A Unified Messaging-Based Architectural Pattern for Building Scalable Enterprise Service Bus

Karim M. Mahmoud[1,2]
[1] IBM, Egypt Branch
Pyramids Heights Office Park, Giza, Egypt
kmahmoud@eg.ibm.com
[2] Computer and Systems Engineering Department
Faculty of Engineering, Alexandria University
Alexandria, Egypt

ABSTRACT

Service-oriented architecture (SOA) has become the popular methodology to build large enterprise systems. Inside an enterprise, an enterprise service bus (ESB) has provided a reliable and efficient solution to have services and applications communicate with each other. This paper proposes a unified architectural pattern for building a scalable enterprise service bus which relies on messaging middleware that aims at fulfilling the different requirements of the business, while maintaining compatibility of the already deployed services. By analyzing different scenarios and business requirements, the proposed architectural pattern can save time, funds, and development effort needed to extend the currently deployed services in the enterprise.

Keywords

Service-Oriented Architecture (SOA), Enterprise Architectural Patterns, Enterprise Service Bus, Middleware Systems

1. INTRODUCTION

The integration of business services is a problem that has been addressed by different approaches (asynchronous middleware, Enterprise Integration application, etc.). Today, the need to integrate different applications in the enterprise is considered necessary. Service-oriented architecture is popular approach to integrate applications as they provide the level of scalability required to build applications. However, service-oriented computing is today essentially technology-driven. Most available platforms focus on the technology allowing to publish and compose services and to make them communicate (i.e. SOAP [10], WSDL [9], UDDI [8],..,etc.) [4]. Different services may manipulate similar data or services under very different formats. The problem of data and interfaces heterogeneity is not directly addressed and left to the e-applications programmers [4].

The Enterprise Service Bus (ESB) is a framework that helps to create, deploy, and orchestrate (communicate between) service components in a distributed system. It is a middleware architectural component that acts as a messaging backbone; it has bus architecture, and provides the infrastructure to build a SOA application [5]. A service deployed onto an Enterprise Service Bus can be triggered by a consumer or an event. ESB supports synchronous and asynchronous interactions between one or many stakeholders (one-to-one or many-to-many communications).

Using Enterprise Service Bus (ESB) frameworks to build different systems is efficient as it generally provides an abstraction layer on top of an implementation of system. However, there is no unified enterprise pattern for building a scalable Enterprise Service Bus architecture that can be applied on different systems while maintaining compatibility of the already deployed services.

In this paper, a unified architecture is proposed for building a scalable Enterprise Service Bus. This architecture is reliable, reusable, and extendable which can save time, funds, and development effort needed to extend and maintain the currently deployed services in the enterprise.

The paper is organized as follows. The next section provides a background to some concepts. Section 3 describes the proposed architecture and different components for building the ESB. Finally, section 4 provides analysis and conclusion.

2. BACKGROUND AND DOMAIN

To understand the benefits of an Enterprise Service Bus one has to examine the infrastructure requirements that large enterprises have. A typical scenario is that an enterprise runs hundreds of deployed applications, which could be built by the company's development team, acquired from a third party or parts of legacy systems [6]. These deployed applications should communicate and exchange the required data with each other in order to work together to fulfill different business requirements of the company. There are good reasons for having different combinations of many different applications. First, it is nearly impossible to develop one huge application which performs all business functions of a typical enterprise because there are far too many requirements. The second reason is that running multiple applications gives IT managers flexibility to select solutions which are the best for their particular purpose. That means that integration is a fundamental requirement in the enterprises.

