Enterprise Service Bus

By Nima Goudarzi – August 2007

Introduction

The ESB concept is a new approach to integration that can provide the underpinnings for a loosely coupled, highly distributed integration network that can scale beyond the limits of a hub-and-spoke EAI broker. An ESB is a standards-based integration platform that combines messaging, web services, data transformation, and intelligent routing to reliably connect and coordinate the interaction of significant numbers of diverse applications across extended enterprises with transactional integrity.

SOA in an Event-Driven Enterprise

In an event-driven enterprise, business events that affect the normal course of a business process can occur in any order and at any time. Applications that exchange data in automated business processes need to communicate with each other using an event-driven SOA to have the agility to react to changing business requirements. An SOA provides a business analyst or integration architect with a broad abstract view of applications and integration components to be dealt with as high-level services. In an ESB, applications and event-driven services are tied together in an SOA in a loosely coupled fashion, which allows them to operate independently from one another while still providing value to a broader business function.

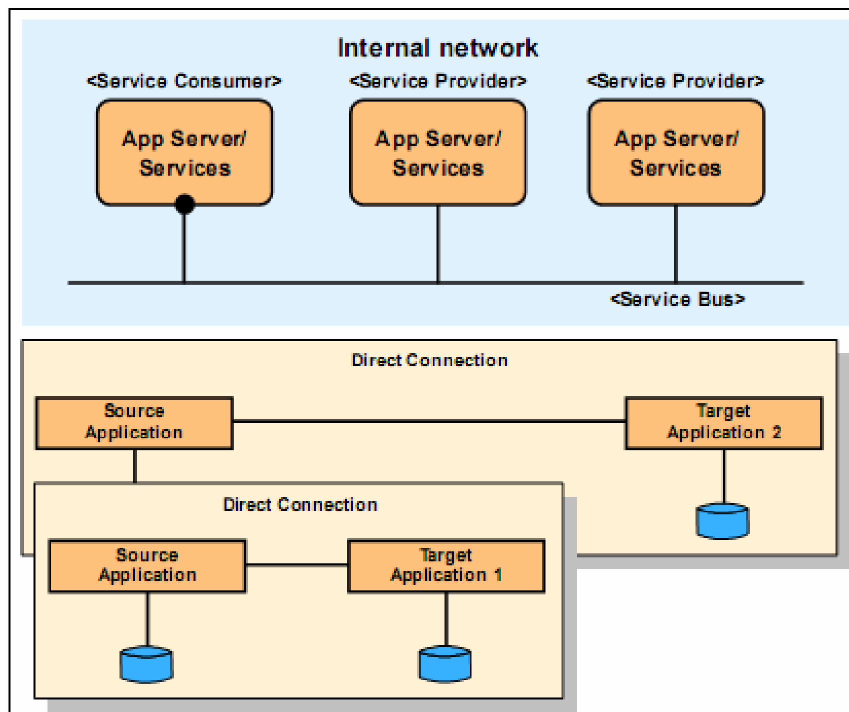
Service components in an SOA expose coarse-grained interfaces with the purpose of sharing data asynchronously between applications. Using an ESB, an integration architect pulls together applications and discrete integration components to create assemblies of services to form composite business processes, which in turn automate business functions in a real-time enterprise.

An ESB provides the implementation backbone for an SOA. That is, it provides a loosely coupled, event-driven SOA with a highly distributed universe of named routing destinations across a multi-protocol message bus. Applications (and integration components) in the ESB are abstractly decoupled from each other, and connect together through the bus as logical endpoints that are exposed as event-driven services.

ESB Runtime Patterns

Direct Connection using a service bus

The Direct Connection runtime pattern shows a service consumer that is connected to two other service providers via a simple service bus. The Application pattern overlays in this figure show that multiple Direct Connection application patterns can be deployed using the service bus.

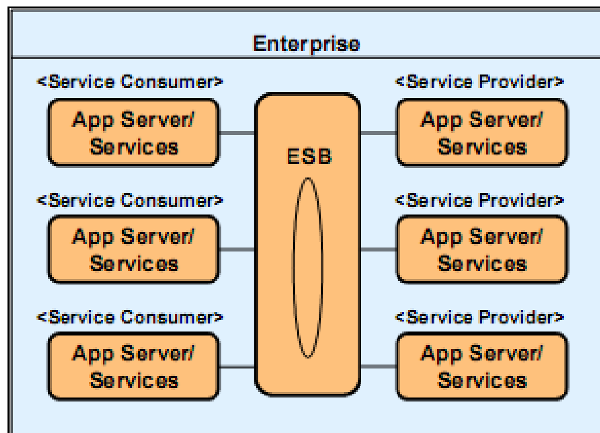


The service consumer (or source application) can use the service bus to initiate direct connections to two service providers — one to Target Application 1 and the other to Target Application 2.
The service bus concept is, however, an extension of the Direct Connection with federated adapter connectors runtime pattern that enables a set of connected Direct Connections.
The service bus approach:

- Minimizes the number of adapters required for each point-to-point connection to link service consumers to service providers.
- Improves reuse in multiple point-to-point scenarios.
- Addresses any technical and information model discrepancies among services.

