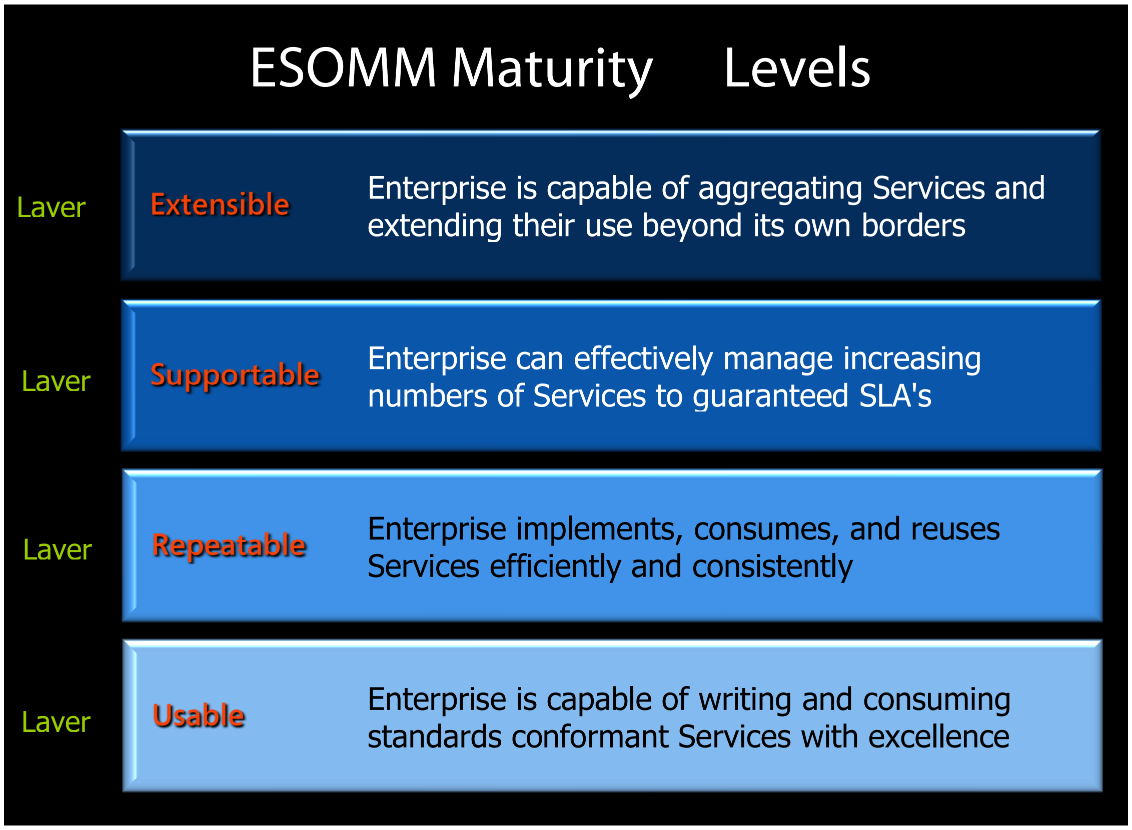
An Infrastructure for SOA

As most SOA enthusiasts will tell you, the "A" in Service Oriented Architecture is still missing in action (and some prefer the term "Service Orientation" for this reason). Service orientation has given us a good set of principles (such as loosely coupled communication), an excellent set of standards that are composable and ongoing (WS-*), and compelling new technologies such as Windows Communication Foundation. In short, while service orientation is still young, it's real enough to be usable here and now and is being put into practice everywhere.

As enterprise adoption of services continues, the need for a service oriented architecture will start to be felt. There's a big difference between the casual use of services and running your enterprise primarily over services. As enterprises travel down the road that will take them from light use of services to deep use of services, many issues will arise, such as how to manage large numbers of services well; how to overcome differences between services; how to enforce SLAs; and how to enforce enterprise policies across distributed collections of services.
The Enterprise Service Orientation Maturity Model (ESOMM) shown in Figure 1, and described in detail at http://www.architec-

turejournal.net/2006/issue7/F6_Enable/default.aspx, reveals the implications of running an enterprise on services. Just about every enterprise is on the bottom rung of this ladder.

system added increases the problem of configuration and management exponentially. This was the impetus that led us to hub-and-spoke architectures, which most EAI products use. This architecture was a vast improvement over point-to-point architectures, and



