

Inter-connected or unified ESB as SOA infrastructure

Abstract

When adopting an SOA, it is now common to use an infrastructure such as Enterprise Service Bus. There are, at least, two different ways of thinking about the infrastructure in an enterprise: use various ESB servers in the company, each one is used to solve a specific integration problem for a specific department, or use an ESB spread over the company, linking all parts of the Information System.

This article discusses several architecture and management topics for the two different approaches.

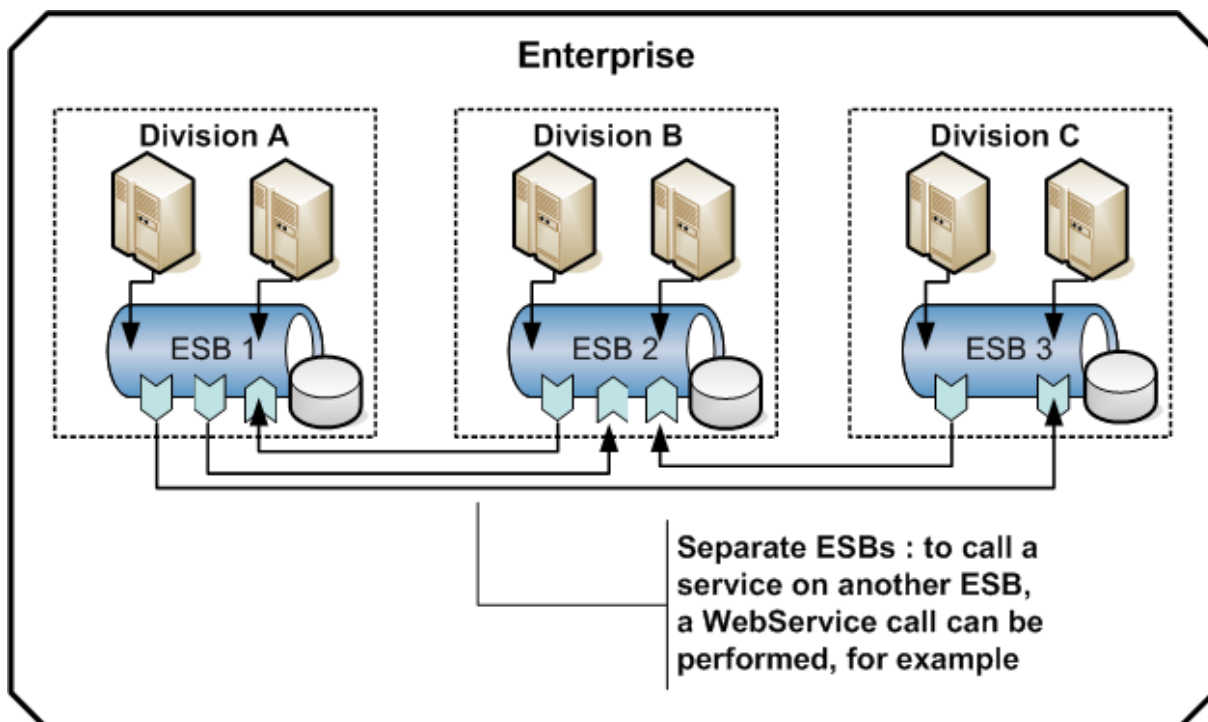
Different infrastructures

For 5 years, ESB is the buzzword for SOA foundation. This concept has been introduced by David Chappell, but today there is still no real definition. Indeed, each ESB vendor promotes its vision and there are different approaches, at the infrastructure level. Each approach has pros and cons.

A first approach is to let each department manages its SOA with its ESB implementation. It manages the integration of the application and the development of its business processes.

When a consumer application needs a business service which is in another department, the ESB has to call the required business service in a standard way (Web Service, FTP...) and can not access it directly.

By using separated ESB, each department is free to adopt the solution it wants. When a communication with other partners is needed, it punctually provides or accesses some business services by standard "bridge" like Web Services.

