Chapter 5. Architectures for Enterprise Level

Table of Contents

Extensive advancements in enterprise computing has been taking place in the last decade and with the advent of the Web it is now possible for organizations to automate and integrate businesses and computer operations. Almost all enterprise organizations face the problem of integrating different applications and database systems at some point. Research has shown that during software development, a third of the time is dedicated to the problem of creating interfaces and points of integration for existing applications and data stores. In situations like this enterprise application architecture becomes very important. **Enterprise application architecture** allows an enterprise to integrate its existing applications and systems and to add new technologies and applications to the mix.

With the emergence of the Web EAI has gone beyond just merging applications within enterprises. Provision of services through the web has rapidly become a trend. External or internal systems, e.g., existing applications, existing databases and interfaces to external business partners, can be integrated into Web applications. There are two patterns that EAI systems implement:

- Mediation (intra-communication):

  Here, the EAI system acts as the go-between or broker between multiple applications. Whenever an interesting event occurs in an application (for instance, new information is created or a new transaction completed) an integration module in the EAI system is notified. The module then propagates the changes to other relevant applications.
- Federation (inter-communication):

  In this case, the EAI system acts as the overarching facade across multiple applications. All event calls from the 'outside world' to any of the applications are front-ended by the EAI system. The EAI system is configured to expose only the relevant information and interfaces of the underlying applications to the outside world, and performs all interactions with the underlying applications on behalf of the requester.

Both patterns are often used concurrently. The same EAI system could be keeping multiple applications in sync (mediation), while servicing requests from external users against these applications (federation). Enterprise Application Integration is related to middleware technologies such as message-oriented middleware (MOM), and data representation technologies such as XML. Other EAI technologies involve using web services as part of service-oriented architecture as a means of integration. Enterprise Application Integration tends to be data centric. In the near future, it will come to include content integration and business processes. For our point of view, the service-oriented architecture is the one which plays an important role, while this is the most frequently used in Web context.

Service-oriented architecture

In the IT industry a frequently used buzzword is ‘Service Oriented Architecture (SOA)’. The Web, IT literature and other resources provide different definitions and interpretations on SOA. Some refer to it as a “Sophisticated Product”; some as an “Architectural Approach” while others treat it as a mere marketing gimmick. Service-oriented architecture is somehow like a design pattern that consists of discrete pieces of software with some functionalities and other applications can utilize these functionalities. This pattern does not require us to use some specific product or a platform. A service provides some functionality and this can be used by other large software applications to complete its use.

In layman terms, Service Oriented Architecture provides a framework where different (Heterogeneous) platforms (Windows, Linux, and UNIX), technologies (.NET, Java) and applications (ERP, CRM, and SCM) operate in synchronization. More specifically, this is achieved through “Services” and “Web based Open Standards”.

Service is the base of this architecture. Services are the structures which have ability to interact with each other. In other words they are the listener of the other side of the phone which is an endpoint.

In the classical layered architecture layers interact with each other. This interaction and hierarchy of this system should be constructed very properly for this system to work proper. Therefore we can say that SOA simply means that these layers are created as services. For example if we make a service which provides us the data, then we have accomplished the functionality of data layer by this. Afterwards we can also use this service from other applications as well. Achieving this we have a flexible architecture to use.

Why SOA?

Reusability

This property of SOA is the most important benefit of this architecture. Base of this architecture is constructed with services so that it provides us functionality to multiple applications or clients. The purpose is actually to reuse them.

Connecting Heterogeneous Systems

Nowadays in our software world its almost impossible to create a common infrastructure. What I mean by this is technologies can be different and even if they are same the versions of them might be different as well. Thus SOA provides us a good communication quality.

Loose Coupling

Services are not connected with classes or libraries they are connected with contracts and endpoints. So that they are loosely coupled. A service which obeys the same schema structure can replace another service very easily. To give an example for this we can say that a Java service can be replaced with a .NET service or another service which obeys the same schema.

Facade

This architecture can be use to simplify interface of complex structures or complex services. This type of approach is used when the service is created to simplify technical usage for other developers to use. It can be also used to separate the service from work flow. It is enough to expose only the relevant data to the consumer.