ESOMM Maturity Levels

Laver	Extensible	Enterprise is capable of aggregating Services and extending their use beyond its own borders
Laver	Supportable	Enterprise can effectively manage increasing numbers of Services to guaranteed SLA's
Laver	Repeatable	Enterprise implements, consumes, and reuses Services efficiently and consistently
Laver	Usable	Enterprise is capable of writing and consuming standards conformant Services with excellence

Figure 1: ESOMM

What would an ideal service oriented architecture look like? Although many in the SOA camp are championing complete decentralization, it's instructive to visit past architectures that have fallen in and out of favor over the years (Figure 2). Point-to-point architecture works all right on a small scale, but its problems become apparent when used at the enterprise level. If each system has to know the connection details of every other system, then each new

each system needed to communicate with only the hub. In addition, the hub could provide excellent management features since it was a party to all communication. It only took time to reveal some shortcomings with the hub-and-spoke approach, and today it is often associated with concerns about scalability, single point of failure, and vendor lock-in. The lesson from this is that it is possible to be overly decentralized or overly centralized. Fortunately,

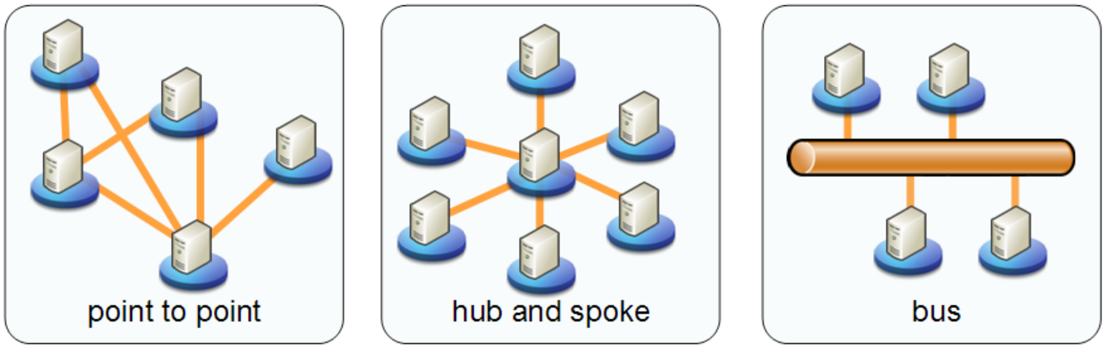


Figure 2: Messaging Architectures

An Infrastructure for SOA continued:

there is a sound compromise to be found in the bus architecture, which provides the benefits of logical centralization but is physically decentralized. The bus architecture in earlier days was often used in message bus systems based on proprietary technologies, but an ESB implements this architecture using WS-* standards.

Think of an ESB as a set of infrastructure services that complement your business services. Those infrastructure services provide valuable functions such as routing, storing and forwarding of messages, activity monitoring, transformations, and EAI functions like applying rules or executing an orchestration. They also happen to be in on the enterprise's master game plan for configuration, policies, and service levels which they cooperatively help enforce. Infrastructure services make sense, and the idea is hardly new: most enterprises already contain such things as domain controllers and active directories.

The Convergence of EAI, MOM, and SOA

While the ESB can be described as a recent approach to enterprise connection and integration, it stands firmly on the shoulders of three disciplines that have already proven their worth in the enterprise. The ESB concept is made possible through the convergence of Service Oriented Architecture (SO/SOA), Enterprise Application Integration (EAI), and Message Oriented Middleware (MOM). The rapid progress being made in each of these disciplines is all leading in the same direction, convergence.

It's tempting to define ESB in familiar terms, and thus many with a singular EAI, MOM, or SOA orientation tend to simplify their concept of ESB to reflect their favorite discipline and all but ignore the others. In actuality, an ESB needs to be strong in all three areas or it will be seriously deficient. The very term Enterprise Service Bus makes reference to all three disciplines.

The superset of capabilities that comes from combining SOA, EAI, and MOM is extremely compelling. SOA makes loosely-coupled, decentralized solutions possible that are enterprise-ready and based on interoperable standards. EAI allows integration of any combination of systems, with sophisticated message brokering, message translation, business process orchestration, and rules engine processing. MOM provides intelligent routing such as publish-subscribe topical messaging and strong managerial controls over routing, auditing, activity monitoring, and throttling.

