International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2018 IJSRCSEIT | Volume 3 | Issue 5 | ISSN : 2456-3307

A Case Study on Enterprise Application Integration (EAI) Using Enterprise Service Bus (ESB)

Dr. Manish L. Jivtode
Department of Computer Science, Janata Mahavidyalaya, Chandrapur, Maharashtra, India

ABSTRACT

The Enterprise Service Bus (ESB) promises to build up a Service-Oriented Architecture (SOA) by iteratively integrating different kinds of isolated applications into a decentralized infrastructure. It combines best features of EAI, like MOM, (Web) services, routing and XML processing facilities. ESB refers to architecture, a product or a way of doing things. This research paper distinguishes the ESB from earlier EAI solutions. It then discusses the key ESB components and their functions and their role in ESB architecture.

Keywords: Application Integration, Enterprise Service Bus (ESB), Cloud Integration, Azure Integration, Enterprise Application Integration (EAI), Distributed application integration, SOA Integration.

I. INTRODUCTION

Due to the globalization, enterprises across the world have to face immense competition. In order to stay in business, they constantly have to automate business processes, integrate with business partners and provide new services for the customers. The goal of IT is to actively support enterprises in this scenario. It has to make information available across the enterprise in order to allow software developers to integrate business processes in a unified way. Due to this need, cloud computing has evolved over the past years which employs Enterprise Service Bus (ESB).

Secondly, cloud computing saves on IT spending, reduces new systems implementation time, efforts and risks, eliminates regular system maintenance, and provides pervasive IT services[1][2].

Most enterprises try to achieve business integration using Service Oriented Architecture (SOA) [3]. SOA addresses business integration using coarse grained, loosely coupled business services. These business services easily allow creating and automating business processes by reusing the provided business functionality [4].

The Enterprise Service Bus (ESB) addresses the integration problems faced by diverse applications. It iteratively integrates isolated applications into a decentralized infrastructure [5]. In Gartner terms, ESB is the most promising approach for enterprise application integration (EAI) [6].

ESB is often used to name different things like architecture, a product or a "way of doing things". This research paper discusses what ESB exactly is. It discusses its key components, their functionality and their features. Finally it concludes with an answer to the question "What is an ESB?"

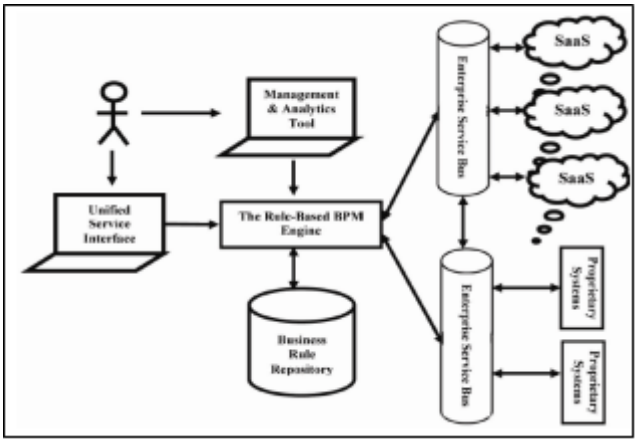

Figure 1. ESB Based Integration Architecture

II. ESB: APPROACH TO EAI

Enterprise Application Integration (EAI) utilizes ESB to build SOA applications. This approach to EAI through ESB is very innovative and challenging. This research paper focuses on need for integration, need for ESB and problems addressed by ESB.

III. NEED FOR INTEGRATION

Current heterogeneous IT landscape in most enterprises consists of variety of different applications. Different IT projects conducted to develop new applications, to refractor existing applications, customize and introduce standard applications. Each of these applications has been bought for a particular purpose, supports people in a specific domain and owned by certain department in the enterprise. The heads of these departments try to protect their resources (machines and applications) and the information gathered and maintained by their people. They only share their resources if it is either beneficial to them or if the enterprises management forces them to do so. This results in IT landscape across the enterprise in many isolated applications.

Due to globalization, enterprises have to face immense competition. To stay in business, they have to reduce their costs through reduced IT spending, new systems' implementation time, efforts and risks, eliminates regular system maintenance and provides pervasive IT services, process optimization and gain new market shares through process and product innovations. In order to achieve this, applications from different domains and departments have to be integrated. To integrate their applications they have to setup one or more integration projects. Each project has the goal to integrate affected applications.

