Chapter 6 – Service-Oriented Architectures, Enterprise Service Bus, Middleware from Oracle and TIBCO

Eduardo Felipe Zecca da Cruz

Summary

This paper introduces the concept of Service-Oriented Architectures, its uses, applications, and its effects on the decisions of the stakeholders. Web-Services, which is the most common implementation of Service-Oriented Architecture, is also introduced and discussed. In addition, the Enterprise Service Bus is discussed and several concrete examples are given to show the areas of application that an ESB can be used.

Introduction

Currently, services have become more used by architects and designers to develop software. A service is an unassociated, loosely coupled unit of functionality that is self-contained and implements at least one action such as getting information about a bank account or changing an online order at Amazon. The Service-Oriented Architecture, as known as SOA, is the underlying structure supporting communications between services, which means that services are going to use defined protocols to describe their characteristics and the data that drives them. However, several people see SOA as a Web-Service. Web-Services are the most common implementation of SOA but they are not the same thing, and there are non-Web Services implementation of SOA. Finally, Enterprise Service Bus, as known as ESB, is another concept of using SOA that is very used but still not clear its definition, benefits and when it should be used on a system.

Service-Oriented Architecture

Service-Oriented Architecture is defined as the underlying structure supporting communications between services and it defines how two computing entities interact in a way as to enable one entity to perform a unit of work on behalf of another entity. For example, a business A could get some service b from vendor B, service c from vendor C, service d from vendor D, and so on. A deeper example is, when a user order something at Amazon with his/her credit card, the Amazon needs to interact with his/her credit card company to get the information about his/her account. This interaction is supported by SOA mechanisms and it can be seen on the model below.
SOA has several features, benefits and infrastructure that can be seen on the table below.

**Table 1 - SOA Features, Benefits and Infrastructure**

<table>
<thead>
<tr>
<th>Feature</th>
<th>Benefits</th>
<th>Supporting Infrastructure</th>
</tr>
</thead>
</table>
| Service             | Improved information flow  
                      Ability to expose internal functionality  
                      Organizational flexibility                     |                           |
| Service Re-use      | Lower software development and management costs                          | Service repository         |
| Messaging           | Configuration flexibility                                               | Messaging program         |
| Message Monitoring  | Business intelligence  
                      Performance measurement  
                      Security attack detection                                | Activity monitor          |
| Message Control     | Application of management policy  
                      Application of security policy                          | PDPs and PEPs             |
| Message Transformation | Data translation                                                | Data translator            |
| Message Security    | Data confidentiality and integrity                                      | Encryption engine          |
These features and benefits, make SOA has services readily available and results in quicker time to market, which is the amount of time that takes from a product being designed until its being available on consumer markets.

From an architectural perspective, SOA has three important perspectives that are the application architecture, the service architecture and the component architecture. First, the application architecture is the client that has an objective, such as order a product at Amazon and call other services to achieve this objective. Second, the service architecture works like a bridge between the implementations and the applications, which means that it going to have the services available for use when the application invokes one or more services. Finally, the component architecture is the one that has the environments supporting the applications and their implementations.
Finally, there are several myths about SOA, and it is important to have a really knowledge of them before go dipper into it. The table below describes these myths and the facts to demystify then.

Table 2 - Myths and Facts about SOA

<table>
<thead>
<tr>
<th>Myth</th>
<th>Fact</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOA is a technology</td>
<td>SOA is a design philosophy independent of any vendor, product, technology or industry trend. No vendor will ever offer a “complete” SOA “stack” because SOA needs vary from one organization to another. Purchasing your SOA infrastructure from a single vendor defeats the purpose of investing in SOA.</td>
</tr>
<tr>
<td>SOAs require Web Services</td>
<td>SOAs may be realized via Web services but Web services are not necessarily required to implement SOA.</td>
</tr>
</tbody>
</table>
SOA is new and revolutionary

CORBA and to some extent even older EDI and DCOM were conceptual examples of SOA

SOA ensures the alignment of IT and business

SOA is not a methodology

A SOA Reference Architecture reduces implementation risk

SOAs are like snowflakes – no two are the same. A SOA Reference Architecture may not necessarily provide the best solution for your organization

SOA requires a complete technology and business processes overhaul

SOA should be incremental and built upon your current investments

We need to build a SOA

SOA is a means, not an end

Web Service

Web Service is a method that enables the communication between two electronic devices over a network, and is the most common implementation of SOA. It is based in HTTP and XML, can be used by other applications, and makes functional building blocks accessible over the Internet protocols independent of the platforms and programming languages. A building block can be a service provider, a service requester, or both.