If combining disciplines was all there was to an ESB, we'd simply call it "consulting". To properly leverage these disciplines they need to be combined in the right way through an architecture that lets their strengths shine and overcome their inherent weaknesses. Each discipline has some weak areas that the others help to resolve: SOA needs better enterprise manageability; EAI needs to become decentralized; MOM needs to get away from proprietary technologies. Combining these disciplines properly in an ESB overcomes these weaknesses.

Standards-Based

An ESB primarily connects to services using the standards that have been developed for first-generation web services and second-generation SOA services (SOAP, WSDL/XSD, WS-*). This approach should be followed not only for business endpoints but also infrastructure services such as a transformation service. An ideal ESB not only uses standards for external connections but is itself made up of standardized services.

The use of open standards provides the enterprise with enormous flexibility. It's often difficult to predict where change will come from; events such as taking on a new partner or acquiring another company can make sudden demands on the enterprise that were not anticipated and have to be accomplished quickly. With standards based connections between the ESB and its endpoints, any endpoint can be revised or reimplemented without disrupting the rest of the enterprise. Disruption can even be avoided if contracts change because the ESB can mediate differences between programs. An ESB based on open standards gives customers complete freedom to select best of breed business products and technology products.

The use of open standards protects the enterprise by minimizing risk. The extensible, composable WS-* standards represent long-term thinking by the industry. Using these standards instead of something proprietary also makes it much easier to find qualified developers and I.T. personnel.

Intelligent Routing

Traditionally, enterprise developers write applications that include both business logic and communication logic. In the past, that communication logic made assumptions about which systems information came from and which systems information should be sent to. The problem with that model is that it requires the developer to have an enterprise-level view of how systems interconnect. Even worse, that understanding is then embedded into the code. If the enterprise connections change, the application has to change as well. Intelligent routing takes that burden off of the developer and makes it a concern of business analysts and the I.T. department. Routing is determined by a centralized model, using mechanisms such as publish-subscribe and message brokering. The routing can be changed at any time, without requiring any changes to application software.

Intelligent routing provides more options for message routing than are typically available to the application developer. Publish-subscribe messaging makes it easy to add or remove senders and receivers, linking them through subscriptions. Message brokering routes messages based on payload type (schema) or content. Workload sharing can distribute messages evenly across multiple instances of a service.

Intelligent Routing continued :

Failover routing can redirect messages to an alternate location when a service has failed or been taken out of service.

Some ESBs provide an alternative to middleware routing where clients are matched up to services via the ESB, and then communicate directly. Although this avoids the intermediate communication over middleware, it also means giving up most of the benefits of intelligent routing and mediation.

Mediation

Differences in applications can be overcome through mediation. An ESB can bridge differences in protocols, message formats, security models, and communication semantics. It's much simpler to configure an ESB for mediation than it is to make changes to the programs themselves.

One form of mediation is transformation, where messages are converted into a format understood by the receiver. Transformation is a key tool for supporting message versioning. Transformation can also be used to reconcile differences in messages to or from parties who may have differing requirements about format and content.

Protocol mediation overcomes differences in communication protocol, such as connecting a program that sends via queuing to a service that receives via HTTP. Differences in communication semantics may also need to be mediated, as in connecting a program that expects a request-reply pattern of communication with a program that performs one-way communication. Security mediation handles differences in security models, as in the case of an enterprise application that uses Windows integrated security conversing with a service that uses X.509 certificates.

The final aspect of mediation is differences in time. A sender may be transmitting messages when some receivers are not currently available. Mediation can store messages until a receiver is available to receive them. Even though a sender and receiver might be using non-durable protocols such as HTTP, mediation between the endpoints is free to use other means, such as transacted durable queuing.

Integration

As important as it is to support open standards and provide a good foundation for promoting the use of services, enterprises often have legacy systems that need to be full participants in the messaging arena. An ESB needs to be able to leverage adapter mechanisms in order to bring non-services onto the bus. This might be accomplished by leveraging the adapters in one or more existing EAI products.

EAI products often contain a suite of adapters that the hub uses to talk directly to legacy systems. To avoid the hub-and-spoke model and vendor lock-in concerns of the past, a better arrangement is to have the ESB call a service to communicate with a legacy system. By placing the adapter within a service, any implementation changes take place in the service, not the ESB's internals.

Logically Centralized, Physically Decentralized

Much of the value an ESB provides is in the area of centralized management. An ESB provides a single place to handle management functions such as configuration, deployment, monitoring, and control. Having a central facility like this makes change management straightforward and rapid response possible. Not having centralized management creates a significant problem for I.T. departments and imposes unnecessary cost and delays when change is required.