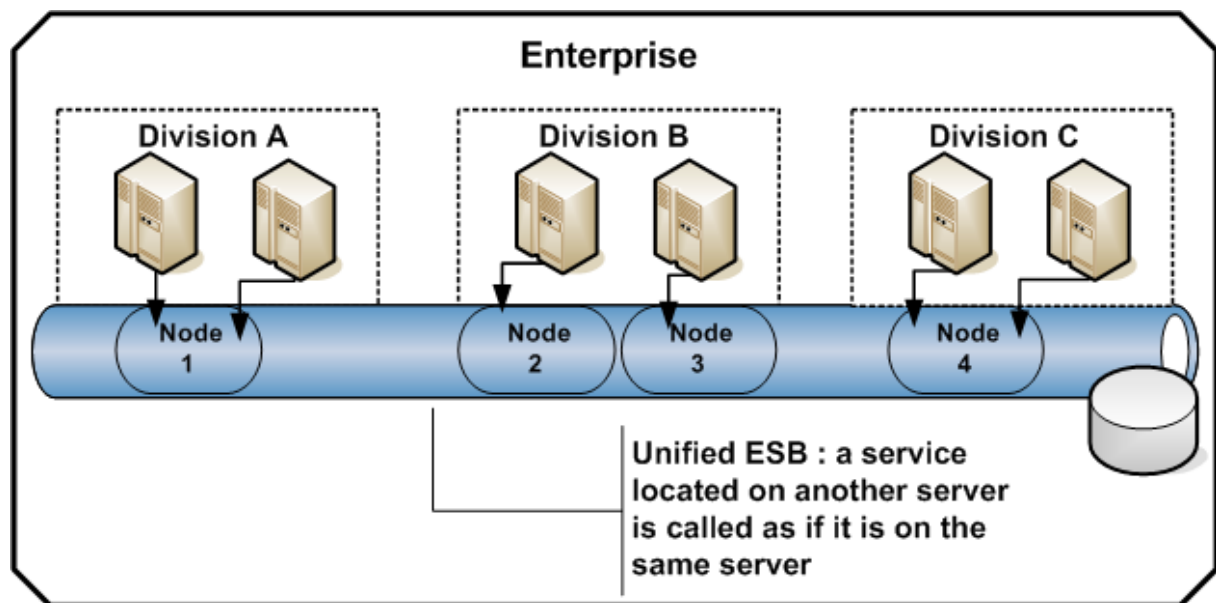


A second way to see the SOA infrastructure is an overall view, where the ESB is unified throughout the company. The ESB spreads (in a reasonable way) over the various servers of the departments, ensuring communication between them. Calling a service from another department is done in an easy way, since the consumer contacts the ESB the same way that it does for a local service.

In this view, the ESB is really seen as a backbone infrastructure, like an Ethernet network.

In more practical terms, the ESB is a set of natively inter-connected nodes; a node is a connecting point for services consumers or providers.

Note: An ESB node is in fact an ESB server that has internal network capabilities with other instances.



An ESB spread throughout a company network is not Big Brother

At first glance, a Chief Information Officer might say “Oh no, I don’t want this kind of ‘everywhere’ tool that allows everybody to see and access everything in the company”, administrators might claim “What can I manage or supervise?”, “Other administrators can change things on my servers, this is not possible”, or the well-known “What about the security?”, and so on...

Obviously, a generalized ESB does not allow everything. It just simplifies communication over the network and offers a set of quality of services throughout the network, in a simplified way.

