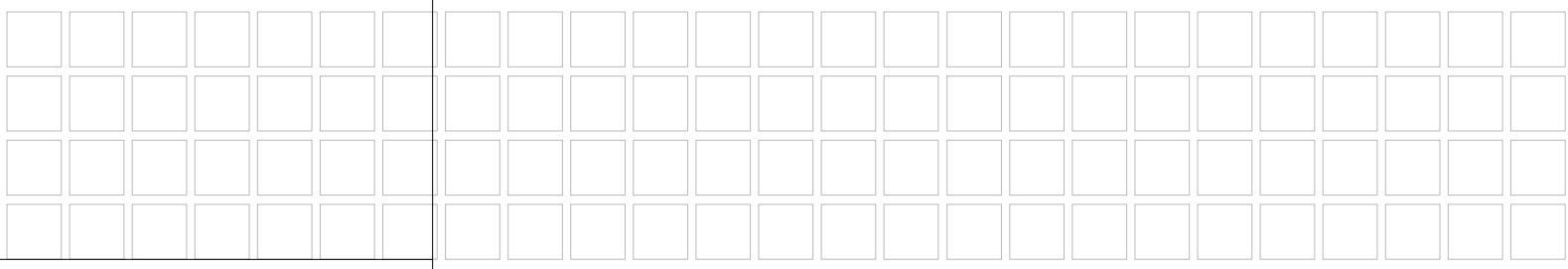
DISTRIBUTED SERVICE-ORIENTED ARCHITECTURES:

Delivering Standards-Based Integration

The Advantages of an Enterprise Service Bus



sonic SOFTWARE™

TABLE OF CONTENTS

ARCHITECTURE FOR A DISTRIBUTED WORLD

Distributed service-oriented architectures allow system architects to create a distributed environment in which any number of applications, regardless of geographical location, can interoperate seamlessly in a platform and language neutral manner. This claim has been made before with the introduction of COM, CORBA, and other distributed computing technologies, but the difference this time is the incorporation of proven, widely accepted standards into the integration model, as well as a focus on service-based interfaces. Use of these standards ensure that applications built and deployed within these architectures will all interoperate seamlessly using standardized business rules, and all speaking the same business language.

The conversion to a distributed architecture begins when new and/or existing applications are exposed for other systems to invoke. This is accomplished by converting proprietary business application components to new application business services based on a standardized XML integration format. Linking several services together can create new, aggregated services and distributed processes. As outlined in Figure 1, the first service-based applications will most likely coexist with existing applications using RPC-based communications.