The service provider is the responsible to provide a web service and includes the application, the middleware, and the platform on which they run. In addition, the provider needs to make some decisions such as the services that it is going to expose, the price of the services and the amount of the offered information. On the other hand, a service requester is the one that is going to request a web service from a service provider. It also contains the application, the middleware, and the platform on which they run. Finally, a service requester could access multiple services if the service provider provides them.

The figure below shows the steps that are necessary to “engage” the provider and the requester on a web service. The steps may be automated or performed manually.
It is very common to think that Service Oriented Architecture and Web Service are the same thing but this is wrong. "Web services are about technology specifications, whereas SOA is a software design principle. Notably, Web services' WSDL is an SOA-suitable interface definition standard: this is where Web services and SOA fundamentally connect." said Yefim V. Natis in April 2003. An easy way to know the difference between SOA and Web Service is think that the SOA is the architectural pattern while Web Service is a way to implement SOA that gives several benefits such as platform independence, loose coupling, self-description, discovery, and it can separate the provider and the requester on its interface. Finally, there are some principles of good service design that are enabled by characteristics of either Web services or SOA as seen on the table below.
Table 3 - Web Services vs SOA. The principles that are enabled by each one

<table>
<thead>
<tr>
<th>Enabled by Web services</th>
<th>Technology neutral</th>
<th>Endpoint platform independence.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Standardized</td>
<td>Standards-based protocols.</td>
</tr>
<tr>
<td></td>
<td>Consumable</td>
<td>Enabling automated discovery and usage.</td>
</tr>
<tr>
<td>Enabled by SOA</td>
<td>Reusable</td>
<td>Use of Service, not reuse by copying of code/implementation.</td>
</tr>
<tr>
<td></td>
<td>Abstracted</td>
<td>Service is abstracted from the implementation.</td>
</tr>
<tr>
<td></td>
<td>Published</td>
<td>Precise, published specification functionality of service interface, not implementation.</td>
</tr>
<tr>
<td></td>
<td>Formal</td>
<td>Formal contract between endpoints places obligations on provider and consumer.</td>
</tr>
<tr>
<td></td>
<td>Relevant</td>
<td>Functionality presented at a granularity recognized by the user as a meaningful service.</td>
</tr>
</tbody>
</table>

Enterprise Service Bus

Enterprise Service Bus, as known as ESB, is defined as a middleware that uses service-oriented architecture to provide fundamental services for more complex architectures. The duties of an ESB include monitoring and controlling routing of message exchange between services, controlling deployment and versioning of services, etc. About ESB architecture, it is divided into Service Consumer, the ESB and the Service Provider.
The ESB provides several benefits when it is used. The benefits include increasing flexibility, which means that it is easier to change according to the requirements; distributing information across the ESB is quickly and easily. In addition, ESB requires more configuration rather than integration coding, and it can normalize a number of protocols to a single protocol.

However, when should ESB be used? With several benefits, it is normal to think that an ESB should be used whenever is possible. Although this may be true, using ESB without criteria can become a problem because it can increase overhead, and data replication is not done in an efficient way. Furthermore, if large volumes of data is being sent by the bus as a large number of individual messages the ESB will have a slower communication speed. Therefore, ESB should be used in appropriate scenarios such as when three or more applications need to be integrated, or with services that are going to be incorporated from external service providers, or when a number of protocols need to be normalized in a single protocol.
Additionally, there are some scenarios for the practical use of an Enterprise Service Bus. These scenarios are often present in the applications that are used every day by many users. The symbols in the figure below are going to be used to describe the scenarios.