Domains in a unified ESB

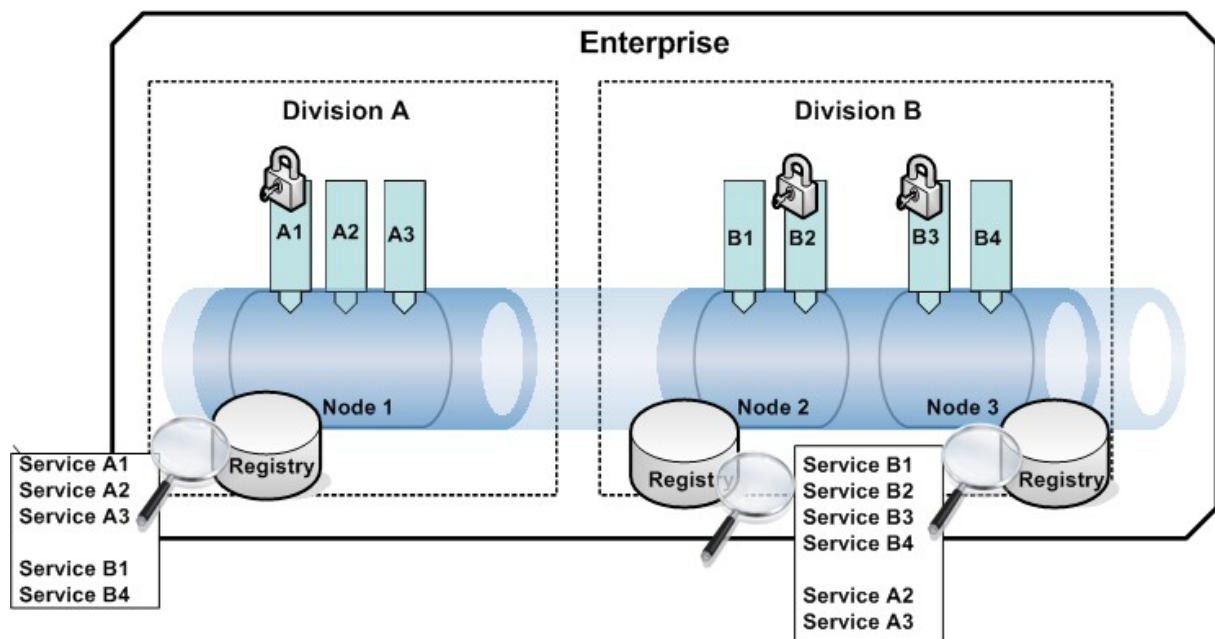
Domains and sub-domains define borders between entities. Thus, from an administrative point of view, each “node” of the ESB is only manageable by the sub-domain administrator. He or she is the only person that can start, stop, install connectors, deploy services and so on.

The administrator manages the ports used by the ESB nodes of his domain and can set proxies or firewalls to protect these ports.

A business service deployed on the node of a domain can be public or private. That is to say, a private service is only visible, in the registry, by consumers which are in the same domain. Moreover, the private service is only accessible for the same domain consumers.

Note: These private services will not be discussed in the following sections, unless specified otherwise.

The service and process monitoring can only display information of the domain or public information of the others’ domains. It is not possible to see other private domain processes or service statistics.



Now that we have cleaned up any potential misunderstandings about what a unified ESB is, let’s take a look at some of its advantages.

Registry

In a disconnected ESB environment, a service from another domain exists only if a bridge is explicitly created for this service between the two domains’ ESBs. For example, a service from one domain is exported as a classic Web Service and another ESB has to know the URL to call it. A company-wide registry is needed to reference all these Web Services. In this case, administrators of each domain have to publish the domain’s Web Services to this company

registry. These Web Services can be considered as the “public” services of the company, as they are visible by all.

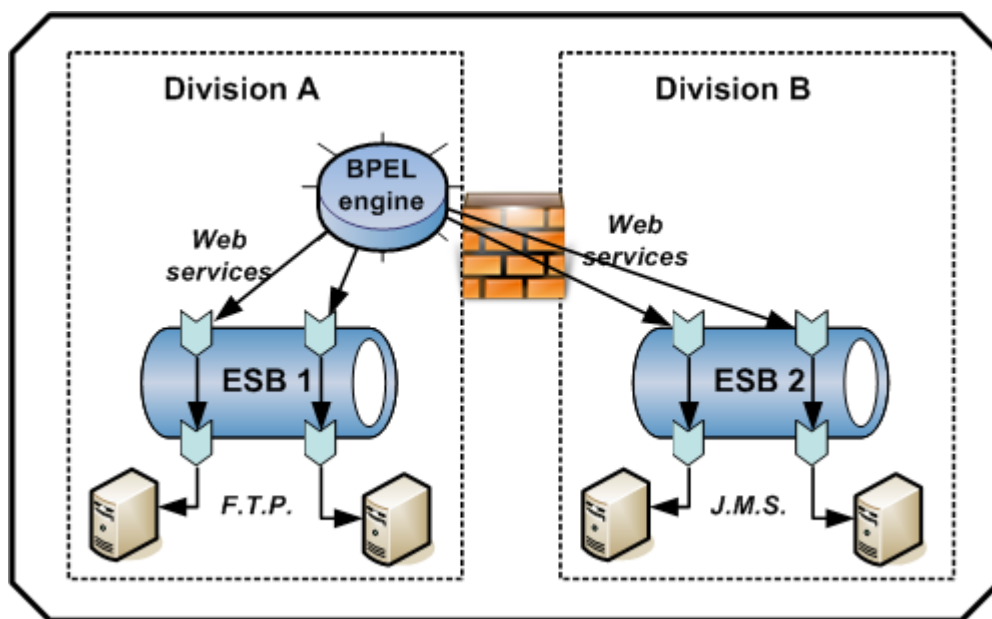
In a unified ESB, the business services registry is itself unified; the registry of each ESB instance is synchronized with the other ones. A consumer or a service browser that connects to an ESB in a domain can see all the services on the bus. When a new service is exposed on the ESB, it is immediately seen and accessible for every consumer on each ESB instance, despite its physical localisation. All services are accessible in the same way; they are “ESB services”. No matter their implementation or their protocol.