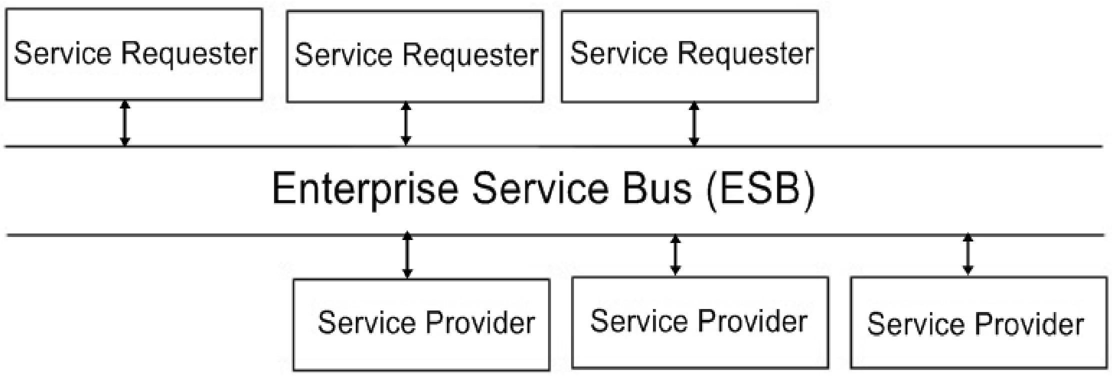

Figure 1: Enterprise Service Bus (ESB) Infrastructure

The Enterprise Application Integration (EAI) is needed if the applications of external business partners are required to be integrated into the current system and the selection of applications is not fixed. In this case the applications may be exchanged or new applications may be deployed so that the integration infrastructure has to be able to handle new scenarios [6].

2.1 Message-Oriented Middleware

The traditional solution for Enterprise Application Integration is to use Message-Oriented Middleware. In this case the asynchronous messaging is used to decouple the applications from each other. Message-Oriented Middleware systems are built using a message queueing system which is often called message broker. The main role of message broker is to provide a unified interface for sending and receiving messages.

The message broker is able to store the messages so that sender and receiver do not need to be connected at the same time. Within this middleware layer messages can be routed which makes it possible to deliver a single message to more than one recipient. Furthermore messages cold be transformed by the message broker to fit the requirements of the receiving application. The transformation facilities allow the connected applications to use their own native message formats.

The main problem of Message-Oriented Middleware solutions is that they use proprietary protocols and platform specific interfaces and deployments. This leads to a total dependency of the applications on the infrastructure and causes interoperability problems with Message-Oriented Middleware products of alternative vendors. As a result islands of Message-Oriented Middleware based infrastructures can often be found.

2.2 Service-Oriented Architecture

Service-Oriented Architecture (SOA) is an architecture pattern which allows the applications to provide their business functionality in the form of reusable services. A service is a self-contained and stateless business unit which is accessible through a standardized, implementation neutral interface. Services are used by other applications which could also be implementations of services. With this approach complex business processes are implemented through combination of several services. This is called service orchestration.

SOA can be implemented using any service-based technology. For example, web service technologies like SOAP or REST can be used. SOA allows complex enterprise applications and end-to-end business processes to be composed from these services, even when the providers of those services are applications hosted on different operating system platforms or written in different programming languages or even based on separate data models. This gives flexibility to support the fundamental goals of business integration, which connect different business systems across the enterprise and also it allows to extend business services to customers.

The adoption of SOA in business-critical applications is occurring only incrementally. Replacing or refactoring legacy applications with new standards-aware equivalents is due to be a slow process. That implies that an integration infrastructure can not be purely service-based. Nowadays, the enterprises need powerful integration solutions but they want them to be built on open standards and to support Service-Oriented Architecture. Those requirements led to the idea of an Enterprise Service Bus [6].

2.3 Enterprise Service Bus

Figure 1 shows the basic infrastructure of ESB. Technically, ESB is considered a messaging backbone which handles protocol conversion, message format transformation, routing, orchestration, accept and deliver messages from different services and applications, which are linked to ESB. Besides these it also provides a consistent approach for integrating applications. It supports webservice specifications, and is a pluggable framework where any new service can easily be plugged into the bus [5]. This helps in agile integration. The ESB implementation normally provides an abstraction over the messaging system. Thus the services providers and the service consumers on the ESB do not interact directly with each other; rather they will communicate via the bus which acts as a message broker between the applications, making it a loosely coupled architecture [5].

An ESB provides a variety of transport bindings for invoking services including HTTP, FTP, JMS etc. It also provides API for developing services and making services interact with each other reliably. The advantage of an ESB is that it reduces the number of point to point connections, which would otherwise lead to ambiguous type systems, which are very difficult to extend and maintain.

An ESB is an ideal backbone for implementing service oriented architectures because it provides universal mechanism to interconnect all the services required in the composed business solution without compromising security, reliability, performance and scalability.

In the following section, a unified enterprise pattern for building a scalable Enterprise Service Bus architecture is proposed. This pattern can be applied on different systems while maintaining the compatibility of the already deployed services. This pattern is reliable, reusable, and extendable which can save time, funds, and development effort needed to extend and maintain the currently deployed services in the enterprise.

