

Service-Centric vs. Message-Centric ESBs

A critical comparison of two ESB approaches



Service-Centric vs. Message-Centric ESBs (July 2005)

Copyright © 2005 Cape Clear Software, including this documentation, all demonstrations, and all software. All rights reserved. The document is not intended for production and is furnished "as is" without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

Trademarks

Cape Clear is a registered trademark of Cape Clear Software. Cape Clear Software, Cape Clear Studio, Cape Clear Server, Cape Clear Data Transformer, Cape Clear Orchestrator, and Cape Clear Manager are trademarks of Cape Clear Software in the United States and other countries.

All other company, product, and service names mentioned in this product may be trademarks or service marks of others.

CPV-DOC-3066

Contents

- Perspectives on the ESB 4
- The Message-Centric ESB 5
- The Service-Centric ESB 8
- Conclusion10

Perspectives on the ESB

The approach of vendors to ESB architecture has very significant implications for user organizations. The type of ESB chosen dictates the reach of the integration platform, the developer productivity that can be achieved, the ease of administration, and the kinds of integrated application that can be developed. User organizations must understand the underlying architecture of the ESB in order to make an informed vendor selection.

The ESB market can be segregated into message-centric products, which emphasize the transport capabilities of the offering, and service-centric products, which accentuate service development and deployment capabilities.

Message-centric vendors such as Sonic and Fiorano have taken to packaging their messaging implementation as an ESB. The strength of these products lies in the level of capability offered for messaging Quality of Service (QoS), asynchronous, and pub/sub interaction styles.

By contrast, service-centric platforms such as that offered by Cape Clear tend to be messaging-agnostic, enabling the user organization to reuse existing transport infrastructure and messaging layers. Service-centric products emphasize capabilities that facilitate the creation and hosting of standards-based services.

According to Gartner's definition, an ESB is standards-based middleware that uses a Service-Oriented Architecture (SOA) and that has messaging, intelligent routing, and transformation capabilities. Some industry experts validly extend that definition to include features like orchestration, security federation, and a common management framework.

With so many components to the ESB it is possible for vendors in this space to emphasize the things that their products do best. This phenomenon results in ESB products with significantly divergent capabilities and architectures. Let's take a look at how these respective solutions differ.

The Message-Centric ESB

Message-Oriented Middleware (MOM) delivers essential capabilities to enterprise IT architectures. At its heart is a message broker that provides asynchronous message delivery, enforcement of reliability, transactional guarantees, and support for pub/sub and broadcast-type interaction styles.

Because SOAP is just another message payload, MOM vendors recognized the opportunity to combine the interoperability of Web services with the messaging capabilities of MOM, so the message-centric ESB was born.

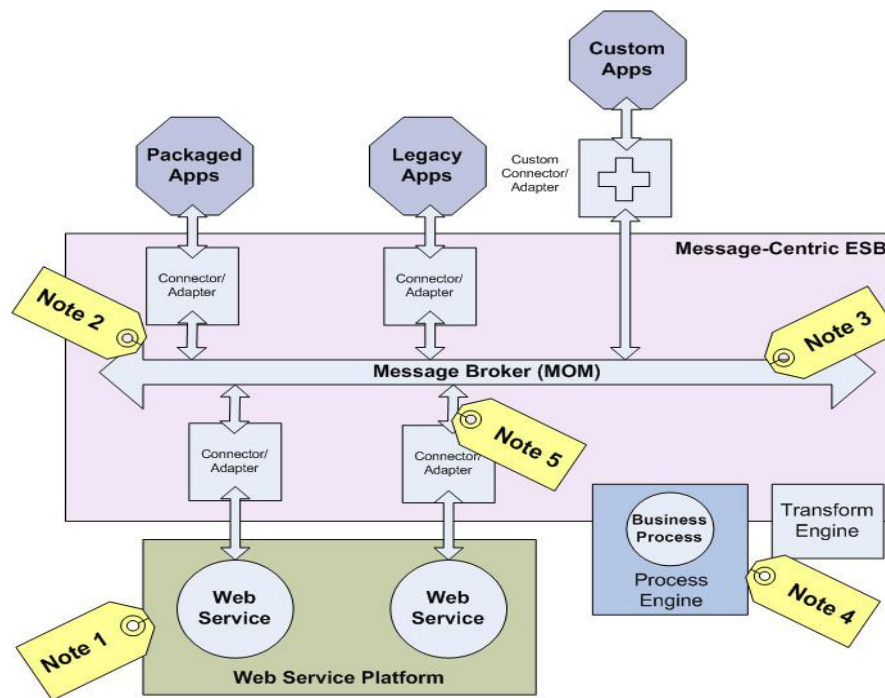


Figure 1: Architecture of the message-centric ESB

Figure 1 is a high-level conceptual architecture of a message-centric ESB. Consider the suitability of this architecture for the implementation of an ESB:

Note 1: Message-Centric ESBs Don't Support Service Creation or Hosting

Message-centric ESBs merely provide an infrastructure that you can use to access services that you have already built in a separate platform. Many vendors, such as Sonic, ship Apache AXIS out of the box. However, the underlying assumption is that there is a separate, scalable, and compatible service creation and hosting environment available onto which you deploy the Web services that form your SOA. This in turn means that SOA development and SOA management are now distributed over two platforms instead of one. The result is a disjointed and less productive environment for developers and administrators alike.