Centralized management used to come at the cost of a centralized architecture, but an ESB is physically decentralized. There are no hubs. The only thing that is really centralized is the enterprise game plan itself; the infrastructure services that cooperatively execute and enforce the plan are themselves distributed. This is just like a football team where the individual players are distributed on the field, but the team is carrying out one plan.

Some ESB detractors claim there's no need for any kind of centralization at all as WS-* services become smarter and smarter nodes. It's difficult to see how this could be accomplished without a common plan and some key infrastructure services to enforce security and other policies. A completely decentralized system sounds like a return to point-to-point architecture.

An ESB repository tracks such things as endpoint metadata, schemas, contracts, and routing connections. An ESB repository doesn't have to be a single entity in its implementation, and may well draw from other directories in the enterprise such as UDDI directories and Active Directory.

An ESB promotes configuration changes instead of program changes. Configuration changes are simpler, safer, and can be applied without disrupting running systems.

EAI Capable

An ESB needs to be able to provide the kind of processing traditionally provided by EAI products. This potentially includes validation, transformations, rules engine processing, business activity monitoring, and execution of orchestrations.

In a traditional EAI product, these EAI functions tend to be applied, pipeline-fashion, on a single server or server cluster. An ESB provides the same functionality, but these EAI functions are exposed as individual services. While messages are en route, the ESB calls these EAI services as they are needed.

EAI Capable continued :

Exposing EAI functions as services keeps the ESB true to its philosophy of standards-based connections and allows any individual function to be replaced. An enterprise is free to choose whatever implementation they wish for these EAI services, possibly choosing to use one or more established EAI products to power them.

SLA Aware

SLA enforcement is a difficult job, but ESBs are in the best position to report on and control enterprise messaging activity. Many feel strongly that SLA monitoring, reporting, and enforcement should be left to companies and products that specialize in this area, and there isn't a reason why an ESB cannot team up with such products.

Since most enterprise messaging will pass through an ESB's middleware messaging system, messaging activity can be easily tracked. Controls may be available to throttle messaging activity. For example, it may be desirable to slow down lower priority messaging in order to give more bandwidth to higher priority messaging.

Modular and Product Agnostic

A subject of much debate is whether an ESB is a pattern or a product. The most obvious answer would seem to be "both": an ESB can certainly be described as an architectural pattern and there is more than one path to creating one. You've got to build your ESB out of something; so it's natural that one or more products would be used to accelerate the process.

We feel the ESB is first and foremost a pattern, and one built for the long term. It should be possible to survive incremental and generational changes in technology without having to throw away the pattern. The ESB should walk its own talk: not only should business endpoints be exposed as services that can be modularly replaced but so should the ESBs own internals.

Traditional EAI/MOM/SOA products will be helpful in creating an ESB, as may products specifically designed with an ESB in mind. But the design pattern aspect of an ESB is a higher order bit than any products it is created from. It should be possible to mix and change out the underlying products that power the ESB without disrupting the enterprise.

Business Benefits of an ESB

An ESB improves business agility, streamlines business execution, expands business intelligence, and decreases costs.

IMPROVE BUSINESS AGILITY

An ESB makes an organization more nimble, more responsive, and more adaptive. The ESB matches up events—whether expected or unexpected—with appropriate I.T. services. Unlike many traditional infrastructures that are resistant to change, an ESB is designed to accommodate change easily and painlessly. An ESB's flexibility makes it easier to add new partners at a moment's notice, pursue a sudden business opportunity, or improve time to market for a new product or service.

STREAMLINE BUSINESS EXECUTION

An ESB improves an organization's ability to execute efficiently and meet its commitments consistently. Business rules can be enforced across the breadth of the enterprise. The availability, capacity, and responsiveness of enterprise systems can be managed to match service level agreements. Enterprise activity can be captured for compliance auditing.

EXPAND BUSINESS INTELLIGENCE

An ESB can provide new insights into enterprise activity. As the clearing house for all messaging, the ESB has a full view of enterprise activity. Business activity monitoring tracks key performance indicators for the business based on the messaging between enterprise systems. Collaborative reporting combines information from multiple sources into comprehensive aggregate reports. Business information can be accessed in real-time.