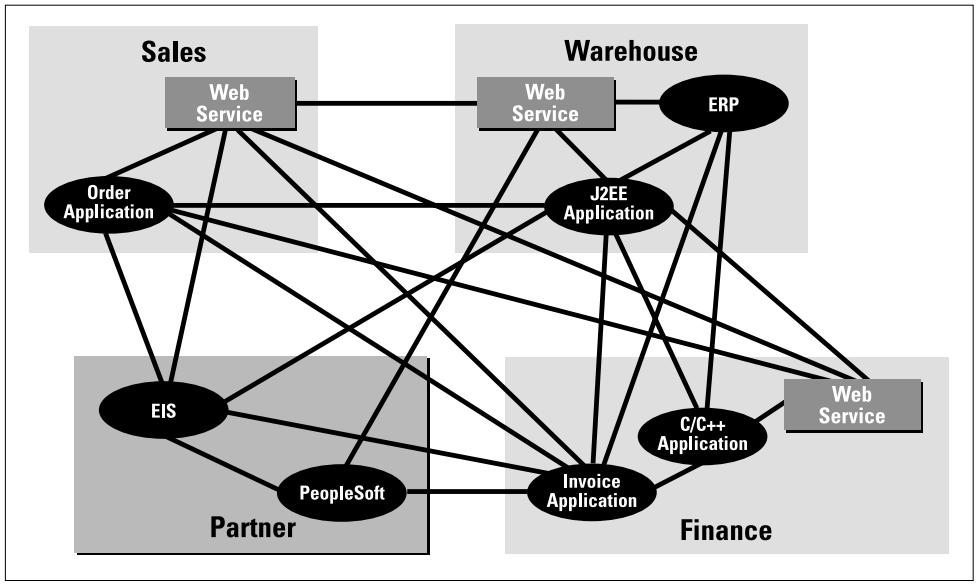


Figure 1. RPC-based distributed architecture with applications and services

Traditional distributed systems rely on a tight semantic coupling between applications and for all systems to be online and functioning flawlessly for any work to occur. If one application goes down or experiences intermittent disconnects the entire distributed system is at risk. Furthermore, if one or more applications need to be modified or relocated, it typically necessitates code changes to all other applications that depend on it. This legacy of tight coupling results in distributed systems that are too brittle and inflexible for the dynamic world of business. If the applications that reside on these systems are exposed in an RPC-like manner, they will fail to meet the requirements of a distributed service-oriented architecture, which are:

- **Loosely Coupled.** Senders and receivers (producers and consumers) are abstractly decoupled from each other. Senders construct self-contained messages (including but not restricted to XML documents) and send them using a standards-based communication backbone without knowing the actual location of the receiver. The communication mechanism is responsible for the abstraction between the sender and the receiver.
- **Coarse Grained.** Traditional object-based languages like Java expose their interfaces on a fine-grained, method basis. These fine-grained interfaces need to be aggregated into larger, coarse-grained services that closely resemble real business processes. Service technologies are designed to support the ability to assemble services into larger, more business suitable processes.
- **Asynchronous.** Asynchronous communication is essential for distributed architectures. Senders and receivers cannot always depend on the availability of each other to do their work. Through a system of intermediaries, messages are exchanged in real-time with high performance. Yet, when receivers are unavailable, the messages can be persisted for later delivery.

To successfully build and deploy a distributed service-oriented architecture, there are four primary aspects to be considered:

1. **Service enablement:** Expose each discrete application as a service
2. **Service orchestration:** Configure distributed services and orchestrate the services in a defined distributed process.
3. **Deployment:** Move from the test environment to the production environment, addressing security, reliability, and scalability concerns.
4. **Management:** Audit, maintain and reconfigure the services. Make changes in processes without rewriting the services or underlying application.

Services are created using application development tools (like Microsoft .NET, Borland JBuilder, or BEA WebLogic Workshop), which allow new or existing distributed applications to be exposed as Web services. Technologies like JCA may also be used to create services by integrating packaged applications (like ERP systems), which would then be exposed as services.

The most difficult problems in developing a distributed service-oriented architecture become apparent after the services have been developed. There are many factors to be considered when configuring, deploying and managing services and distributed processes.

- **Different integration formats.** There is often an impedance mismatch between the technologies used within internal systems and with external trading partner systems. In order to seamlessly integrate these disparate applications, there must be a way in which a request for information in one format can easily be transformed into a format expected by the called service.

- **Systems are sometimes unavailable.** Each organization has its own window in which it can conduct business. When distributed applications span multiple company divisions and partner systems, an unavailable service can significantly impact the organization's ability to conduct business. To be successful, you need to assume systems will be unavailable from time to time and build a distributed architecture that deals gracefully with this system downtime and intermittent connectivity failures.
- **Inherent lack of security.** There is a current industry-wide movement towards standardizing security solutions for Web services. However, until these technologies evolve, it is unclear how services will adhere to a number of security protocols that have been established to prevent fraudulent use of these services.
- **Scalability requirements.** With a distributed service-oriented architecture, there will be the need to scale some of the services or the entire infrastructure to meet demand. For example, transformation services are typically very resource intensive and may require multiple instances across two or more machines. At the same time, it is necessary to create an infrastructure that can support the hundreds-if not thousands-of nodes present in a global service network. Overall, a distributed architecture needs to be implemented which meets scalability requirements in a cost-effective manner.
- **Difficulty managing large networks.** A large distributed service-oriented architecture can consist of thousands of services and millions of messages sent between them on a daily basis. In addition to the scalability and security requirements mentioned, tools are required to deploy and manage all the services, endpoints and communications between them. These tools also need to have remote capabilities that enable review of the entire distributed architecture from a single location.
- **Managing real-world business processes.** A major problem is how to handle the interactions between the applications so they happen in a logical sequence that correlates to actual business processes. Some architectures rely on a single process controller (often residing within an integration broker) that coordinates distributed processes. In these architectures, every step in a distributed process is executed and the result returned to the central coordinator. At best, this effectively degrades the distributed process into a series of synchronized calls, and at worst, creates a single point of failure.