Two common approaches for application integration are point-to-point integration and centralized EAI broker integration. Point-to-point integration directly connects two applications. EAI broker integration connects two or more application via a centralized mediator. This mediator is capable of routing and transforming messages sent between the applications.

IV. NEED FOR ESB

Point-to-point integration leads to unreliable, insecure, non-monitor able and non-manageable communication channels between applications. However, these are tightly coupled applications meaning integration application has to know the target application, interface methods to call, the required protocol to talk and the required data format to send. The problem is that the process and data transformation logic are encoded into the application. Thus, a new integration project has to be launched to refractor the depending applications each time when a change occurs in an application.

EAI integration uses a centralized EAI broker to integrate all applications. However, this results in so called islands of integration. This approach solves most of the point-to-point integration problems. However, managing resource in this approach is somewhat difficult.

V. ESB PROMISE

ESB promises to construct an SOA by iteratively integrate different kinds of isolated applications into a decentralized infrastructure called service bus. ESB is based on EAI platform in terms of special message routing and transformation. Decentralized infrastructure does not force enterprise departments to integrate their applications into a centralized EAI broker. Instead, it allows departments with limited access to their business functionality and information. ESB infrastructure is not only decentralized but also highly distributed and versatile thus allowing bringing all kinds of applications step-by-step to the service bus.

ESB compared with EAI replaces all direct application connections through reliable, secure and manageable virtual channels. With virtual channels applications are decoupled leading to loosely coupled interactions and interfaces. It allows standardized message exchange between different business services using XML as data format and SOAP, HTTP/REST as message exchange protocol.

In a heterogeneous computing environment like today's, integrating disparate systems is risky and a great challenge. However, it is important to business success particularly for automating and streamlining business process management (BPM). Earlier API (Application Programming Interface) integration was a major success for integrating two different systems or services. However, API methods are tightly-coupled and only works for one-to-one connection. API is primarily used for data exchange between systems and lacks security management. Moreover, the industry lacks standards for API due to systems being greatly different and proprietary.

In the late 1990's, the IT industry designed and developed a Service-Oriented Architecture (SOA) aimed at creating a distributed computing architecture in which all software services could be integrated. In SOA, software services are distributed on networks and they are integrated with each other via a central service registry which is called broker. When a software service needs another service, it queries the registry with certain criteria. If the registry finds the service that matches the criteria, it sends a service contract with an endpoint address back to the requester. The requester then remotely invokes the requested services with the granted contract and address. When a new service is plugged into the SOA network, it registers itself in the central service registry for future requests. A SOA service is self-contained to implement certain predefined business logics; that is, the service interfaces. The term Enterprise Service Bus (ESB) is widely used in the context of implementing an infrastructure for enabling a service-oriented architecture (SOA). However, real-world experience with the deployment of SOAs has shown that an ESB is only one of many building blocks that make up a comprehensive Service-Oriented Infrastructure (SOI). The term ESB has morphed in a number of different directions, and its definition depends on the interpretation of individual ESB and integration platform vendors and on the requirements of particular SOA initiatives.

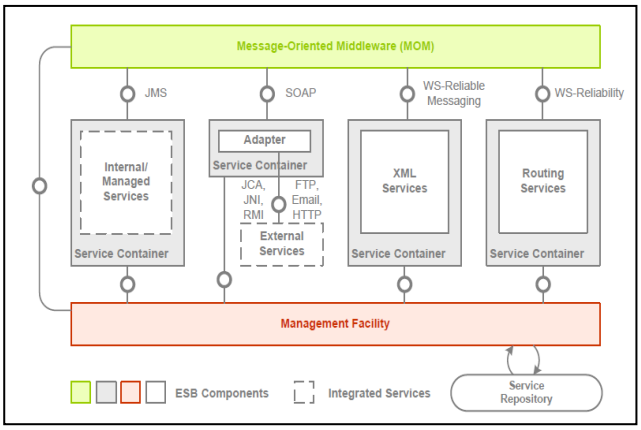

Figure 2. ESB Architecture

Based on the experience Service Oriented Integration (SOI) implementations, one can think of an Enterprise Service Bus as a collection of architectural patterns based on traditional enterprise application integration (EAI), message-oriented middleware, Web services, .NET and Java interoperability, host system integration, and interoperability with service registries and asset repositories.