Note 2: Message-Centric ESBs Can Only Communicate in a Managed Way, through a Single MOM Transport

In message-centric ESBs, all communications must be routed via a specific MOM transport in order to provide managed communication and policy enforcement. This means that applications that already communicate over different transports (for example, other MOM implementations, FTP, e-mail, and so on) must either be modified to be connected to the ESB or must communicate via a bridge or gateway. This may be suitable if these interactions require additional QoS offered by the new transport that are not provided by other existing transports, however, this is not often the case.

Essentially, message-centric ESBs are guilty of tying communications policy (that is, reliability requirements, auditing requirements, and so on) to the transport implementation, when in fact these policies apply across multiple transport channels. The requirement to deploy a new transport in order to achieve the benefits of an ESB is an unnecessary imposition. It constrains communications between services and adds cost and complexity to SOA development, deployment, and administration.

Note 3: Message-Centric ESBs Often Require Duplication of Messaging Infrastructure

Most organizations have already deployed MOM, however, vendors such as Sonic and Fiorano require that their MOM implementation be installed as a prerequisite to their ESB. This causes an unnecessary and costly duplication of messaging infrastructure in the enterprise, which results in complexity from an architectural and development perspective. Perhaps even more significant is the additional administration and support needed in these duplicate environments. It should be possible for the ESB implementation to reuse existing message infrastructure (if required), rather than requiring a second MOM implementation.

Note 4: Bus Capabilities Are Not Exposed as Services

In the message-centric ESB, the capabilities offered by the bus (for example, transforms, routes, and processes) are separate to the business services deployed in the SOA (for example, order processing, account inquiry, and so on). These bus capabilities must be deployed as part of the message broker or, in many cases, as separate server capabilities in the configuration. These ESB capabilities are not defined or deployed as reusable services in the SOA. This means that they must be separately managed, secured, and developed with resulting implications for manageability, productivity, and maintenance.

Note 5: Message-Centric MOMs Provide Low-Level APIs to the Bus

While differing in terms of the implementation specifics, performance, management, and proprietary extensions, all MOM vendors expose a message-oriented programming model. This model requires the developer to deal with entities such as queues, topics, destinations, sessions, connections, and so on. It provides various ways to read and write

messages to and from a bus. The bus then delivers the messages reliably, using the required interaction style. From this, we learn that the focus of the developer is on configuring the bus, placing messages on the bus, reading messages off the bus, and so on. Many MOM implementations conform to the Java Message Service (JMS) specification, which defines just such a programming model. There is no higher-level abstraction of business function or provision of a model for delivering coarse-grained services. This abstraction must be provided via a separate services platform and then integrated with the message-centric ESB.

The Service-Centric ESB

A service orientation is about representing business logic as a set of standalone, self-describing, and interoperable entities. Following are two important characteristics of services in a SOA:

- **Loosely coupled:** Loose coupling implies that services can communicate over multiple transports transparently, hence the ESB must support multiple transports and messaging infrastructures.
- **Coarse-grained:** Services provide a less granular and more business-centric interface definition, which makes them less complex, more flexible, and easier to reuse. Rather than working on low-level APIs to get messages on and off the bus, the developer encodes business interfaces, rules, and logic.