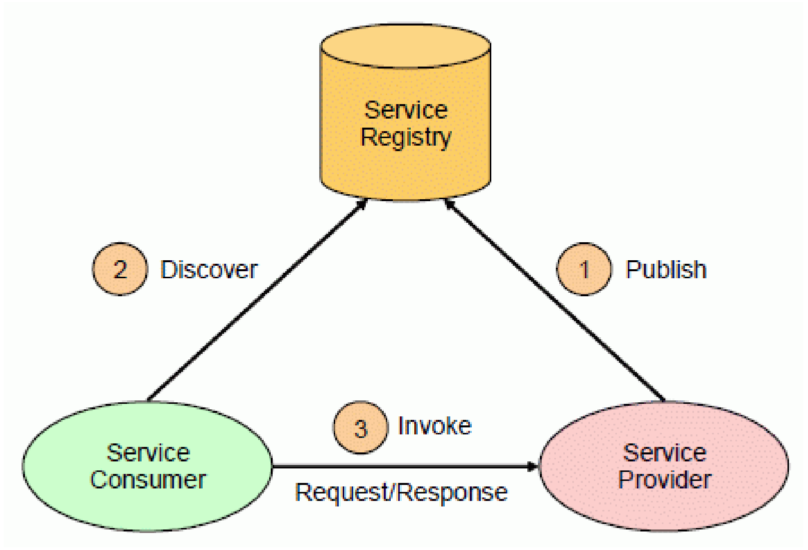
Abstraction

Every service is responsible for their own functionality. For example : If you want your service to produce passwords, than it only produces passwords while another service checks the username and password. In this way application is partitioned into simpler parts. This increases the readability and reduces the time of maintenance.

Where to use SOA then ?

This architecture should be preferred when there are multiple heterogeneous systems, applications or consumers. Otherwise achieving flexibility is an additional cost. Consider the scenario where Enterprise systems have different owners and multiple teams working on them. In order to improve visibility and accessibility, centralized control is required. The Service Oriented Approach is appropriate in such a scenario as it provides complete information about the system components through its Service Registry.

The common service-oriented architecture is illustrated in the following diagram:



Service-oriented architecture

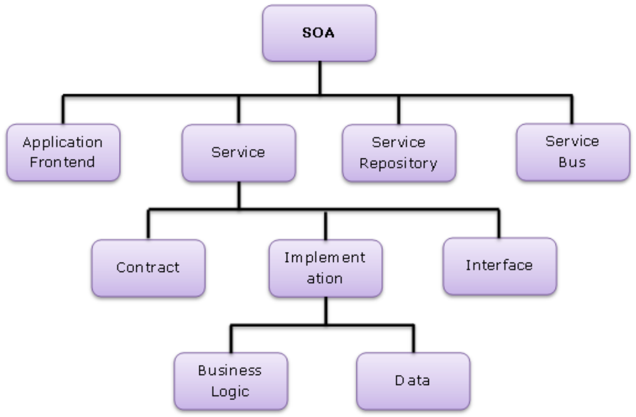
Service provider is responsible for providing the services and the details of the services. The service provider can decide whether the service needs to be secured or can be used by anyone. The cost implication and the traffic of using the service needs to be thought about. There will be an interface provided by the service provider so that any service that needs to be accessed can be achieved with the help of the interface.

The service provider can decide whether the services needs to be listed or not and what should be the agreement that should be set between the consumer for accessing the services. Most of the cases the service provider publishes the interface and the access details with the Service Registry

Service Registry or broker implementers should carefully think and decide on the implementation and access strategy of the services by the service consumers. The ownership of making the service interface public resides with the service registry. The implementers should consider about the scope that is involved. There are pubic service registries that can be accessed over the internet and there are private service registries which is accessible to only restricted or role based consumers

Service consumer ensures that it’s possible to locate the service that is registered in the service registry and binds to the service provider is obtained. The service consumers invoke the service that is defined.

Elements of SOA:



SOA elements

These architectures have a number of benefits, including quicker and more cost-effective responses to changing market conditions. SOAs can also simplify connections to legacy IT systems. SOAs also have a number of challenges in implementation, such as managing the large number of services and how they interact with each other. There are also concerns about lack of testing and security levels of services. Overall, service-oriented architecture is a widely-used technology that is changing the way large businesses function today.