The key components of ESB architecture are MOM, service container and management facility. A service container contains adapter module, mediation module, message routing module, security module, and management module.

ESB greatly improves the central service registry mechanism in SOA and provides a software infrastructure for SOA implementation in enterprise applications as well as system integration across

organizational boundary. Thus, ESB greatly enhances the usage of SOA with a virtual bus to connect many disparate systems and services together.

ESB is message-based bus architecture which consists of a set of software components called service containers. Service containers are interconnected over a reliable and secure messaging channel. A service container connects one or many software services or systems through service adapter(s).

A system or service sends request messages directly to its connected service container which in turn processes and routes the messages to a destination (requested) service container. The destination service container processes and forwards the messages to its connected destination (requested) service or system through a service adapter. During this process, ESB service containers process, monitor, logs and manage messages to make sure all services and systems are connected reliably and securely.

VI. MESSAGE ORIENTED MIDDLEWARE

MOM is Message Oriented Middleware which is highly distributed network of message servers and backbone of ESB. It establishes a reliable, secure, and manageable virtual channel and sends messages over them.

In MOM (Message Oriented Middleware), all direct communication channels between applications are replaced by virtual communication channel. So, all synchronous remote calls are replaced by asynchronous message exchange. All tightly coupled point-to-point interactions are replaced by loosely coupled indirect interactions. MOM actually consists of network of message servers and number of message clients. The message server manages various queues, topics and relays and stores messages sent. ESB contains multiple message servers connected to each other. MOM routes the messages reliably through network of message servers. Each message server on the route stores the message, tries to send it to the next message server and deletes it only if the target server has acknowledged the reception. Thus, MOM guarantees the message delivery.

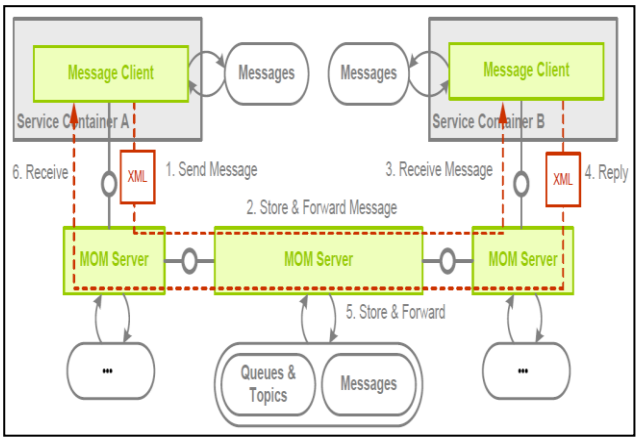

Figure 3. Message flow in MOM

A. Queues

It allows one-directional communication. Each queue acts as an intermediary (sometimes called a broker) that stores sent messages until they are received. Each message is received by a single recipient.

B. Topics

It provides one-directional communication using subscriptions-a single topic can have multiple subscriptions. Like a queue, a topic acts as a broker, but each subscription can optionally use a filter to receive only messages that match specific criteria.

C. Relays

It provides bi-directional communication. Unlike queues and topics, a relay doesn't store in-flight messages; it's not a broker. Instead, it just passes them on to the destination application.

A service container manages an application internally or provides access to an external application via an appropriate adapter. Adapter manages access to all kinds of applications. It allows upload, download files,

to send and receive emails or to invoke a remote method via RMI. The service container makes the business functionality implemented by the managed application available as business services. These intelligent service containers and highly distributed MOM give the ESB its decentralized nature. In ESB, many special services are available which includes routing and XML processing services.

The adapter module acts as a service connector to connect various software services and systems such as ERP or CRM. A service adapter, very similar to a hardware or software driver (e.g. printer driver, database driver), uses the native transaction interfaces in the proprietary service to transfer messages between the service container and the service. ESB solution providers provide a range of service adapters for various systems and services.