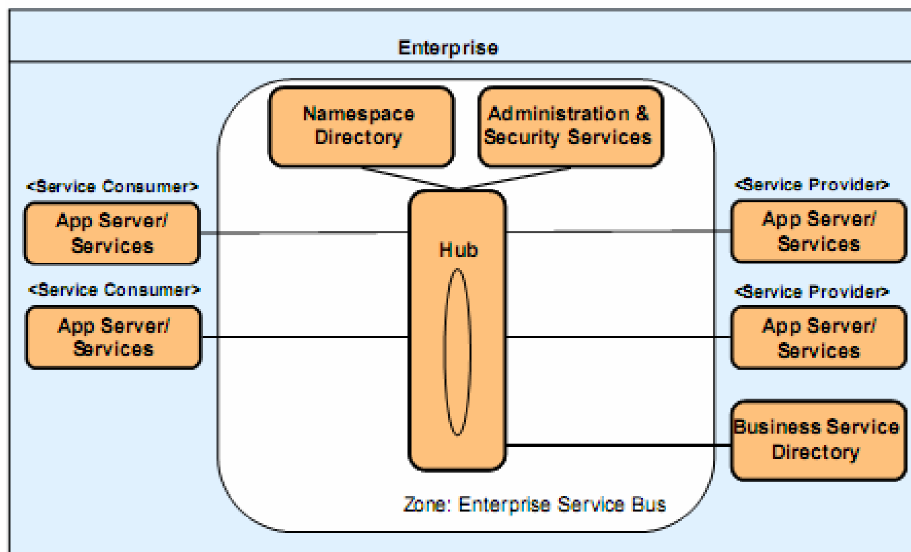
The service bus can span multiple system or application tiers, and can extend beyond the enterprise boundary. A rules repository node can also be included to model a service directory, allowing services to be discovered within and outside of the enterprise.

ESB runtime pattern



The ESB is a key enabler for an SOA because it provides the capability to route and transport service requests from the service consumer to the correct service provider. The ESB controls routing within the scope of a service namespace, indicated symbolically by the ellipse on the ESB node representation.

The true value of the ESB concept, however, is to enable the infrastructure for SOA in a way that reflects the needs of today's enterprise: to provide suitable service levels and manageability and to operate and integrate in a heterogeneous environment. Furthermore, the ESB must be centrally managed and administered and have the ability to be physically distributed.



This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

App server/services node

These nodes represent applications that request a service from the ESB or provide a service to the ESB. These applications can be implemented in any technology as long as they are able to interact using one of the protocols and messaging models that is supported by the ESB.

Services can be implemented in a variety of technologies and can be custom-developed, enterprise applications, such as those typically implemented in CICS Transaction Server, IMS Transaction Manager, and software packages.

Hub node

This node supports the key ESB functions and, therefore, fulfills a large part of the ESB capabilities. The hub has a fundamental service integration role and should be able to support various styles of interaction. There are two interaction styles that the hub supports. Those styles are the Router and Broker interaction patterns. The Router interaction pattern is where a request is routed to a single provider. The Broker interaction pattern supports requests that are routed to multiple providers, including aggregation and disaggregation of messages. The hub must contain rules for routing messages, and in the case of hubs that support the Broker interaction pattern, the rules must also describe how messages should be disaggregated or aggregated.

The minimum sets of functions that this node should support are:

- *Routing*
  This function removes the need for applications to know anything about the bus topology or its traversal. The interaction that a requester initiates is sent to one provider.
- *Addressing*
  Addressing complements routing to provide location transparency and support service substitution. Service addresses are transparent to the service consumer and can be transformed by the hub. The hub obtains the service address from the namespace directory.
- *Messaging styles*
  The hub should support at least one or more messaging styles. The most common are request/response, fire and forget, events, publish/subscribe, and synchronous and asynchronous messaging.
- *Transport protocols*
  The hub should support at least one transport that is or can be made widely available, such as HTTP/S. The hub can provide protocol transformation. If a protocol transformation is required that is not supported by the hub, then a specific connector can be used to perform the transformation.
- *Service interface definition*
  Services should have a formal definition, ideally in an industry-standard format, such as WSDL.
- *Service messaging model*
  The hub should support at least one model such as SOAP, XML, or a proprietary EAI model.

In addition to these capabilities, the hub can support more advanced capabilities, such as:

- *Integration*
  Additional integration services that can be provided include service mapping and data enrichment.
- *Quality of service*
  These services can include transaction management (for example, ACID properties, compensation, or WS-Transaction), various assured delivery paradigms (such as WS-ReliableMessaging), or support for Enterprise Application Integration middleware.