Orchestration

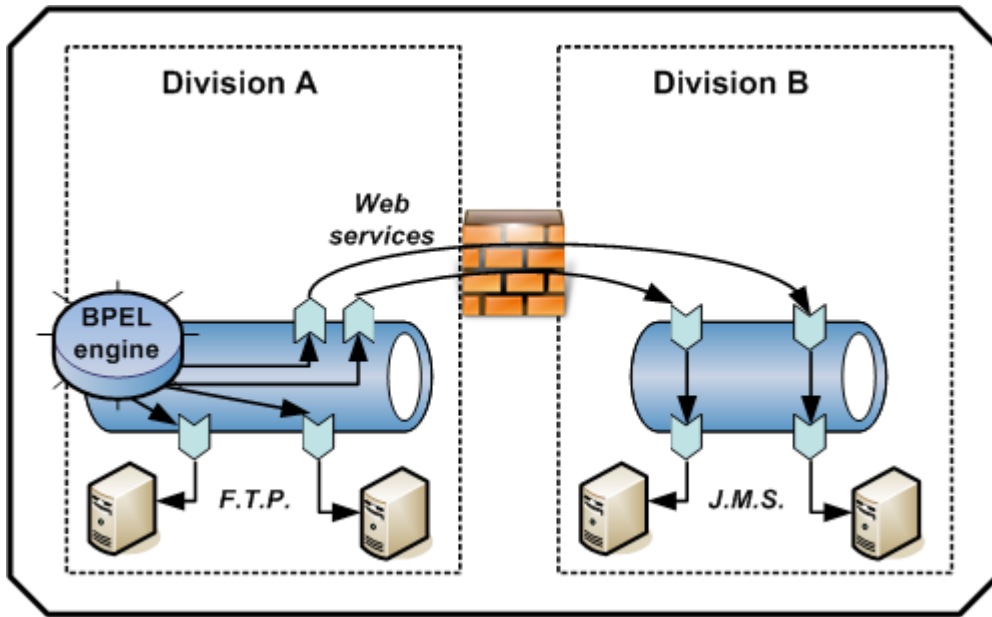
To orchestrate services spread over multiple disconnected ESBs, there are two ways of conceptualizing the orchestration.

First, the BPEL engine is exterior to the ESB environment, for example it could be a Web Services BPEL engine. In this way, the process designer needs to know all the services across the domains. It would be also necessary to integrate these services as Web Services to access them.

At runtime, the BPEL engine has to make some http connections over the global network to access services.