DECREASE COSTS

An ESB decreases an organization's operating costs. The agility and efficiency an ESB provides allows more to be accomplished with less staffing. Additional value can be created from existing software assets and existing technical skills by extending their reach. The time and cost for I.T. to develop, integrate or deploy solutions is reduced.

I.T. Benefits of an ESB

An ESB increases flexibility, lowers total cost of ownership, strengthens operational reliability, boosts manageability, makes I.T. roles more effective, improves the development process, and decreases risk.

INCREASE FLEXIBILITY

The ESB's event-driven architecture is both responsive and adaptive. Change management becomes simpler and more powerful. Communication and routing details are no longer embedded in application code and are easily reconfigured. Changes can be applied dynamically without disrupting systems in operation. Real-time integration replaces lengthy integration projects. The increased flexibility of an ESB can greatly accelerate mergers and acquisitions.

LOWER TOTAL COST OF OWNERSHIP

An ESB contributes to a lower TCO in several ways. Its long-term architecture avoids the need to periodically reinvent the enterprise. Additional value is created from existing software by extending its reach and maximizing reuse. Expensive development and integration projects become smaller or disappear entirely in favor of configuration changes. The use of open standards in an ESB makes it easier to find qualified I.T. and development staff.

I.T. Benefits of an ESB continued :

STRENGTHEN OPERATIONAL RELIABILITY

An ESB reinforces and improves operational reliability. The physically decentralized, service-based architecture is highly scalable. The logically centralized management promotes high availability, as does the freedom to apply configuration changes without taking down critical systems. Operational health and activity monitoring, throughput controls, and failover routing collectively help enforce service level agreements. Quality of service can be adjusted to provide desired throughput, security, and delivery assurances.

HEIGHTEN MANAGEABILITY

An ESB brings manageability to new heights. An enterprise-level view of management provides one-stop, unified configuration, monitoring, and control of services, messages, routing, security, and deployment. Enterprise-wide activity monitoring makes it possible to enforce SLAs, audit compliance, and enforce policies.

DECREASE RISK

An ESB decreases risk for the enterprise. It provides the freedom to select and combine best-of-breed products, and preserves the enterprise architecture when those products need to be changed out. The use of open standards in an ESB opens up a greater spectrum of compatible tools and systems and makes it easier to locate qualified personnel.

Making the Case for ESB

The ESB concept certainly seems real enough. Let's see what critics and analysts have to say.

RESPONDING TO ESB CRITICS

Critics tend to view ESBs as ill-defined; hype without substance; proprietary; overly centralized; short-term transitional solutions; another name for EIA; a departure from SOA; or simply unnecessary. Our examination of ESB demonstrates the opposite:

- While there are many ESB definitions, a composite definition shows more similarities than differences.
- While some ESB solutions may turn out to be hype without substance or involve proprietary solutions, not all of them are.
- It's misleading to call ESBs centralized; rather, they are logically centralized but physically decentralized.
- An ESB is not a short-term solution but a long-term architectural pattern; it may certainly leverage products but is not tied to any particular product.
- An ESB is not another name for EAI; rather, it is a convergence of EAI, MOM, and SOA.
- An ESB is not a departure from SOA; rather, an ESB is a service oriented architecture enabler.
- ESBs are necessary for enterprises that plan to take service orientation adoption to mature levels, or who require easy, low-cost change management.

WHAT ANALYSTS SAY

Analysts forecast a bright future for SOA adoption and cite the ESB as the right vehicle for getting there:

> "Forrester expects to see 67% of firms with 40,000 or more employees implementing SOA this year, and 44% of small and medium-size businesses (SMBs) already report that implementing SOA is a high or critical priority. Nearly 70% of users say they will increase their use of SOA, while only 1% of users will decrease their use. The data also shows that most firms — 83% — are using SOA for internal integration. This data reflects a perfect storm of market conditions to drive the growth of ESBs, which are the most straightforward way to get started with service-oriented integration today.
>
> "ESBs are also typically less costly than other ways of doing integration, such as integration-centric business process management suites (IC-BPMS), for three reasons:
>
> - Configuration is easier.
> - Standards support drives skills availability.
> - Costs are lower."
>
> —Forrester Research, 2006

In Part 2 of this article, we present Neuron ESB, Neudesic's architecture and framework for an ESB based on the Microsoft technology stack.

David Pallmann is an Architectural Consultant for Neudesic LLC, a Microsoft Gold Partner consulting firm headquartered in Southern California.