- *Message processing*
  The hub can support more advanced message processing capabilities such as encoded logic, content-based logic, message and data transformations, message/service aggregation and correlation, validation, intermediaries, object identity mapping, service/message aggregation, and store and forward.
- *Modeling*
  The hub can support more advanced modeling capabilities such as object modeling, common business object models, data format libraries, public versus private models for business-to-business integration, and development and deployment tooling.
- *Service level*
  Service level indicators might have to be measured, particularly in an enterprise mission-critical environment. The key indicators are availability and performance, which includes response time, throughput, and capacity.
- *Infrastructure intelligence*
  More advanced infrastructure capabilities can be provided. These include:
  –Business rules
  –Policy-driven behavior, particularly for service levels
  –Security and quality of service capabilities (WS-Policy).

Namespace directory
This node provides routing information in order for the hub to perform routing of service interactions. This node could be implemented as a routing table in the more simple implementations of an ESB.

Administration and security services
This section covers both administration and security services.

Administration
An ESB should be controlled by a single administration infrastructure. This node provides these administration services which, at a minimum, should support service addressing and naming.
The key services that must be provided by this node are:

- ESB configuration
- Service provisioning and registration
- Logging
- Metering
- Monitoring
- Integration with systems management and administration tooling

More advanced administration features that can be provided by this node include self-monitoring and self-management.

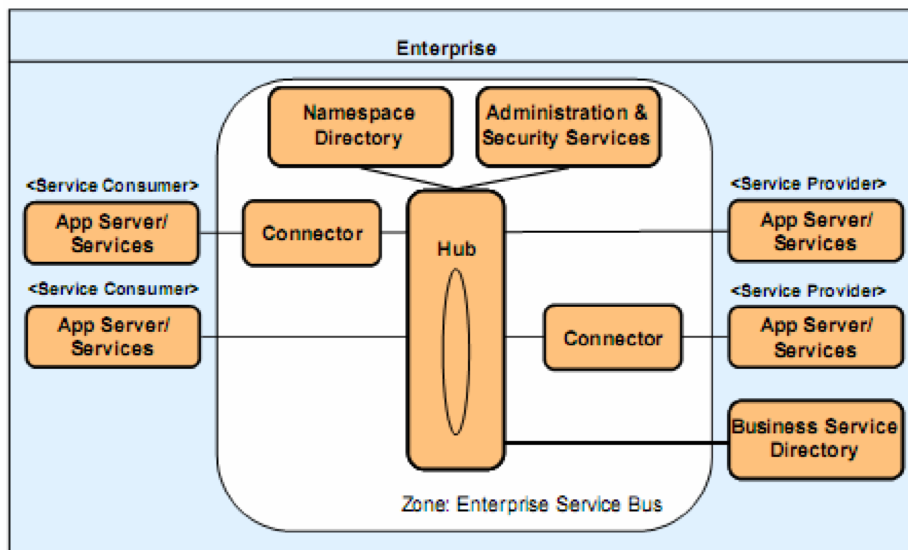
Security
In a mission-critical environment and, depending on the confidentiality, integrity, and availability requirements of the applications, the hub should support security capabilities such as authentication, authorization, non-repudiation, confidentiality, and security standards, such as Kerberos and WS-Security.

Business service directory

The role of the business service directory is to provide details of services that are available to perform business functions identified within a taxonomy. The business service directory can be implemented as an open-standard UDDI registry. More basic implementations can make use of an HTTP server. Catalogs, such as a UDDI registry, can achieve one of the primary goals of a business service directory: to publish the availability of services and encourage their reuse across the development activity of an enterprise. The vision of Web services defines an open-standard UDDI registry that enables the dynamic discovery and invocation of business services. However, although technologies mature toward that vision, more basic solutions are likely to be implemented in the near term.

Connectors

If we model the connectors that facilitate the interactions between service consumer/providers and the ESB, we find that we might require that some of these are both adapter connectors and path connectors, while other service consumer/providers need only a path connector to the ESB.
An adapter connector is concerned with enabling logical connectivity by bridging the gap between the context schema and protocols used by the source and target applications (in this case, between the service consumer/providers and the ESB).
A path connector is concerned with providing physical connectivity between source and target applications. It can be very complex (for example, the Internet) or very simple (an area of shared storage).



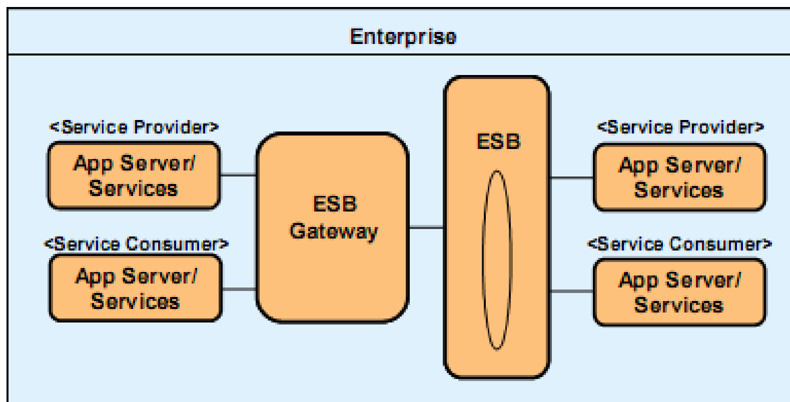
Adapter connectors facilitate integration in a heterogeneous environment with diverse technology, protocols, application types, and integration styles. Adapters perform the following key types of functions:

- Technology adaptation
  This type of adapter handles service consumers and providers that are built using technologies that are not natively supported by the hub. Examples of technologies that can be supported via adapters are CORBA, COM, JDBC, JMS, and EJB. Some of these technology adapters can use data handlers for particular data formats such as EDI, SOAP, XML, and various text formats.