3. PROPOSED UNIFIED ARCHITECTURAL PATTERN

Figure 2 shows the proposed unified architectural pattern for building the Enterprise Service Bus. The Enterprise Service Bus is the backbone which handles the communications between the service requester and the service provider. The proposed architecture aims at providing a unified pattern for building ESB layer which can be used in any enterprise. The service requester system (Frontend System) is expected to communicate with different protocols and message formats (ex: SOAP/HTTP, XML/Queue, TCP/IP,...,etc), for that reason the first internal layer in the ESB consists of a set of adapters components which provide an abstraction layer for the correspondent service requester system.

The second internal layer in the proposed ESB design is called Service Gateway. The Service Gateway is used to accept the requests received from the Service Requester Adapters layer. The main function of the Service Gateway component is routing the request messages to the appropriate business service in the next layer which is the Atomic Services components.

The third layer, which is the Atomic Services components, is considered the business layer which handles different business scenarios and makes the needed transformations. The last layer consists of a set of adapters components which provides an abstraction layer for the messages received from the service provider (Backend System).

In the next subsections, more details and design issues of different components are discussed.

3.1 Service Requester Adapter

The requester adapter component is an implementation of the adapter/wrapper design pattern [2]. The Adapter pattern enables a system to use components whose interfaces don't quite match its requirements. The input for this component depends on the service requester system specification (ex: SOAP/HTTP, XML/Queue,..etc), while the output format is XML [7] message which is called ESBMsg. The main function of this component is to wrap the message format received by the service requester system by simply transforming it into the unified internal messages format which is ESBMsg. As it is noticed in the proposed ESB, all of the internal layers use messaging-based (queueing) system to send and receive messages in between. It is well to notice that nowadays, many products provide a well-engineered architecture for a building a message-oriented middleware, one of this popular products is IBM Websphere MQ [1,3].

The service Requester Adapter supports both request and response communications, as it has two interfaces; one for the request message and the other for the response message. The role of the request interface is to convert the message received by the service requester into the ESB's internal message format which is ESBMsg. The ESBMsg format is XML message with some common information needed to build and provide efficient method for components' communication inside the ESB layer. The second interface is the response interface, the role of this interface is to convert the ESBMsg format into the service requester's message format in order to provide it with the required information. As it is shown in figure 2, the ESBMsg is sent to the queue of the next component which is the Service Gateway.

3.2 Service Gateway

The Service Gateway component acts as a router for the request and the response messages. For the request mode, the Service Gateway is responsible for routing the request messages from the Service Request Adapters to the appropriate Atomic Service Component based on the service name or the service number. For the response mode, the Service Gateway is responsible for routing the response messages from the Atomic Service Component to the the appropriate Service Requester Adapter Component. The following XML message shows the proposed format and the needed fields which should be included in the ESBMsg message.

```
<ESBMsg>
 <Header>
  <MsgId>..</MsgId>
        <ServiceId>..</ServiceId>
        ...
 </Header>
</ESBMsg>
```

In ESBMsg message, a common aggregate XML field called "Header" is used, it contains some common fields necessary for routing mechanism like "ServiceId" field.

3.3 Atomic Services

The Atomic Services Components are considered the business layer which handles all the needed message transformation, aggregation, and composition. These Atomic Services components are aware of the different business and services specifications. The Atomic service component uses two interfaces. The first interface is the business request interface which handles the transformation from the Service Requester Adapter message into the Service Provider Adapter message. The second interface is the business response interface which handles the transformation from the Service Provider Adapter message into the Service Requester Adapter message. Both messages are included in the "Body" aggregate field as follows.

```
<ESBMsg>
 <Header>
  <MsgId>..</MsgId>
        <ServiceId>..</ServiceId>
        ...
 </Header>
 <Body>
 ... Business request/response ...
 </Body>
</ESBMsg>
```