This paper will discuss the advantages for building distributed service-oriented architectures and identify the necessary components for the successful deployment and management of distributed processes, including Web services. The concept of an enterprise service bus is introduced as a solution to solve the common problems faced when configuring, deploying and managing a distributed service-oriented architecture.

ENTERPRISE SERVICE BUS

The enterprise service bus (ESB) addresses the challenges in assembling, deploying, and managing distributed service-oriented architectures. The ESB provides the distributed processing, standards-based integration, and enterprise-class backbone required by the extended enterprise.

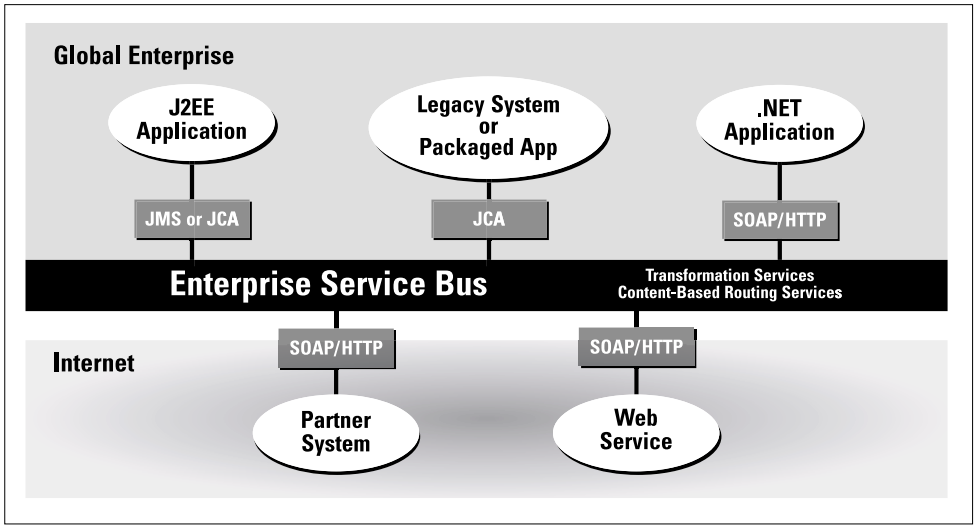


Figure 2. Enterprise Service Bus

The ESB is unique in that it provides containers or "docking stations" for hosting services. Services can be easily assembled and orchestrated in a standard framework. Once a service is deployed into a container, it becomes an integrated part of the ESB and is exposed for use by any other application or service. Physical network connections within the ESB are abstracted, so applications or services can be re-deployed in different configurations without having to rewrite the existing code. Furthermore, through a set of standards-based integration technologies, distributed processes can include legacy applications, can invoke external Web services, can be exposed as Web services, and in turn, Web services can invoke legacy applications.

The remainder of this paper discusses in more detail how an ESB solves integration problems and enables distributed service-oriented architectures.

To effectively orchestrate the behavior of services in a distributed process, the ESB infrastructure includes a distributed processing framework and XML-based services. To better illustrate these features, a typical distributed process as shown in Figure 3, will be configured and deployed using an ESB.