The mediation module transforms protocol, message format, and message content between the requesting service and the service provider. Mediation is critically important for the system integration because services on the ESB use different protocols and data formats. The mediation module is implemented using the XML-based transformer components which are configured through XSL (Extensible Style sheet Language). More complex transformers are required to invoke other SOA services or query database.

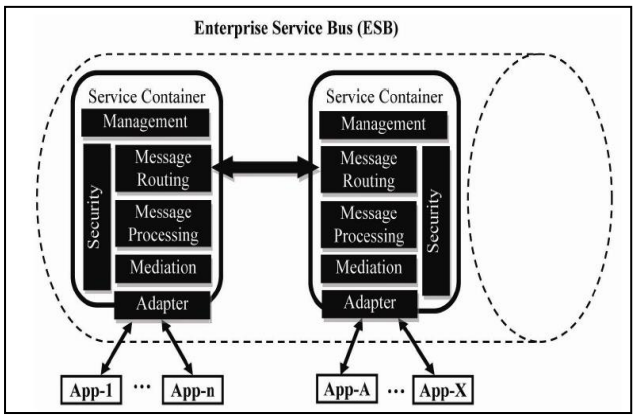

Figure 4. Detailed Service Containers

The message processing module processes incoming and outgoing messages and implements event handling. This module sorts, prioritizes, delays, and reschedules message delivery as needed to guarantee both synchronous and asynchronous communications. It monitors and logs messages for quality of services (QoS) and security management. Using the message-encoded logic and content-based logic, the module conducts message validation, transformation and aggregation, and message buffering and delaying. In addition, the module supports various event handlings such as event triggering, noticing, filtering, and mapping. Artificial intelligence technologies can be built in this model to improve message processing and event handling. For example, in the ESB of an investment bank, a proactive event handler automatically checks a stock market and updates the data correspondingly so that the future event handling always uses the latest market information.

The message routing module routes messages from a service requester to the service providers using the XML-enabled content-based routing method. The content-based method provides a highly configurable, dynamic and intelligent routing. For example, when the message content indicates a customer doesn't want to book a flight ticket, the request will not be routed to the service provider that books a flight ticket. The routing module is implemented with popular web service techniques such as Simple Object Access Protocol (SOAP) and XML Path Language (XPath). The routing module provides a reliable and secure message exchange channel between the service requester and the service provider.

The security module enforces compatibility between all modules and security policies. Particularly, the security module implements security standards and policies including authentication, authorization, encryption, auditing, and intrusion detection. The security module is also responsible for unifying security management throughout the service

container during message receiving, processing, and sending.

The management module plays a critical role in ESB. This module tracks activities happening in a service container and handles a variety of exceptions. It also manages workload, schedules tasks, creates threads, registers services, manages service transaction and its lifecycle, and manages service state and quality of service (QoS). ESB is a virtual service bus architecture that connects many distributed and decentralized service containers.

Additionally, service registration and invocation are highly dynamic. This requires ESB to be highly configurable and customized to meet different service demands. The management module therefore provides administration tools that configure, manage and control service container. A centralized management tool can be built to manage and configure all service containers in the ESB.

With administration tools, users can configure ESB containers without requiring shutdown or interrupting the integrated services. ESB adopts SOA and highly enhances the SOA implementation and functionalities by replacing the central registry with the bus architecture. It makes the system and service integration a truly plug-and-play process.

ESB is message-based distributed integration software platform in SOA. It is open-standard, platform-independent and vendor-neutral. It can run on any operating system and hardware structure, and can be implemented with different technologies (e.g. J2EE, Microsoft .NET). With many service containers distributed and decentralized on the Internet, ESB creates a virtual service bus for system and service integration.

VII. ROUTING FACILITIES IN ESB

Itinerary-Based Routing: Itinerary based routing is often used to manage short-living, transient process fragments called as micro flows. A micro flow consists of a sequence of logical steps. Each logical step refers to a business service. Thus, to enact a micro flow, a message is sent through the service bus in such a way that all business services are invoked. Therefore, the service bus can be thought as a highly distributed routing network that is build up by a variety of message servers and service containers.

In order to route a message through the bus, each message contains an itinerary. The itinerary consists of a list of ESB endpoints that needs to be visited and the information about already visited ESB endpoints. The message also contains the current processing state as message payload. Because the itinerary and the process state is carried by the message as it travels across the bus, each service container is able to evaluate the itinerary and to decide in which virtual channel the message has to be placed, to send it to the next ESB endpoint in the list.