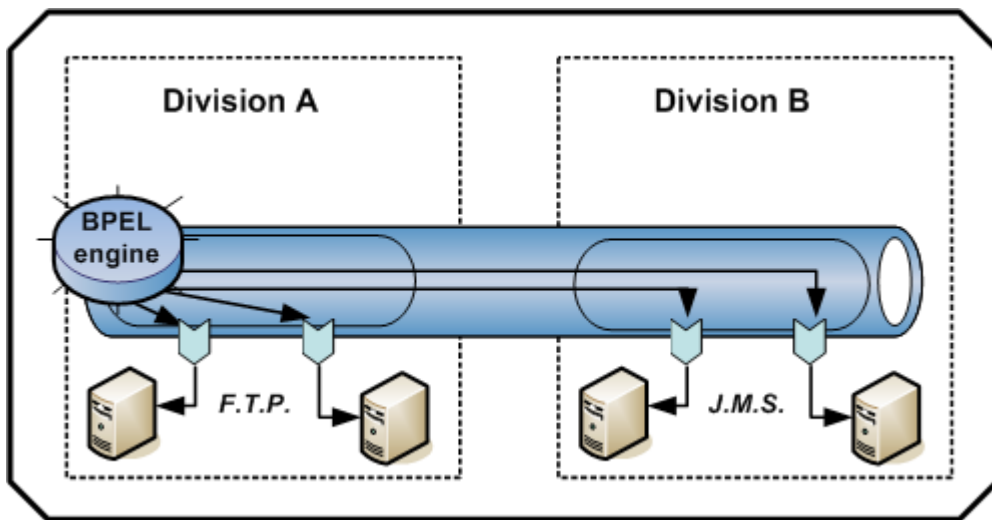


On the other hand, the BPEL engine could also be integrated in an ESB, calling services that are available in the local environment. When it has to access a service which is on another ESB, a bridge (like a Web Service call) must be performed to access the other ESB and then the service.



With a unified ESB, the “internal” registry contains all the services of the bus and their description. It is easy to design processes by connecting the BPEL designer to the ESB registry.

The design of the process only deals with the ESB services. All the available services can be retrieved from the unified registry and directly integrated into the process. At runtime, transaction or compensation problems are easier to take into account as there are no bridges between the BPEL engine and the services.