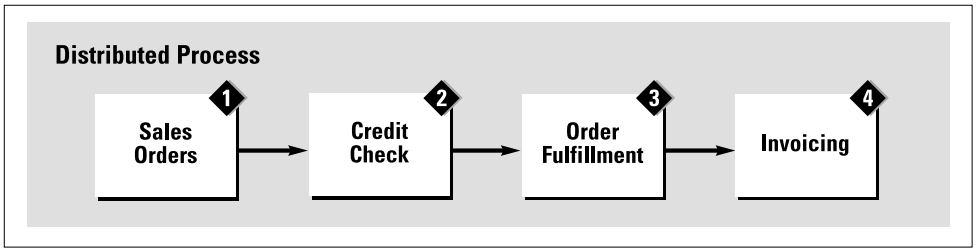


Figure 3. Typical distributed business process (Ordering)

In this example, the invocation of the ***Sales Order(1)*** procedure call initiates a series of events that are much more involved than the simple passing of a document from one machine to the other. Each of these events are created as services distributed across a network (including the Internet) that accept a message as input, process it, and result in another message, or a call to an external service. The purpose of the distributed processing framework is to effectively configure, deploy, audit and manage distributed services.

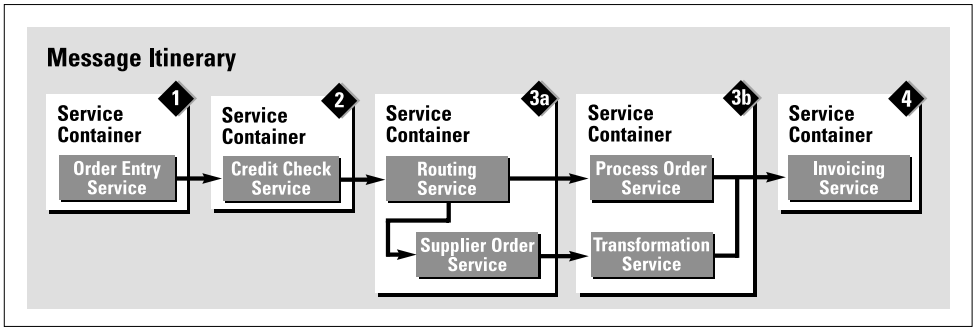


Figure 4. Message itinerary for the Ordering business process

At run time, each service executes within a ***service container***. A service container can host multiple services, even if they are not part of the same distributed process. It also drives messages through the services in a ***distributed process***. The distributed process uses message ***itineraries*** to control the flow of the message through a prescribed sequence of services, effectively orchestrating the process behavior. A ***process*** can include several services and they may be hosted in multiple containers, running on different systems throughout the extended enterprise.

Figure 4 illustrates how the services created for the distributed process described in Figure 3 are assembled through a message itinerary.

Messages enter a distributed process through entry points in the ESB, referred to as an ***endpoint***. Here an *itinerary* is added to the message so that movement through the process can be managed through a decentralized mechanism. Inside the distributed process, messages move through a series of services. When messages enter and exit their respective service containers, they do so through endpoints.