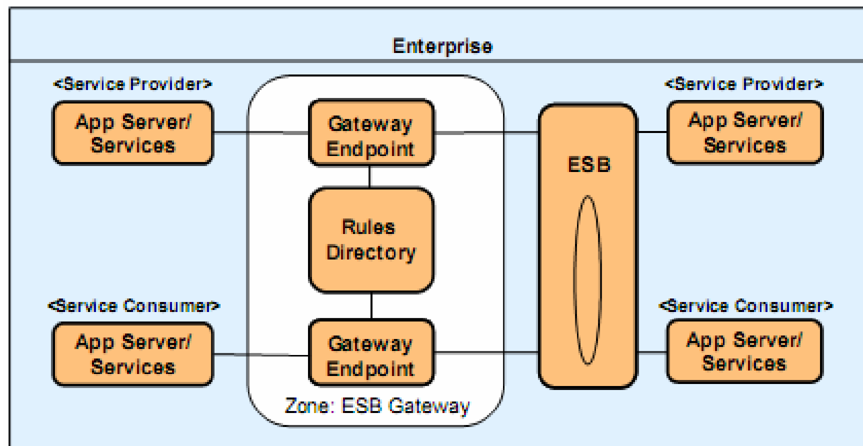
These adapters can also support different application server environments such as J2EE and .NET and different language interfaces such as Java, C, C++, and C#.

- Application adaptation
  This type of adapter facilitates integration with package solutions. Many examples of package solutions provide application adapters, such as Siebel, PeopleSoft, and SAP, among others.
- Legacy adaptation
  This type of adapter facilitates exposing valuable enterprise applications as services. These enterprise applications can be implemented using technologies such as CICS Transaction Server, IMS Transaction Manager and ADABAS amongst others.

ESB Gateway runtime pattern



The ESB Gateway acts as a proxy to provide controlled access to the ESB. A common use of the ESB Gateway is to expose services to external parties as well as allow internal applications to access external services in a secure and controlled manner.



This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

ESB

The ESB is a key enabler for an SOA because it routes and transports service requests from the service consumer to the correct service provider.

Rules directory

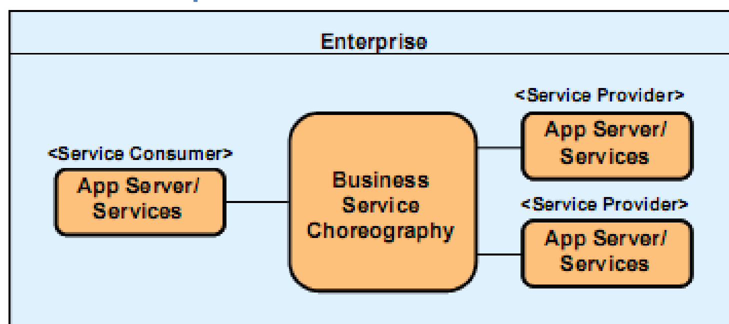
This node contains the necessary configuration information that the ESB Gateway needs to support secure and controlled access to services. The rules directory has configuration rules that can include mapping of service interface definitions to gateway endpoints, mapping of ESB gateway-provided service names to destination service names, and access control lists.

The configuration rules can also include information about service level policies to control throughput. These rules protect associated service implementations from operating beyond the established capacity levels.

