ESB, the beginning

The broker model of EAI was successfully implemented by some companies, but the vast majority of integration projects using this model ultimately failed. The lack of clear standards for EAI architecture and the fact that most early solutions were proprietary meant that early EAI products were expensive, heavyweight, and sometimes did not work as intended unless a system was fairly homogenous.

The effects of these problems were amplified by the fact that the broker model made the EAI system a single point of failure for the network. A malfunctioning component meant total failure for the entire network. In 2003, one study estimated that as many as 70 percent of integration projects ultimately failed due to the flaws in early broker solutions.

In an attempt to move away from the problems caused by a brokered hub and spoke EAI approach, a new EAI model emerged - the bus. While it still used a central routing component to pass messages from system to system, the bus architecture sought to lessen the burden of functionality placed on a single component by distributing some of the integration tasks to other parts of the network.

These components could then be grouped in various configurations via configuration files to handle any integration scenario in the most efficient way possible, and could be hosted anywhere within the infrastructure, or duplicated for scalability across large geographic regions.

As bus-based EAI evolved, a number of other necessary functionalities were identified, such as security transaction processing, error handling, and more. Rather than requiring hard-coding these features into the central integration logic, as would have been required by a broker architecture, the bus architecture allowed these functions to be enclosed in separate components.

The ultimate result - lightweight, tailor-made integration solutions with guaranteed reliability, that are fully abstracted from the application layer, follow a consistent pattern, and can be designed and configured with minimal additional code with no modification to the systems that need to be integrated.

This mature version of the bus-based EAI model eventually came to be known as the Enterprise Service Bus, or ESB.

Core ESB Features

- Location Transparency: A way of centrally configuring endpoints for messages, so that a consumer application does not require information about a message producer in order to receive messages
- Transformation: The ability of the ESB to convert messages into a format that is usable by the consumer application.
- Protocol Conversion: Similar to the transformation requirement, the ESB must be able to accept messages sent in all major protocols, and convert them to the format required by the end consumer.
- Routing: The ability to determine the appropriate end consumer or consumers based on both pre-configured rules and dynamically created requests.
- Enhancement: The ability to retrieve missing data in incoming messages, based on the existing message data, and append it to the message before delivery to its final destination.

- Monitoring / Administration: The goal of ESB is to make integration a simple task. As such, an ESB must provide an easy method of monitoring the performance of the system, the flow of messages through the ESB architecture, and a simple means of managing the system in order to deliver its proposed value to an infrastructure.
- Security: ESB security involves two main components - making sure the ESB itself handles messages in a fully secure manner, and negotiating between the security assurance systems used by each of the systems that will be integrated.

ESB Features

- Lightweight: because an ESB is made up of many interoperating services, rather than a single hub that contains every possible service, ESBs can be as heavy or light as an organization needs them to be, making them the most efficient integration solution available.
- Easy to expand: If an organization knows that they will need to connect additional applications or systems to their architecture in the future, an ESB allows them to integrate their systems right away, instead of worrying about whether or not a new system will not work with their existing infrastructure. When the new application is ready, all they need to do to get it working with the rest of their infrastructure is hook it up to the bus.
- Scalable and Distributable: Unlike broker architectures, ESB functionality can easily be dispersed across a geographically distributed network as needed. Additionally, because individual components are used to offer each feature, it is much simpler and cost-effective to ensure high availability and scalability for critical parts of the architecture when using an ESB solution.
- SOA-Friendly: ESBs are built with Service Oriented Architecture in mind. This means that an organization seeking to migrate towards an SOA can do so incrementally, continuing to use their existing systems while plugging in re-usable services as they implement them.
- Incremental Adoption: At first glance, the number of features offered by the best ESBs can seem intimidating. However, it's best to think of the ESB as an integration "platform", of which you only need to use the components that meet your current integration needs. The large number of modular components offers unrivaled flexibility that allows incremental adoption of an integration architecture as the resources become available, while guaranteeing that unexpected needs in the future will not prevent ROI.

EAI Pitfalls

- Constant change: The very nature of EAI is dynamic and requires dynamic project managers to manage their implementation.
- Shortage of EAI experts: EAI requires knowledge of many issues and technical aspects.
- Competing standards: Within the EAI field, the paradox is that EAI standards themselves are not universal.
- EAI is a tool paradigm: EAI is not a tool, but rather a system and should be implemented as such.
- Building interfaces is an art: Engineering the solution is not sufficient. Solutions need to be negotiated with user departments to reach a common consensus on the final outcome. A lack of consensus on interface designs leads to excessive effort to map between various systems data requirements.
- Loss of detail: Information that seemed unimportant at an earlier stage may become crucial later.

- Accountability: Since so many departments have many conflicting requirements, there should be clear accountability for the system's final structure.
- Other potential problems may arise in these areas:
- Lack of centralized co-ordination of EAI work.[6]
- Emerging Requirements: EAI implementations should be extensible and modular to allow for future changes.
- Protectionism: The applications whose data is being integrated often belong to different departments that have technical, cultural, and political reasons for not wanting to share their data with other departments