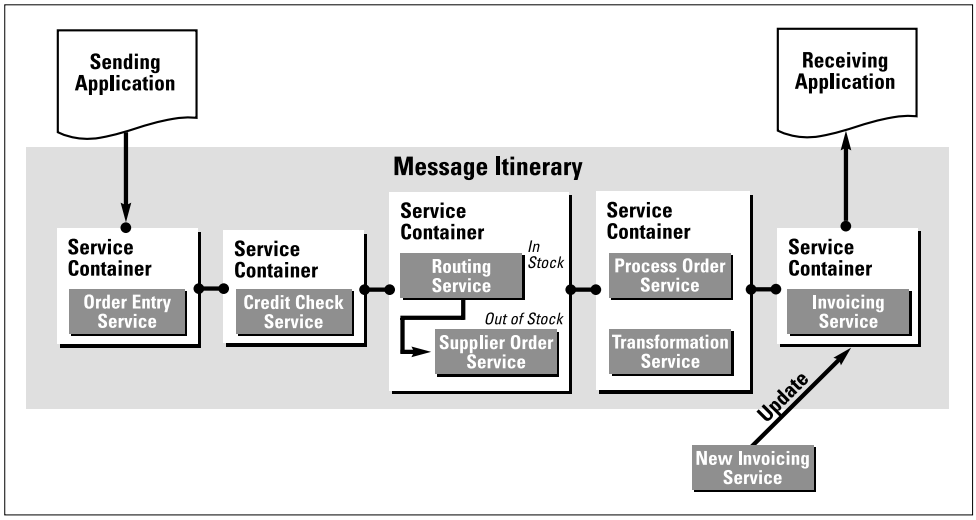


Figure 5. Message itinerary with endpoints

Endpoints provide abstraction of physical destination and connection information (like TCP/IP hostname and port number) that limit the integration capabilities of traditional, tightly-coupled, distributed software components. Endpoints allow services to communicate using logical connection names, which an ESB will map to actual physical network destinations at runtime. This destination independence gives the services that are part of the ESB the ability to be upgraded, moved, or replaced without having to modify code and disrupt existing business systems. Figure 5 illustrates how the Invoicing service can easily be updated with a New Invoicing service without disrupting other applications. Additionally, duplicate processes can be set up to handle fail-over if a service is not available. Endpoints also provide the asynchronous and highly reliable communication between service containers. The endpoints can be configured to use several levels of Quality of Service (QoS), which guarantee communication despite network failures and outages.

When the distributed process has completed its itinerary, the messages leave the distributed process, are addressed to one or more exit endpoints, and then a reply is sent to their respective destinations.

The ESB distributed processing infrastructure is aware of applications and services and makes intelligent decisions about their communications. This intelligence allows the ESB to order, route, and transform communications between members, so that communications can more closely resemble the real-world business flow. Some of the services that are part of the distributed processing infrastructure can be seen in use in Figure 4. Service container ***(3a)*** uses the ESB's content-based routing service to examine the incoming message and based on content and business rules, automatically routes the message to different services. For this example, if the inventory is ***In Stock***, the message is routed to the internal Process Order service; however, if the inventory is ***Out of Stock***, the message is routed to the Supplier Order service. The Supplier Order service, which executes a remote Web service at a supplier to fulfill the order, generates its output in an XML message format that is not expected by the next service (Invoicing service) in the itinerary. To avoid problems, the message from the Supplier Order service leverages the ESB's transformation service to convert the XML into a format that is acceptable by the Invoicing service.

Now that services have been chained together, it is necessary to provide a way to manage the services and reconfigure them to follow changes in business processes. Ideally, this is done through a central graphical console used to configure, deploy and manage services and endpoints. This allows the free movement and reconfiguration of services without having to rewrite or alter the services in any way.

STANDARDS-BASED INTEGRATION

The ESB supports both Web services and J2EE Connector Architecture (JCA) technologies, providing a standards-based architecture for integrating applications within the enterprise and across the Internet. This eases integrations between existing and future applications, including those built with J2EE and .NET in addition to packaged applications and legacy applications.

JCA

J2EE Connector Architecture (JCA) is an emerging technology that has been specifically designed to address the hardships of integrating applications. JCA provides a standardized method for integrating disparate applications in J2EE application architectures. Using JCA to access enterprise information systems is akin to using JDBC (Java Database Connectivity) to access a database. The latest versions of many application servers, including BEA WebLogic and IBM WebSphere, support JCA adapters for enterprise connectivity. In addition, major packaged application vendors have also announced plans to support JCA in future product offerings.

The ESB uses JCA to facilitate application integration between existing applications and services. The ESB provides a remoteable JCA container that allows packaged or legacy applications to be plugged into the ESB through JCA *resource adapters*. This connectivity is illustrated in the service containers ***(3)*** in Figure 6. In this example, the Process Order service uses JCA to talk to a J2EE application that internally fulfills incoming orders.