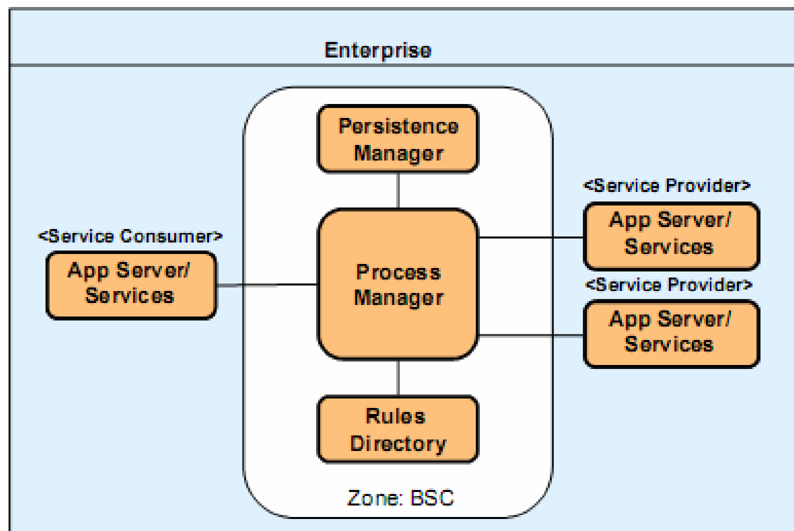
Gateway endpoint

This node is the entry point into services that the ESB provides or that are external to the ESB. It provides the address where messages are received, and it is mapped to particular protocols that the ESB Gateway supports (for example, HTTP/S). The gateway endpoint controls access to and from the ESB based on configuration rules that include access control lists and service level policies. It maps requests to the appropriate service and facilitates the interaction.

BSC runtime pattern



With the Business Service Choreography (BSC) runtime pattern, you can develop and execute business process flow logic that governs the sequence and control of service invocations. The business process is controlled centrally and is not part of the program logic in individual applications. Therefore, rather than having the business process defined in multiple applications and within the interactions between these multiple applications, the business process can be modeled and implemented by a central function. The Business Service Choreography facilitates the implementation of changes to the business process and monitoring and analysis of business process execution.



This basic topology leverages the nodes with their associated responsibilities as described in the following sections.

Process manager

This node contains the process flow execution engine. It provides the capability for model-driven business process automation. It also enables tracking by leveraging the process execution rules stored in the associated database.

These processes can span multiple applications and organizational boundaries within an enterprise. The node maintains state and tracks sequencing through the process flow. In doing so, it often leverages the persistence manager to store intermediate results. Finally, it invokes target services as necessary via the ESB.

The process manager node can support serial processes in which there is a sequential execution of process steps and parallel processes where process steps or subprocesses can execute concurrently.

The process manager should support the following key capabilities:

- Process definition standards, such as WS-BPEL, and the ability to execute process definitions that have been defined and exported from a modeling tool.
- Monitoring and analysis of processes by capturing information about process execution for historical analysis. It should also support integration with system management and administration tools.
- Ability to meet non-functional requirements such as performance, availability, and scalability will be important for mission-critical enterprise applications. Other key non-functional requirements are security and transaction management, particularly supporting the integrity and recovery of long-running business processes.
- Multiple levels of process abstractions.
- Correlation of events or incoming messages with existing process instances.
- Support for branching, parallel branch execution, and recomposing if the process manager supports parallel process execution.

Persistence manager

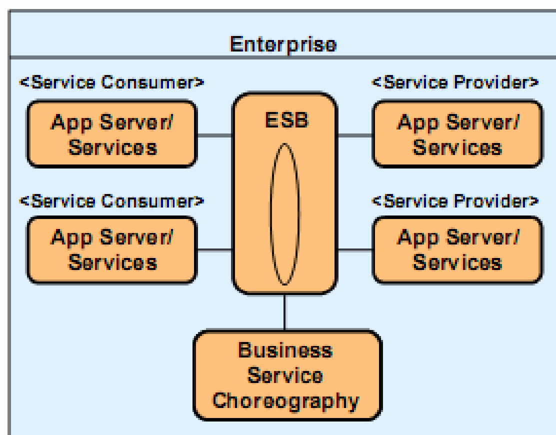
This node provides a persistent data storage service in support of the process flow execution. It holds results from the execution of certain activities within the context of an end-to-end process flow. These can be intermediate results that are valid within the context of a particular process flow and process data for the purpose of process monitoring and analysis. The intermediate results are necessary to support state management.

The implementation of this node typically involves a persistent data technology, such as a DBMS. In some cases, you can use non-persistent storage to store the intermediate results.