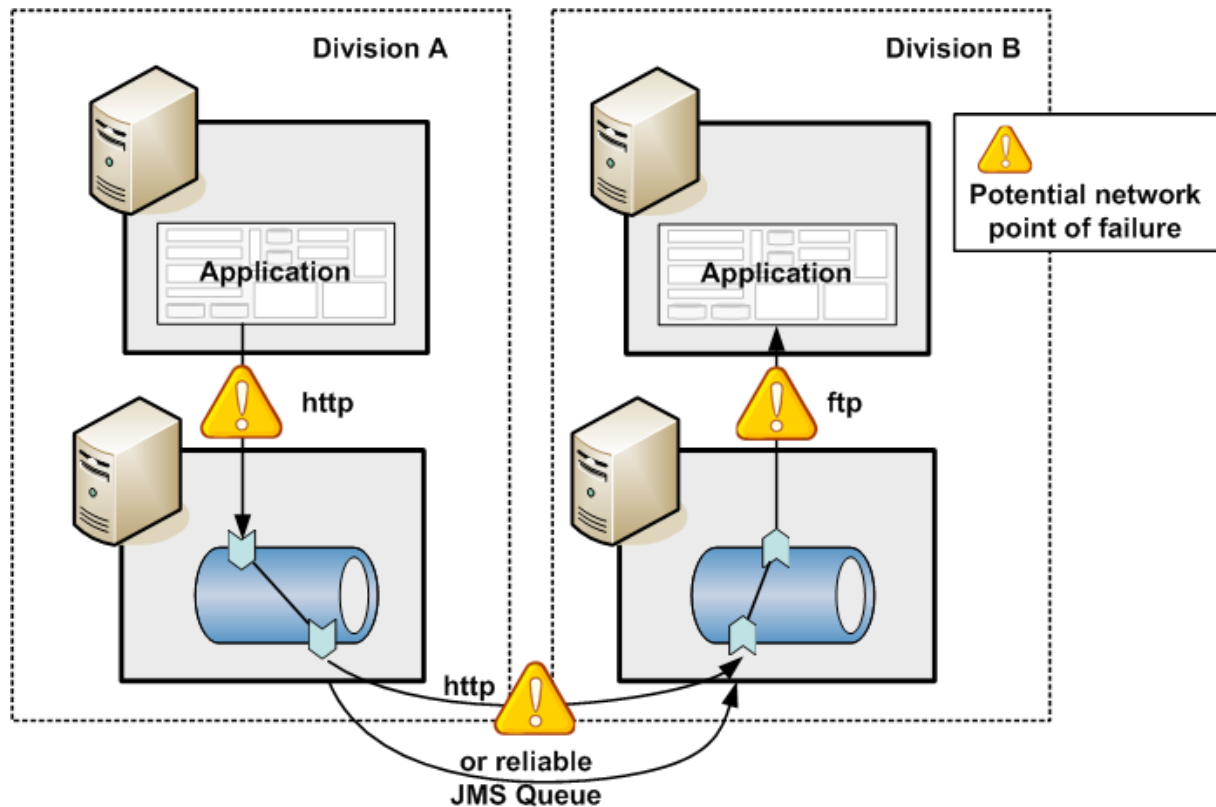


Reliability

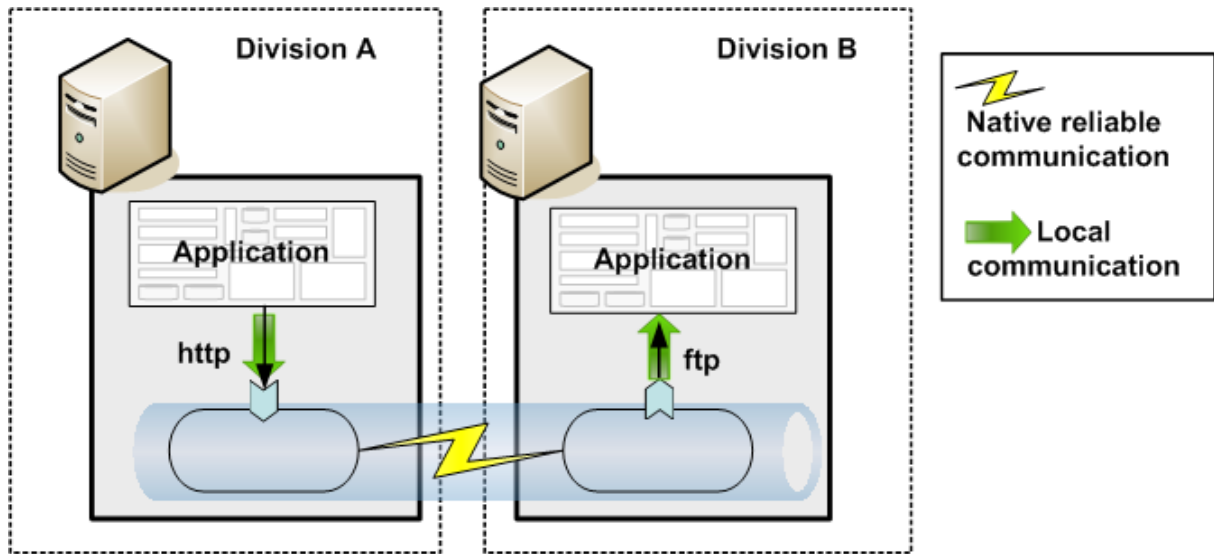
In an integration environment like a SOA, there is, in essence, a lot of communication between applications. They are spread over different servers and information is largely conveyed through the network. The transport layer of the infrastructure needs to be as strong

as possible to avoid information loss. The ESB transport layer often offers message reliability, but the bridge between the ESB and the application relies on the protocol capabilities (such as Web Service, FTP, JMS, RMI ...).

With disconnected ESB instances, the reliability needs to be ensured between instances too. *For each service* located on another ESB instance, a reliable communication has to be configured. A JMS Queue per service can be used to do so. Over that, the communication between the sender ESB, the JMS Queue and the receiver ESB needs to be transactional. The work becomes hard when there are a lot of other domain's services to access.



Using a unified ESB allows installing one ESB instance on each physical server that hosts an application to integrate. In this case, a Web Service call or an FTP connection still remains on the server. A network breakdown will not affect the communication between the application and the ESB node. The network communication is achieved by the ESB nodes. The reliability is guaranteed by the transport layer of the nodes (generally based on a JMS MOM).