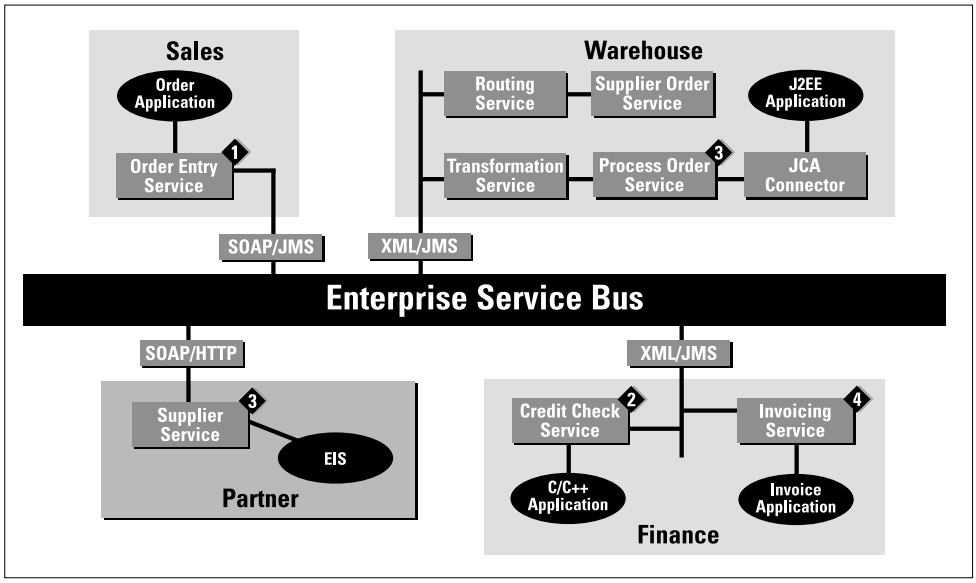


Figure 6. ESB with JCA and Web services

WEB SERVICES

Web services facilitate integration by using standardized XML data formats and communications protocols to connect service-based and traditional application components. Standards like HTTP(S), XML, and SOAP enjoy widespread acceptance as the optimal way to integrate applications within the enterprise and beyond. Emerging standards specific to Web services, like WSDL and UDDI, are prodigies of the open standards community and are quickly gaining widespread acceptance in the industry.

The ESB fully supports integration of services that are themselves Web services or that connect to other Web services. This can be seen in Figure 6, specifically with the Supplier service provided by a trading partner. Using the ESB, any service can use the bus to invoke services that are located anywhere on the bus or Web services that are geographically not part of the bus. This integration is accomplished using HTTP as the underlying network protocol, which transports XML-based messages (SOAP) to and from the bus.

Alternatively, Web services by themselves can interact with any other Web service without the presence of the ESB; however, this peer-to-peer interaction fails to meet the requirements of most enterprise environments where security, scalability, and reliability are of critical importance. The ESB incorporates an enterprise-class transport, which provides comprehensive security, asynchronous communications and the reliability to deal with service interruptions.

ENTERPRISE CLASS BACKBONE

The ultimate success of any distributed services architecture is not only based on the ability to integrate and reconfigure new and existing services, but also to have the high availability and security that distributed systems require. To achieve total success, services need to reside in an environment that neutralizes their inherent dependencies on problematic networks (like the Internet).

At the core of the ESB is an enterprise-class backbone, which ensures that services, processes, and applications connected to the ESB have end-to-end reliability, comprehensive security, and unsurpassed scalability.

The enterprise-class backbone provides:

> Guaranteed delivery of documents between services and endpoints.
> Scalability to meet the demands of large, multi-national enterprises.
> Security of services, Web services and access to enterprise systems.