Rules directory

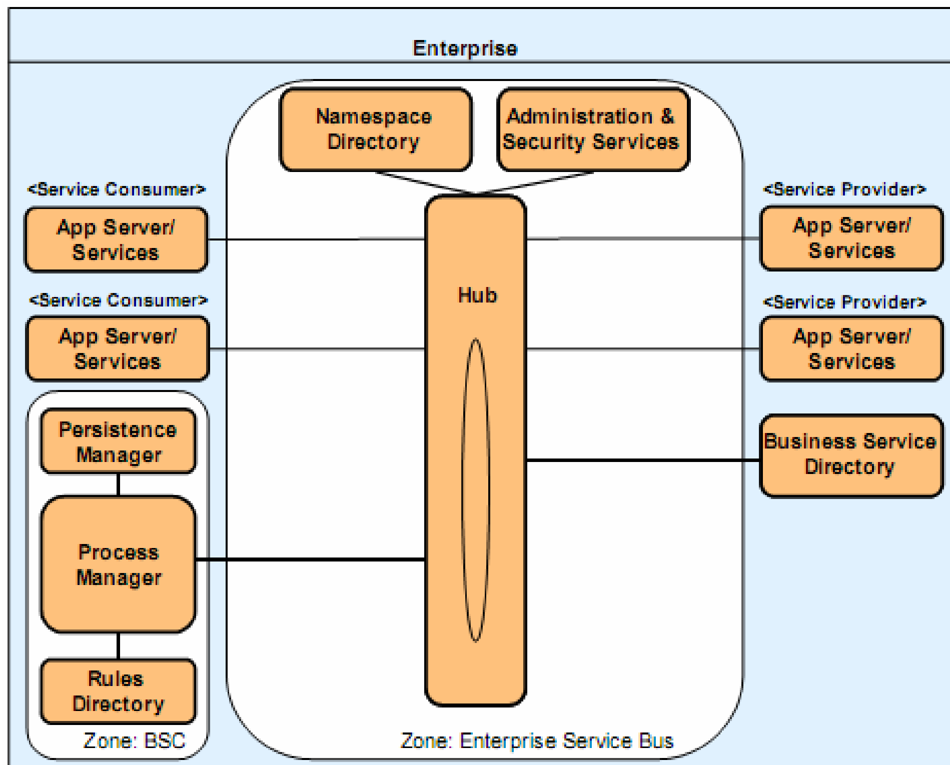
This node holds the read-only process execution rules in support of the process flow execution. These rules control the sequencing of activities and, therefore, support flow control within the context of an end-to-end process flow. The implementation of this node involves persistent data technologies, such as a flat file or a DBMS.

ESB, BSC composite pattern



The BSC node is implemented as a service consumer or service provider of the ESB. The BSC node is focused on process management function, and the ESB node provides the integration capabilities with other services. This pattern generally provides a loosely coupled and more functionally cohesive architecture where functional responsibility of nodes is clearly defined. The business process governs the sequence and control of service invocations that are mediated through the ESB.

The BSC has two core components, the process manager and repository nodes, that support the development and execution of business process flow logic. This logic is controlled centrally outside the application logic. Shielding the applications from the business process flow facilitates the implementation of changes to the business process and the monitoring and analysis of business process execution.



This section provides only a brief description of the ESB and BSC nodes.

BSC

This node is limited to process management and contains only the process manager, rules directory, and persistent manager nodes. The BSC relies on the ESB for integration and security functionality, and both receive requests from the ESB and send requests to the ESB via the hub node. The ESB can issue a request to the BSC to start execution of a process. The process execution will in turn most likely require services to be invoked as part of the process flow. Therefore, the BSC will request services from the ESB.

ESB

The ESB nodes are described in "ESB runtime pattern" on page34. As far as the ESB is concerned, the BSC is another application that can both request and provide services.

Java Business Integration (JBI)

Java Business Integration (JBI) is a specification developed under the Java Community Process (JCP) for an approach to implementing a service-oriented architecture (SOA). The JCP reference is JSR 208.

JBI is built on a Web Services model, and provides a pluggable architecture for a container that hosts service producer and consumer components. Services connect to the container via *binding components* (BC) or can be hosted inside the container as part of a *service engine* (SE). The services model used is Web Services Description Language 2.0. The central message delivery mechanism, the normalized message router (NMR), delivers normalized messages via one of four Message Exchange Patterns (MEPs), taken from WSDL 2.0:

- In-Only: A standard one-way messaging exchange where the consumer sends a message to the provider that provides only a status response.
- Robust In-Only: This pattern is for reliable one-way message exchanges. The consumer initiates with a message to which the provider responds with status. If the response is a status, the exchange is complete, but if the response is a fault, the consumer must respond with a status.
- In-Out: A standard two-way message exchange where the consumer initiates with a message, the provider responds with a message or fault and the consumer responds with a status.
- In Optional-Out: A standard two-way message exchange where the provider's response is optional.

To handle functionality that deals with installation, deployment, monitoring and lifecycle concerns amongst BCs and SEs, Java Management Extensions (JMX) is used. JBI defines standardized packaging for BCs and SEs, allowing components to be portable to any JBI implementation without modification.

JBI defines standard packaging for composite applications: applications that are composed of service consumers and providers. Individual service units are deployable to components; groups of components are gathered together into a service assembly. The service assembly includes metadata for "wiring" the service units together (associating service providers and consumers), as well as wiring service units to external services. This provides a simple mechanism for performing composite application assembly using services.

ESB Implementations

Open ESB

Open ESB implements a Java Business Integration (JBI) runtime that incorporates the JSR 208 specification for Java Business Integration and other open standards. Open ESB allows you to integrate web services and enterprise applications as loosely coupled composite applications, realizing the benefits of a service-oriented architecture (SOA).

Open ESB 2.0 Beta 2 includes the following:

JBI Runtime

The runtime includes the Java EE Service Engine and HTTP Binding Component

Open ESB Components

Open ESB supports pluggable service engines and communication protocol bindings as well as dynamic, configurable, message management and delivery. When installed with Java Application Platform SDK Update 3 Preview or NetBeans IDE 6.0 Preview with SOA (M9), Open ESB 2.0 Beta 2 includes the JBI Runtime and the service engines and binding components listed below. Developers can also create additional plug-in components to fit specific integration tasks.