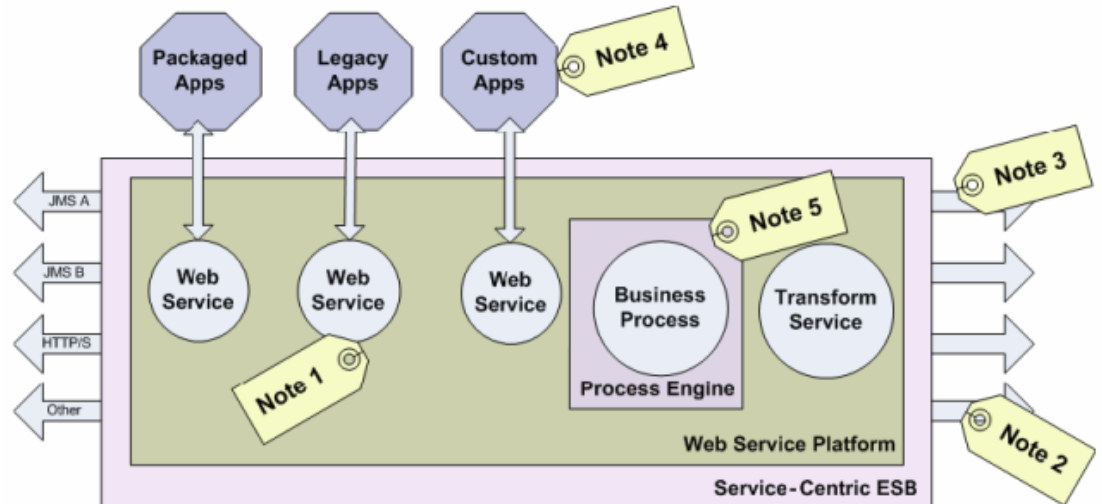


Figure 2: Architecture of the service-centric ESB

Figure 2 depicts the conceptual architecture of a service-centric ESB. When compared to a message-centric ESB, this architecture delivers the following characteristics:

Note 1: Service-Centric ESBs Support Web Service Creation, Deployment, and Management

In a service-centric ESB, Web services are hosted in the ESB and can be developed using tools provided by the ESB. This means that SOA development and management are supported in a single platform, resulting in a more cohesive and productive environment for developers and administrators.

Note 2: Service-Centric ESBs Support Multi-Channel Communication

Services communicate in a transport-agnostic way by specifying the endpoint of the target service. The ESB then routes this communication via the appropriate transport to the target service. This approach is also advocated by the Java Business Integration (JBI) standard (JSR 208), which separates service-layer communications from transport-layer communications. The advantage of this approach is obvious: the service implementation is not constrained to use a specific MOM implementation and is free to communicate in a managed way over transports such as HTTP, FTP, WS-RM, and e-mail where appropriate.

Note 3: Service-Centric ESBs Can Reuse Existing Messaging Infrastructure

Because service-centric ESBs are transport-agnostic, they can reuse existing enterprise messaging infrastructure for communication if necessary. This protects existing investments and eliminates the need for duplication of infrastructure.

Note 4: Bus Capabilities Are Deployed as Reusable Services

In the service-centric ESB, the capabilities offered by the bus (for example, data transformation, routing, and orchestration) are Web services that are deployed in the services platform. These services are deployed and managed in the same way as the business services offered in the SOA. This means that all services—those offered natively by the ESB and those deployed on the ESB—can use a common infrastructure for deployment and management.

This issue is particularly pertinent when you consider business processes. Processes are high-level services that are implemented by combining the capabilities of other services and sub-processes. If services and processes are deployed in separate containers, the developer must build and debug integrated processes across two different platforms. In addition, the administrator must manage and monitor two different solutions for information relating to process state and performance. This is not an acceptable architecture and again is one of the drivers behind the JBI (JSR 208) specification. The service-centric approach delivers a unified platform today for creating and hosting services and processes.

Note 5: Service-Centric ESBs Focus on Programming in the Large

SOA is successful only if it achieves abstraction from the underlying infrastructure and enables user organizations to build robust and reusable business services that are easily used both internally and by customers and partners. This abstraction requires a development model that encourages developers to design services from the top down, and then to map these business-centric interfaces to low-level technical interfaces. This approach is known as meet-in-the-middle service design. Service-centric ESBs support this development model by providing tools to design business services, tools to enable rapid connection to the underlying technical infrastructure, and mechanisms to bind these interfaces together.

Conclusion

The ESB category consists of vendor offerings with differing approaches to the implementation of SOA. The choice of a message-centric or service-centric ESB is one that dictates how quickly end-user solutions can be built and how easily they can be managed.

Messaging is a very important component of an ESB implementation. However, the message-centric approach to ESB overemphasizes the transport implementation while insufficiently addressing the core of the SOA: the services. The ability to create, deploy, and manage services must be front and center in an ESB offering. Solutions that do not offer these capabilities or that outsource them to a different infrastructure are not helping you solve your integration problems, they are merely exacerbating them. ESBs that require the duplication of existing messaging infrastructure do not result in infrastructure that is easier to develop or easier to administer—in fact, the opposite is true. Solutions that promote a service-oriented architecture, but at the same time present a highly granular programming model that closely couples your implementation to a proprietary transport, should be treated with some suspicion.

On the other hand, the service-centric approach to SOA implementation will yield the following benefits:

- A transport-agnostic infrastructure enhances connectivity and encompasses existing interfaces without requiring code modifications.
- The ability to reuse your existing messaging infrastructure reduces complexity and therefore development and management overheads.
- The ability to develop, deploy, and manage services on a single platform reduces complexity and enhances productivity and flexibility.
- The ability to build coarse-grained, robust, and reusable services enhances business productivity and agility.
- The ability to seamlessly integrate and interchange business services, processes, and transforms in a single platform significantly simplifies the development and management of real-world SOAs.