RELIABLE DELIVERY

In order for geographically diverse services or Web services to work effectively, they need to be immune to disruptions caused by hardware, software, or network failures. To ensure that documents are successfully delivered between services and endpoints, the ESB provides transaction support and guaranteed service delivery on a 24x7x365 basis.

SCALABILITY

One potential area of failure in distributed service-oriented architectures is in the inability of the architecture to meet increases in demand for applications and services. Typical integration broker technologies handle scalability using a centralized hub and spoke model, i.e. they handle changes in load and configuration by increasing broker capacity or by adding brokers in a centralized location. The ESB uses a decentralized model providing complete flexibility in scaling any aspect of the integration network. A decentralized architecture enables independent scalability of individual services as well as the communications infrastructure itself. A key benefit of the decentralized model is the ability to apply fine, granular control over scalability where it is most effective.

For example, in Figure 7, to meet the demand of increased order fulfillment by outside suppliers, transformation services are added to the warehouse system. These additional services, running on additional machines, support the new scalability requirements without the need for replicating complete integration broker instances. In another example, the sales system increases throughput of the already existing Order Entry service by increasing the number of threads that the service can handle. Both methods allow capacity to be added where it is most needed - at the service itself. This differs from typical integration broker models, which add capacity solely through the use of additional homogeneous brokers.

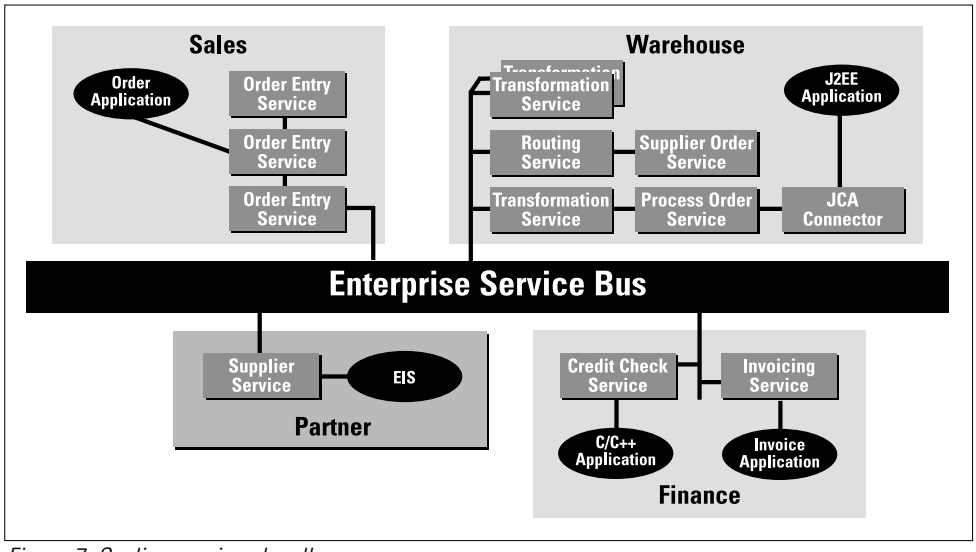


Figure 7. Scaling services locally

In some circumstances, it may be desirable to increase overall scalability by expanding the throughput capacity of the enterprise-class backbone. Figure 8 illustrates the broker functionality that is incorporated in the ESB. The ESB's use of brokers and broker clusters increase scalability by allowing brokers to communicate and dynamically distribute load on the bus. For example, in the event that an increase in the use of the warehouse services has overloaded the capacity of their host machine(s), new machines and new brokers can be added to handle the load -all without the need to change any of the services themselves.

