Understanding Enterprise Application Integration - The Benefits of ESB for EAI

Share this article

(/node/404/share/linkedin)

In today's enterprise infrastructure, system and application integration

started researching application and data integration solutions, it's easy to get lost in a sea of acronyms, opinions, and confusing marketing language.

Rapid advancements in EAI technology to meet the increasing demand for integration in the enterprise often results in arguments about what EAI is or isn't, or how the small differences between one proprietary approach or another make it the only viable solution.

In this article, we'll clear up the confusion, with an easy to understand, clearly organized look at the evolution of EAI.

Starting with a brief history of the origins of EAI, we'll walk through all the major developments in EAI architecture, and learn how traditional "hub and spoke" broker-based EAI systems are now being replaced by agile, distributed, standards-based Enterprise Service Bus architectures.

Table of Contents

Recommended for you

Winning with API-le
(https://www.mulesoft.
led-banking
utm_source=insig
recommendation&utm
recommendation&iesr
deb7-4988-ac
1042d7a8d4fa&at=7&r
ad5d-450d-be19-c705

Learn More

I. The Origins of EAI

Enterprise Application Integration, or EAI, has existed as a technical term since the early 2000s, but the central problem that it attempts to solve is much older. In a nutshell, EAI is an approach, or more accurately, a general category of approaches, to providing interoperability between the multiple disparate systems that make up a typical enterprise infrastructure.

Enterprise architectures, by their nature, tend to consist of many systems and applications, which provide the various services the company relies upon to conduct their day to day business. A single organization might use separate systems, either developed in-house or licensed from a third party vendor, to manage their supply chain, customer relationships, employee information, and business logic. This modularization is often desirable. In theory, breaking the task of running a business into multiple smaller functionalities allows for easy implementation of the best and newest technological advancements in each area, and quick adaptation to changing business needs.

However, to gain the benefits of this kind of distributed, modular system, an organization must implement technologies that deal with the problems presented by this architecture:

- Interoperability: the various components of the infrastructure may use different operating systems, data formats, and languages, preventing connection via a standard interface.
- Data integration: in order for a modular, distributed system to be functional, a standard method of handling the flow of data between applications and systems to enforce consistency across the database is crucial.
- Robustness, Stability, and Scalability: Because they are the glue that holds together a modular infrastructure, integration solutions must be highly robust, stable, and scalable.

were largely handled using point-to-point integration. In a point-to-point integration model, a unique connector component is implemented for each pair of applications or systems that must communicate.   This connector handles all data transformation (/resources/esb/data-transformation), integration, and any other messaging related services that must take place between only the specific pair of components it is designed to integrate.

When used with small infrastructures, where only two or three systems must be integrated, this model can work quite well, providing a lightweight integration solution tailor-made to the needs of the infrastructure. However, as additional components are added to an infrastructure, the number of point-to-point connections required to create a comprehensive integration architecture begins to increase exponentially.

A three-component infrastructure requires only three point-to-point connections to be considered fully integrated. By comparison, the addition of just two more components increases this number to 10 connectors. This is already approaching an unmanageable level of complexity, and once an infrastructure includes 8 or 9 component systems, and the number of connections jumps into the 30s, point-to-point integration is no longer a viable option.

Remember that each of these connectors must be separately developed and maintained across system version changes, scalability changes, and more (or, in some cases, even purchased at high cost from a vendor), and the unsuitability of point-to-point integration for complex enterprise scenarios

Related Articles

- Business Process Integration | BPI (/resources/esb/busines
process-integration)
- What is Application Orchestration? (/resources/esb/what-application-

(/resources/esb/enterpri
modeling-for-it-infrastructure)
- Financial Services and Legacy Systems: The Opportunity of Modernization (/resources/esb/financial
services-legacy-systems-modernization)
- Data Integration and ETL | Data Management (/resources/esb/data-integration-and-etl-data-management)

becomes painfully clear.

The EAI Approach To Integration

To avoid the complexity and fallibility of integrating complex infrastructures using a point to point approach, EAI solutions use various models of middleware to centralize and standardize integration practices across an entire infrastructure.

Rather than each application requiring a separate connector to connect to every other connector, components in an EAI-based infrastructure use standardized methods to connect to a common system that is responsible for providing integration, message brokering, and reliability functionalities to

snarled mess of brittle connections. EAI systems bundle together adapters for connectivity, data transformation engines to convert data to an appropriate format for use by the consumer, modular integration engines to handle many different complex routing scenarios simultaneously, and other components to present a unified integration solution.