Component/Feature	Description
JBI Framework	Runtime that implements a JBI instance. The JBI Runtime is a standard feature of Sun Java System Application Server 9.1 Beta 2.
BPEL Service Engine	Provides services for executing Web Services Business Process Execution Language 2.0 (WS-BPEL, or BPEL) compliant business processes.
Java EE Service Engine	Connects Java EE web services to JBI components.
XSLT Service Engine	Transforms XML documents using XSL style sheets.
Intelligent Event Processor Service Engine	Provides real-time business event collection and processing such as aggregation, filtering, and correlation, and performs event notification and event triggers.
SQL Service Engine	Provides SQL execution services to other JBI components.
File Binding Component	Provides a transport service to a file system and offers a comprehensive solution to interact with the file system from the JBI environment.
FTP Binding Component	Provides inbound and outbound JBI messaging using the FTP protocol as specified in RFC 959.
HTTP Binding Component	Provides external connectivity for SOAP over HTTP in a JBI 1.0 compliant environment.
JDBC Binding Component	Provides a comprehensive solution for configuring and connecting to databases that support the JDBC 3.0 API specification.
JMS Binding Component	Provides Java Messaging Service (JMS) transport for inbound and outbound messages.
SMTP Binding Component	Provides a comprehensive solution for configuring and connecting to SMTP servers and clients within a JBI environment.
WebSphere MQ Binding Component	Provides a comprehensive solution for configuring and connecting to WebSphere MQ servers within a JBI environment.

NetBeans IDE 6.0 with SOA Development Tools

A rich set of editors and tools for building and administering composite applications

Application Server Administration Tools

This includes extensions to the Admin Console, command line administration tools, and Ant administration tasks

Tools Available With NetBeans IDE 6.0 with SOA

The NetBeans IDE 6.0 with SOA includes visual design tools to make it easier to design applications, and also develop and maintain SOA applications. Here are highlights of some of the tools available.

- Graphical WSDL Editor
- XML Schema Tools, for XML schema creation, modification, and visualization
- XSLT Designer, for creating and editing XSLT transforms
- BPEL Editor, for BPEL-based web service orchestration
- Composite Application Service Assembly Editor (CASA), which provides a high level view of a composite application, allowing you to interactively specify service endpoints and configurations.
- Intelligent Event Process Editor, for configuring and monitoring event notifications.
- Composite Application Projects that allow you to create loosely coupled, service-based composite applications for SOA
- Various tutorials and blueprints that illustrate how to build and create composite applications.

Tools Available With Sun Java Application Server 9

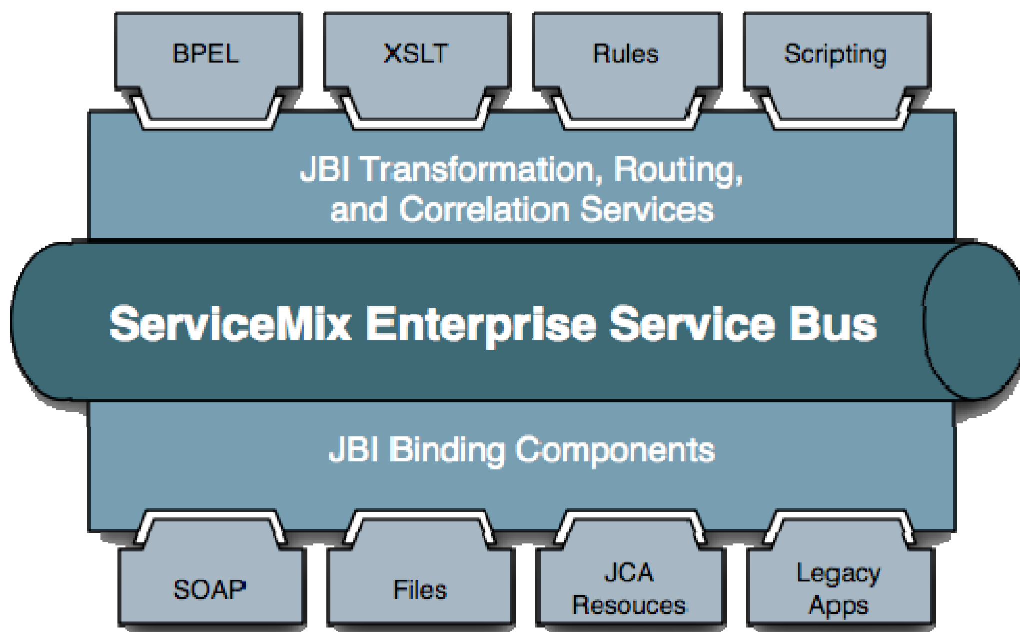
Sun Java Application Server 9 provides the following tools for administration of composite applications:

- JBI Ant Tasks, for administration using the Sun Java Application Server Ant client, asant.
- Command Line Interface (CLI), for administration using the Sun Java Application Server command-line processor, asadmin.
- Admin Console interface for JBI administration, which provides interactive screens for managing Open ESB components

Apache ServiceMix

Apache ServiceMix is an Open Source ESB (Enterprise Service Bus) that combines the functionality of a Service Oriented Architecture (SOA) and an Event Driven Arthitecture (EDA) to create an agile, enterprise ESB.