![Symbols for an ESB](http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html)

**Scenario 1 – Secure Message Processing**

When a system needs to forward messages to another system the ESB can be used to perform this task. In this scenario, the ESB will get the message from the queue, forward it to a Web Service and then the ESB will send to the destination system via a DB adapter.
The message flows are configured in a distributed XA transaction as additional participants. It means that the transaction starts when the ESB gets the message from the queue, and comprises the database operations. When the message flow is completed successfully, the next step is the committing of the distributed transaction.

Scenario 2 – Service Versioning

Services may need to be changed for a variety of reasons that include changing business needs, fixing bugs, or to address other issues. The ESB can be used in these cases to perform the transformation from the old version to the new version of the service. A practical scenario can be seen on the figure below.
On the figure above, the service was upgraded to the 2.0 version and the Consumer B got the newest version. On the other hand, the Consumer A does not want to upgrade because he/she has been using the interface 1.0 and it works perfectly to his/her needs. However, the service provider will not keep running the two version of the service because it may be difficult or not technically possible.

The ESB can simplify the situation if it delivers the version 2.0 directly via a pass-through. The version 1.0 is kept on the ESB and it is not called from the provider. The messages are transformed from version 1.0 to version 2.0 and sent to the new service. A good example of service versioning is the new version of the Internet Explorer, which has a compatibility view button to adapt the websites that were developed to be compatible with the old versions of the browser.

**Scenario 3 – Service Virtualization**

Service Virtualization is the ability to simulate the behavior of specific components in a general component-based application such as API applications, cloud-based applications and service oriented architectures. An ESB it is an elegant solution to virtualize the endpoints in a service virtualization process. A practical scenario can be seen on the figure below.

![ESB Diagram](http://www.oracle.com/technetwork/articles/soa/ind-soa-esb-1967705.html)

On the figure above, the provider is proving a Web Service that is being used by the consumer by the ESB and not directly. The ESB delivers the web service exactly as it looks to the consumers. In addition, the ESB can address any changes that may be made on the endpoints and then the service consumers can continue to run as before. Another role that the ESB can perform is monitoring functions, and if the service provides makes a change to the service contract, a simple transformation of the exchanged messages is enough to not impact the service consumer.

**ESB Vendors**

An Enterprise Service Bus should be seen as an architecture style and not as a product. However, several companies provide the ESB as a software for the companies that need to use it. The most popular ESB vendors on the market are Oracle, IBM and TIBCO.
Oracle is on the market with the Oracle Fusion Middleware, IBM with the WebSphere Enterprise Service Bus, and TIBCO with the TIBCO ActiveMatrix Service Bus. All of them offer the ESB basic capabilities such as support of multiple protocols, protocol conversion, data transformation and data-based routing, support of composite services, support of multiple standards, extensibility, etc. In addition, they offer a set of extended functionality that includes graphical editing tools, SLA monitoring and management, BPEL and other business process support, business activity monitoring, dynamic service provisioning and complex event processing.

These three vendors are considered as the best on the market. Their products include all the capabilities of ESB, and include very broad feature sets that include support for many protocols, interaction models, file formats, error handling features, and more.

Conclusion

Services have become more used for applications. The goal of Service Oriented Architecture is to provide a way to make the communications between the services. SOA has many features and benefits that make the use of SOA be essential to deliver the business agility and IT flexibility by Web Services. However, SOA and Web Services are always seen as the same thing. Web Service is the most common implementation of SOA, but SOA is much more than just the Web Services. A good way to distinguish SOA and Web Services is thinking that SOA is the architectural pattern while Web Service is a way to implement SOA.

Another tool that uses the SOA model to promote interoperability between the services is the Enterprise Service Bus. ESB is a middleware that does the mediation and integration between environments and it has several duties and benefits that are provided when an ESB is used. Although that ESB looks like a software, it should be seen as an architecture style or pattern because there is no standard of ESB. In addition, ESB should be used with criteria, and in appropriate environments, otherwise there are several disadvantages that will make it inefficient. Finally, on the market, there are several companies that provides ESB and they should be carefully analyzed for a company that wants to buy this kind of service.

References