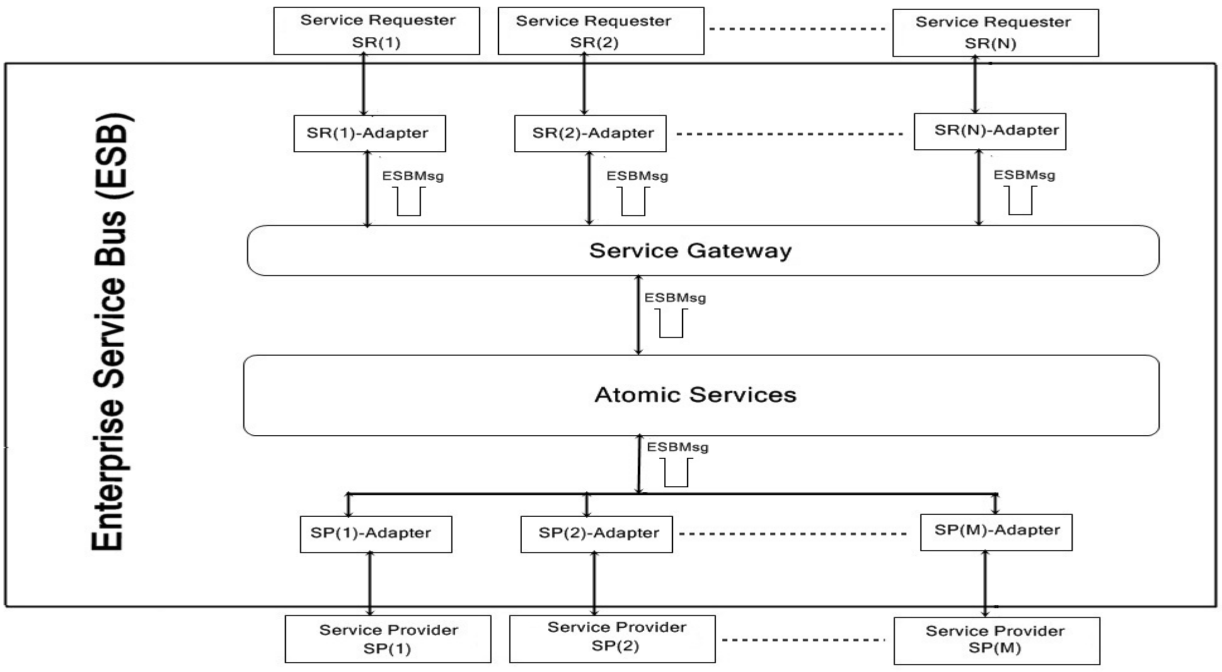

Figure 2: Proposed Unified ESB Architectural Pattern

3.4 Service Provider Adapter

Service providers could use different message formats. In the proposed architecture, each service provider system is wrapped by a Service Provider Adapter component. The Service Provider Adapter component creates an abstraction layer for sending different request messages to the service providers (Backend systems) and receiving response messages from them.

Service providers systems could use different message formats. In the proposed architecture, each service provider is wrapped by a Service Provider Adapter component. The Service Provider Adapter components create an abstraction layer for sending different request messages to the systems that act as a service provider (Backend systems) and receiving response messages from them.

The response interface in the Service Provider Adapter component is responsible for receiving the response from the system provider system and wrapping it into the ESBMsg message, which will be sent again to the Atomic Services layer.

3.5 Integrating ESB Components

In the previous subsections, the different components needed to build the proposed ESB layer have been discussed. In this subsection, we will show how to integrate these components together. Figure 3 shows the sequence diagram of the proposed ESB architecture. First, the system requester (consumer) starts to call the ESB using its own protocol and message format. Then, the Service Requester Adapter receives this message and converts it into the ESBMsg message format and puts this converted message into the queue of the Service Gateway. After that, the Service Gateway determines which Atomic service component the message should be sent based on the value of the "ServiceId" field in the ESBMsg message and then it puts the message into the queue of the appropriate Atomic Service. The Atomic Service receives the ESBMsg message in its queue and starts processing it and handling the business requirements needed by the service provider system and then it puts the transformed data into the "Body" field in the ESBMsg which will be sent to the queue of the appropriate Service Provider Adapter. The Service Provider Adapter receives the ESBMsg message, then it extracts the business request message from the "Body" field and sends it to the service provider system to get the business response.

After receiving the business response message from the service provider system, the Service Provider Adapter wraps this response into the "Body" field in the ESBMsg message, and put it into the queue of the Atomic Service. The Atomic Service transforms the business response into the appropriate message format needed by the service requester system and put it into the queue of the Service Gateway component, which in turn will route it again to the Service Requester Adapter and then the service caller will receive the needed response.

4. ANALYSIS AND CONCLUSION

In this section, different scenarios of possible business changes and new requirements in the enterprise will be discussed. By analyzing these new requirements, it will be shown that applying these changes to a middleware system which uses the proposed ESB architectural pattern is easy and can be maintained without making dramatic changes to the system which in turn will save funds and time needed to rebuild the new ESB layer.