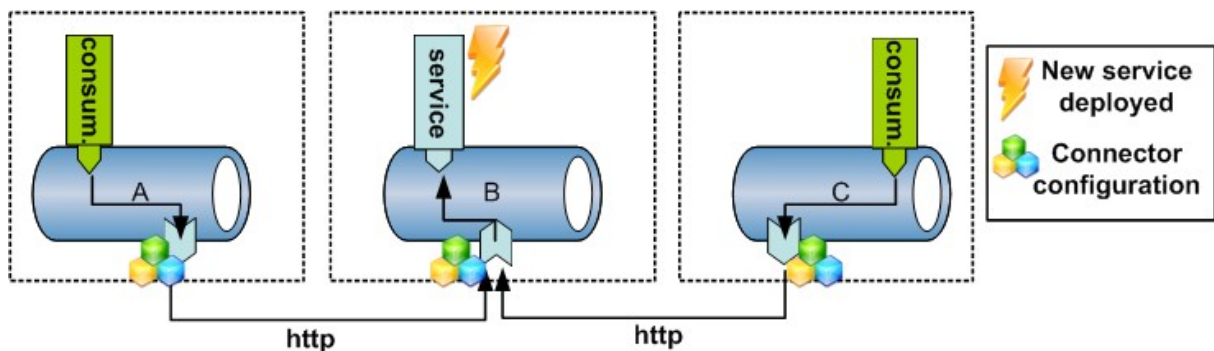


Note: installing an ESB node on each server hosting an application is an ideal vision. Applications that do not need a perfect reliability can connect to a distant ESB node.

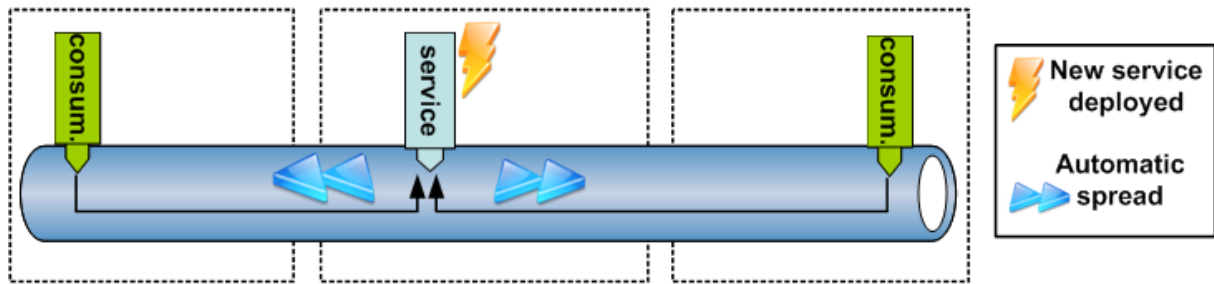
Services governance

Managing services lifecycle or versions is not easy in a disconnected environment. When a new service is exposed in an ESB instance, connectors have to be configured to expose it on other instances in order to allow consumers from other instances to access it. When a new version of this service becomes available, each connector has to be updated to be consistent with this new version.

Moreover, if there is no company-wide registry that references all ESB instances services, consumers from other ESB instances will never know that some services exists on an instance.



In a unified ESB, as each instance is natively interconnected, a service which is deployed on an instance is immediately available on the other ones. This service is visible in the registry of each instance. When this service change, each consumer has benefit from this new version.



Administration

Most of the time, the administrator of a disconnected ESB environment (whose performing connectors installation, services deployment and other lifecycle management) has to connect to each ESB administrative console of the domain. The administrator configures separately the ESB instances behaviours.