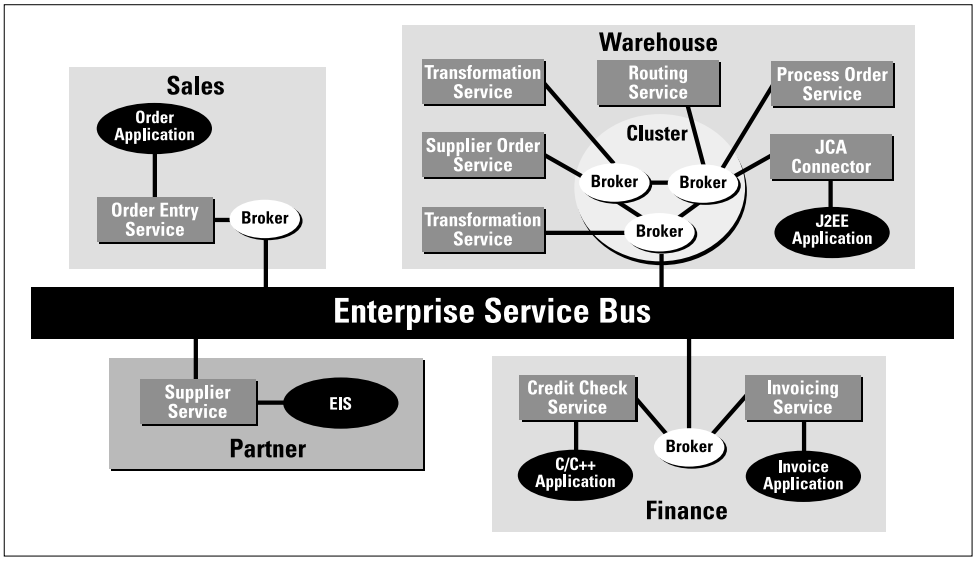


Figure 8. Scaling messaging infrastructure

These scalability options allow distributed service-oriented architectures to be deployed in geographically diverse environments that are seamlessly integrated and reliable. Once this architecture is deployed, the largest problem IT managers will face will be how to properly maintain and reconfigure the network. To manage this complex network, the ESB provides a set of remote management tools that have the ability to see all the applications and services connected to the ESB, regardless of their geographical location. This provides the ability to monitor communications on the ESB, look for scalability issues, and reconfigure any portion of the ESB from one central monitoring station -without bringing systems down.

SECURITY

The widespread acceptance of open-standards like HTTP, XML, SOAP, WSDL, and secure JMS, makes them an ideal foundation upon which to build Web services and the Web services community has quickly embraced them. Initially, services will make use of current technologies like firewalls, SSL and digital certificates to connect diverse locations that are either within the enterprise or located at trading partners. Next generation services will combine these proven technologies with new standards-based security technologies like XML encryption and XML digital certificates, which will provide more granular and optimized security measures.

The ESB contains a comprehensive set of security features that enable end-to-end secure communication between services. Support is provided for firewall-friendly communications, current and emerging encryption and authentication technologies (SSL, Digital Certificates), and application-based and third party authorization technologies. By using the ESB, architects will get the security features they require now, without having to wait for emerging XML-based security standards to be completed.

INTRODUCING SONIC ESB™

Sonic ESB is the world's first enterprise service bus, combining standards-based messaging, Web services, XML transformation and intelligent routing to reliably connect and coordinate the interaction of applications across enterprises, and between enterprises. Sonic ESB is designed to be cost effective and simple to deploy, yet capable of spanning a large number of diverse applications in highly-distributed environments. The ESB treats all applications as services regardless of how they are connected to the bus, allowing enterprises to incrementally move to a service-oriented architecture with minimal risk and reduced up-front investments. Its management framework allows you to easily configure and manage an entire integration network, no matter how large it becomes.

For more information on Sonic ESB, please visit our Web site at www.sonicsoftware.com, or contact your local sales representative.

Corporate and North American Headquarters

Sonic Software Corporation,14 Oak Park, Bedford, MA 01730 USA

Tel: 781-999-7000 Toll-free: 866-GET-SONIC Fax: 781-999-7202

EMEA Headquarters

Sonic Software (UK) Limited, 210 Bath Road, Slough, Berkshire SL1 3XE, United Kingdom

Tel: + 44 (0)1753 217000 Fax: + 44 (0)1753 217001

www.sonicsoftware.com