EAI loosens the tightly coupled connections of point to point integration. An application can send a message without any knowledge of the consumer's location, data requirements, or use for the message - all of this information can be handled by the EAI implementation. This allows for a more flexible architecture, where new parts can be added and removed as needed, simply by changing the configuration of the EAI provider, and simplified modular development, where a single service can be re-used by multiple applications.

Many modern EAI approaches also take advantage of the opportunity presented by adding a central integration mechanism to further consolidate messaging tasks. In addition to data integration, a modern EAI may also include functionalities such as network administration, security (/platform/soa/mule-enterprise-security), acceleration, and scalability.

II. Traditional EAI

The first EAI solutions on the market took the idea of unifying integration literally, and incorporated all the functionality required for integration into central hubs, called brokers.

In this section, we'll look at the advantages and disadvantages of this model, and learn why it is being abandoned in favor of ESB architecture.

The Broker Model

In a broker approach to EAI, a central integration engine, called the broker, resides in the middle of the network, and provides all message transformation, routing, and any other inter-application functionality. All communication between applications must flow through the hub, allowing the hub to maintain data concurrency for the entire network.

Typically, implementations of the broker model also provide monitoring and auditing tools that allow users to access information about the flow of messages through their systems, as well as tools to speed up the

Advantages

Like all EAI approaches to integration, the broker model allows loose coupling between applications.

This means that applications are able to communicate asynchronously, sending messages and continuing work without waiting for a response from the recipient, knowing exactly how the message will get to its endpoint, or in some cases, even knowing the endpoint of the message.

A broker approach also allows all integration configuration to be accomplished within a central repository, which means less repetitive configuration.

Disadvantages

Like any other architecture model that uses a central engine , is that the broker can become a single point of failure for the network. Since the broker is responsible for all concurrency between application's data sets and states, all messages between applicants must pass through it.

Under heavy load, the broker can become a bottleneck for messages. A single central destination for all messages also makes it difficult to use the broker model successfully across large geographical distances.

Lastly, implementations of the broker model are often heavyweight, proprietary products, aimed at supporting a specific vendor's subset of technology. This can present problems if your integration scenario involves products from several vendors, internally developed systems, or legacy products that are no longer supported by the vendor.

III. ESB - The Next Step in EAI

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.

one study estimated that as many as 70 percent of integration projects ultimately failed due to the flaws in early broker solutions.

Bus Architecture - A New Approach to EAI

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.

The Enterprise Service Bus Is Born

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

There are a number of different ESB products available on the market today. Some, such as WebSphere Message Broker or TIBCO BusinessWorks, are traditional EAI products that have been re-factored to offer ESB like

using open messaging and integration standards to implement the ESB model.

Despite their differences, most ESBs include all or most of the following core features, or "services":

- 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.

The Advantages of ESB

Here's a look at the advantages offered by an ESB approach to application integration:

- Lightweight: because an ESB is made up of many interoperating services, rather than a single hub that contains every possible service, ESBs can be

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.

When To Use An ESB - Making Informed EAI

Decisions

All integration solutions have strengths and weaknesses, which are often dependent on the environment in which they are deployed. For this reason, making informed decisions about your EAI strategy is vital to the success of your integration initiative.

In order for your EAI and SOA efforts to be successful, you don't just need the "best" technology around - you need hard facts about the product's intended use scenario, performance under load, maturity, and a deep understanding of the present and future integration challenges your organization must overcome.

Before you make a decision about EAI, it's important to have a good idea of

Products Solutions Industries Services Resources Partners

- Will I need to add additional applications in the future?
- How many communication protocols will I need to use?
- Do my integration needs include routing, forking, or aggregation?
- How important is scalability to my organization?
- Does my integration situation require asynchronous messaging, publish/consume messaging models, or other complex multi-application messaging scenarios?

Ross Mason, MuleSoft founder and original architect of Mule ESB, has written an article called "To ESB or Not To ESB (http://blogs.mulesoft.org/to-esb-or-not-to-esb/)" that provides a good introduction for organizations considering an ESB approach to integration. The article includes an expanded version of the checklist above, to help determine whether or not ESB is a good match for their integration needs.

Want to know more about the future of Mule ESB?

The Mule development team has released new Development features to the platform, including:

- Mule Enterprise Security (/platform/soa/mule-enterprise-security)
- High Availability (/platform/soa/high-availability-mule-esb)
- DataWeave (/integration-solutions/dataweave-integration)