Apache ServiceMix is an open source distributed ESB built from the ground up on the Java Business Integration (JBI) specification JSR 208 and released under the Apache license. The goal of JBI is to allow components and services to be integrated in a vendor independent way, allowing users and vendors to plug and play.



Features

ServiceMix is lightweight and easily embeddable, has integrated Spring support and can be run at the edge of the network (inside a client or server), as a standalone ESB provider or as a service within another ESB. You can use ServiceMix in Java SE or a Java EE application server.

ServiceMix uses ActiveMQ to provide remoting, clustering, reliability and distributed failover.

ServiceMix is completely integrated into Apache Geronimo, which allows you to deploy JBI components and services directly into Geronimo. ServiceMix is being JBI certified as part of the Geronimo project.

JBI Container

ServiceMix includes a complete JBI container supporting all parts of the JBI specification including:

- Normalized Message Service and Router
- JBI Management MBeans
- Ant Tasks for management and installation of components
- Full support for the JBI deployment units with hot-deployment of JBI components

ServiceMix also provides a simple to use Client API for working with JBI components and services.

In addition, ServiceMix provides an implementation of WS Notification.

Standard JBI components

The following components ships with ServiceMix distribution:

- *servicemix-bean* is a JBI component for mapping beans (POJOs) to JBI message exchanges for easy procssing of JBI message exchanges. Note that if you want to support SOAP, type safe business interfaces or JAX-WS / JSR 181 then you should use servicemix-jsr181
- *servicemix-bpe* is a BPEL Service Engine written using Apache ODE Bpe
- *servicemix-eip* contains several routing patterns based on the great EIP book
- *servicemix-file* is a file system Binding Component
- *servicemix-ftp* is an FTP Binding Component
- *servicemix-http* is an HTTP/SOAP Binding Component
- *servicemix-lwcontainer* Service Engine can deploy lightweight components
- *servicemix-jsr181* Service Engine can expose annotated POJOs as services
- *servicemix-jms* is a JMS Binding Component
- *servicemix-wsn2005* is an implementation of WS-Notification
- *servicemix-xmpp* is an XMPP (Jabber) Binding Component

Mule

Mule in a Nutshell

- J2EE 1.4 Enterprise Service Bus (ESB) and Messaging broker.
- Pluggable connectivity such as JMS (1.0.2b and 1.1), VM (embedded), JDBC, TCP, UDP, multicast, http, servlet, SMTP, POP3, file, XMPP.
- JBI Integration.
- Orchestration of services using BPM and Mule components and routers.
- Support for asynchronous, synchronous and request-response event processing over any transport.
- Web Services using XFire (STaX-based) Axis or Glue.
- Flexible deployment Topologies including Client/Server, Peer-to-Peer, ESB and Enterprise Service Network.
- Declarative and Programmatic transaction support including XA support.
- End-to-End support for routing, transport and transformation of events.
- Spring framework Integration. Can be used as the ESB container and Mule can be easily embedded into Spring applications.
- Highly scalable enterprise server using the SEDA processing model.
- REST API to provide technology agnostic and language neutral web based access to Mule Events

- Powerful event routing based on patterns in the popular EIP book.
- Dynamic, declarative, content-based and rule-based routing options.
- Non-Intrusive approach. Any object can be managed by the ESB container.
- Powerful Application Integration framework.
- Fully extensible development model.

ChainBuilder ESB

ChainBuilder ESB is a Java Business Integration (JBI) compliant solution for use in Service Oriented Architecture (SOA) environments.

ChainBuilder ESB hides the complexities of JBI beneath a graphical user interface, allowing developers the full benefit of creating ESB components compliant to the open standard without becoming an expert in the specification.

Graphical Interfaces Simplify JBI

The core of the ChainBuilder ESB development environment includes several Eclipse-based plug-ins. These IDE graphical interfaces are used to create and customize ESB components through wizards and drag and drop functionality. Drag the component onto the ChainBuilder ESB canvas and an easy-to-use wizard guides the developer through selecting values for each property of the component. The ChainBuilder ESB Component Flow Editor hides the complexities of the JBI specifications, allowing developers to code to the JBI standard without becoming an expert in the mundane details of the specification.

References:

- O'Reilly - Enterprise Service Bus (2004)

- IBM Redbooks - Patterns Integrating ESB in a SOA

- IBM Redbooks – Implementing SOA Using ESB

- Wikipedia

- JSR 208 (JBI Specification)

- Open ESB official website

- Apache ServiceMix official website

- Mule official website

- ChainBuilder official website