VIII. SERVICE ORCHESTRATION USING BPEL

Service orchestration using BPEL is used to manage long-running business processes that might run for months or years. A BPEL process definition consists of a number of logical steps that are connected to each other by conditional or unconditional links and can be executed in sequence or in parallel. A BPEL process definition also allows defining time-based, condition-based and event-based triggers. As in the itinerary based routing, each logical step refers to an ESB endpoint.

A service orchestration or BPEL engine is used to enact BPEL processes based on the process definitions. The BPEL engine is provided by the ESB as a special

service via an ESB endpoint can therefore be accessed like any other service. Depending on the setup, an ESB might contain multiple BPEL engines in different geographic locations that manage different BPEL processes.

IX. CONTENT-BASED ROUTING

Content-based routing (CBR) is based on the fact that XML processing services with different capabilities are plugged into the bus. They allow validating, enriching, transforming, and route and operate XML messages. Combinations of these services allow forming lightweight processes with the sole purpose to process messages. Plugging such a lightweight process as CBR service into the message flow between a message producer and a message consumer allows to handle all kinds of complex integration tasks, for example before and after a service invocation.

X. CONCLUSION

This research paper discusses the Enterprise Service Bus (ESB) architecture, Service Oriented Architecture (SOA), integration need and issues in great detail. Finally, it concludes with the answers to need for enterprise integration and the approach to achieve it.

ESB is an incremental approach of constructing a SOA by connecting all kinds of applications to enterprise-wide distributed infrastructure.

ESB is an architectural style in which applications are service-enabled through service containers and connected to a MOM based service bus that is not only capable of routing messages but also of transforming them.

There are many companies that sell ESB infrastructure products allowing enterprises to build up an ESB. These products are often composed out of existing components, such as MOMs, J2EE servers and EAI integration adapters, and provided in a manageable manner.

XI. REFERENCES

[1]. Fowler, M.. Patterns of Enterprise application Architecture. Addison Wesley, 2002.

[2]. Chappell, D. A.: Enterprise Service Bus.O'Reilly Media Inc., 2004.

[3]. Hohpe, G., Woolf, B.: Enterprise Integration Patterns. Pearson Education, 2004.

[4]. Krafzig, D., Banke, K., Slama, D.: Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, 2004.

[5]. Alonso, G., Casati, F., Kuno, H., Machiraju, V: Web Services: Concepts, Architectures and Applications. Springer-Verlag, 2004.

[6]. Keen, M. et al.: SOA with an Enterprise Service Bus in Web Spehere. http://www.redbooks. ibm.com/redbooks/pdfs /sg246494.pdf, 2005.

[7]. Pulier, E.,Taylor, H.: Understanding Enterprise SOA,Manning, 2006.

[8]. Tabeling, P., Groene, B., Knoepfel, A.:Fundamental Modeling Concepts-Effective Communication of IT Systems. John Wiley &Sons, Ltd., 2006.

[9]. PolarLake: Understanding the ESB.http://www.polarlake.com/en/ assets / whitepapers /esb.pdf.

[10]. Sun Microsystems: Service Oriented Business Integration. http://java.sun.com/ integration/.

[11]. Business Process Management Initiative: BPMN: Business Process Modelling Notation 1.0

[12]. FMC Consortium: FMC: Fundamental Modelling Concepts. http://www.f-m-c.org.

[13]. BEA Systems, IBM, Microsoft, SAP AG and Siebel Systems: Business Process Execution Language for Web Services 1.1 (BPEL4WS). http://www- 128.ibm.com

/developerworks/Library /specification /ws-bpel/.

[14]. The Web Services Resource Framework.http://www-106.ibm.com/developerworks /library/ws-resource/ws-wsrfpaper.html

[15]. The SNMP Protocol.http://www.snmp.com /protocol/

[16]. W3C: SOAP specification http:// www.w3 .org /TR/soap/.

[17]. W3C: XML Transformations (XSLT).http://www.w3.org/TR/xslt.

[18]. Mule ESB project page. http://mule .codehaus.org