The first scenario is changing the communication protocol or the message format of one or more of the service requester (consumer) systems. In this case, the only component that will be affected is the Service Requester Adapter Component of that system. This is mainly the benefit of having an

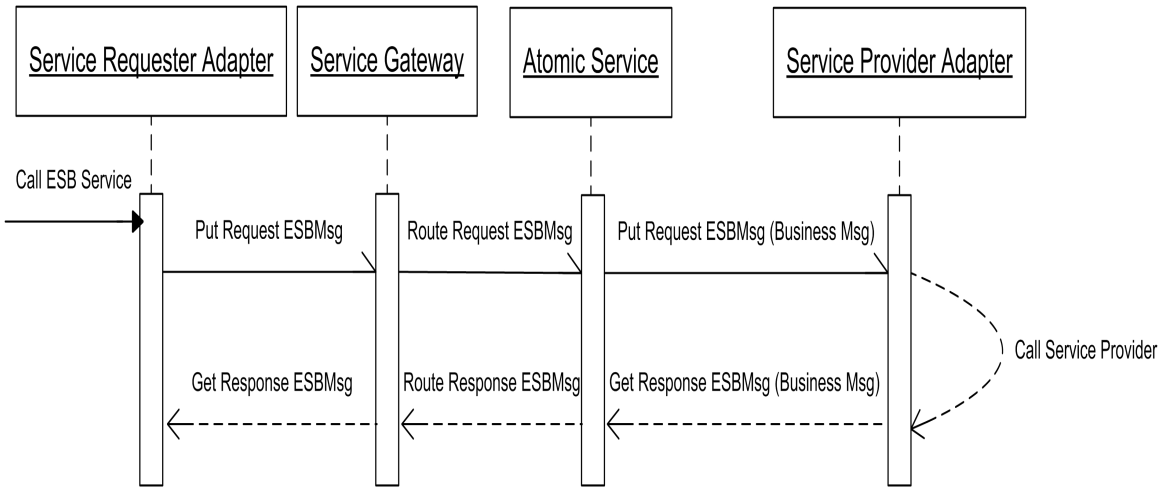

Figure 3: Sequence Diagram of the Proposed ESB

abstraction layer which wraps different requests regardless of the communication protocol. The same case also applies for changing the communication protocol of the service provider system, as in this case the correspondent Service Provider Adapter is the only component that will be affected and will need some modifications.

Another scenario is the case where there is a need to change the business requirements of one or more of the deployed services. In this case, the only components which will be affected are the Atomic Service components. This is mainly the benefit of having a separate layer which wraps the business details.

Also, the proposed architectural pattern is useful in case of adding a new service requester system which will need to consume the already deployed services. In this case, none of the already deployed components will be changed and only one more Service Requester Adapter will be added in the ESB.

In conclusion, the proposed architecture provides a unified pattern to build a scalable Enterprise Service Bus. By adopting this architectural pattern, there will no need to make huge development efforts to fulfill any needed changes whether in communication protocols or the business requirements which in turn will allow the business firms to save funds and time.

5. REFERENCES

[1] IBM Webpshere MQ. [online] www.ibm.com/software/products/us/en/wmq/.

[2] H. Gamma and V. Johnson. Design Patterns (the Gang of Four book). Addison-Wesley, 1994.

[3] L. Gavin, P. G. Gerd Diederichs, K. P. Hendrik Greyvenstein, and A. V. Sreekumar Rajagopalan. An EAI solution using websphere business integration (v4.1). IBM, 2003.

[4] C. Herault and P. L. Gael Thomas. Mediation and enterprise service bus. *Proceedings of the First International Workshop on Mediation in Semantic Web Services (MEDIATE)*, 2005.

[5] M. Keen, S. B. Amit Acharya, S. M. Alan Hopkins, R. R. Chris Nott, and P. V. Jonathan Adams. Patterns: Implementing an SOA using an enterprise service bus. IBM, 2004.

[6] F. Menge. Enterprise service bus. *Free and open source software Conference*, 2:1–6, 2007.

[7] M. Murata and D. K. Simon St Laurent. XML media types. Technical report, January 2001.

[8] OASIS. Uddi executive overview: Enabling service-oriented architecture. Technical report, 2004.

[9] W3c. Web services description language (wsdl) 1.1. Technical report, March 2001.

[10] W3C. Simple object access protocol (soap) 1.2. Technical report, June 2003.