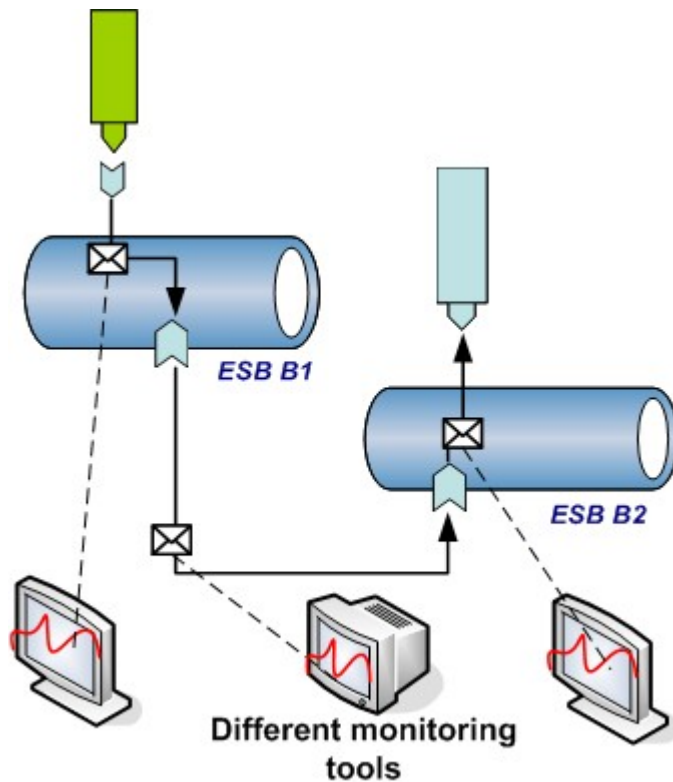
Using a unified ESB allows the administrator of the domain to manage all the nodes (of the domain) in a single administrative console where he looks at all the topology. The main benefit is that the administrator watches all instances activity (resource consumption, services calls statistics...) and can manage the behaviour of the ESB in the domain.

For instance, if a service such as an XSL transformation is very used, the administrator can instantiate a copy of this transformation service, deploy it to another instance, and define a load balancing rule to dispatch the requests. Such capabilities can not (or difficultly) be done in a disconnected ESB environment.

Business Monitoring

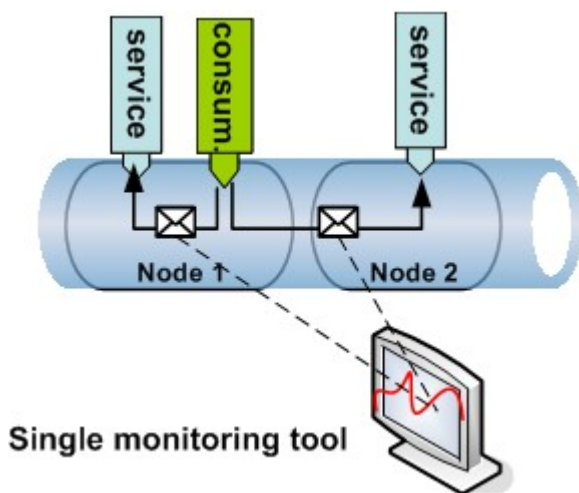
Business activity monitoring is one of the most interesting features that an SOA infrastructure gives to operations managers. Service statistics are available in real time; business processes are monitored and can optimized, alerts can be thrown by emails when anomalies occur...

With disconnected ESB instances, gathering key performance indicators (KPI) is not easy. KPI from each ESB instance have to be collected separately and the communication between two instance (a Web Service call for instance) is not easy to monitor. All the data have to be correlated before being readable by the monitoring tool.



Once again, a unified ESB simplify the KPI collect through the domain. As there is no protocol break between instances, the lifecycle of an exchange can be monitored from the consumer to the service provider even if there are not hosted by the same ESB instance.

The monitoring console connects to any instance of the domain and can display the domain statistics.



Conclusion

Each ESB offers its set of functionalities and tools to integrate applications. With a set of connectors and services like transformation or orchestration, applications integration has become easy.

For a given integration problem, one can use an ESB, some connectors and make the glue between 2 or 3 applications. This “point-to-point” approach is easy and quick to set up, and does not require business service definition (WSDL and other) most of the time.

This “integration” view is sufficient in this case, but is not enough in a real “Services Platform” approach, where the ESB has to be the SOA backbone of the enterprise.

Choosing an ESB implementation that allows a real network communication is a major key for a “Service Platform” infrastructure adoption in the enterprise. It allows a global view of all the activity and business of the Information System and enhances its agility.

About the author

Adrien LOUIS works as chief architect at [EBM WebSourcing](#), and has 8 years of experience working with Java EE technologies. Currently, he is working on enterprise integration solution problems. Adrien LOUIS is SOA consultant and the architect of the [PEtALS](#) open source project. PEtALS provides a leading open source ESB to support SOA.

It is a lightweight, highly distributed and scalable platform for both A2A and B2B integration.

Thanks to its specific distributed architecture and the tools provided, PEtALS offers a very competitive integration solution with support of a large number of protocols, formats and integration features.