Principles of SOA:

1. Explicit Boundaries

2. Shared Contract and Schema, not Class

3. Policy-driven

4. Autonomous

5. Wire formats, not Programming Language APIs

6. Document-oriented

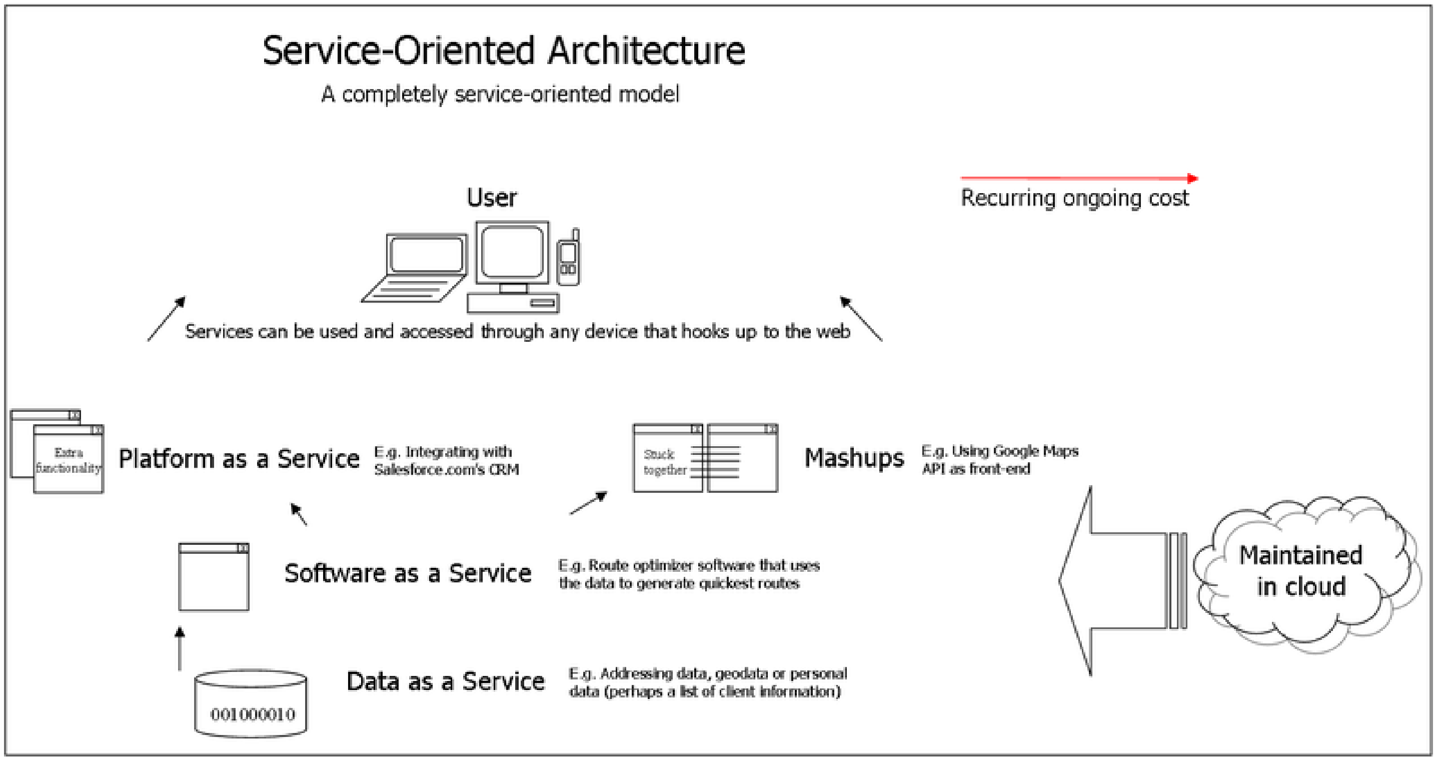
7. Loosely coupled

8. Standards-compliant

9. Vendor independent

10. Metadata-driven

The following figure is a good description about how could we imagine a completely service-oriented model:


Service-Oriented Architecture
A completely service-oriented model
User
Recurring ongoing cost
Services can be used and accessed through any device that hooks up to the web
Extra functionality
Platform as a Service
E.g. Integrating with Salesforce.com's CRM
Stuck together
Mashups
E.g. Using Google Maps API as front-end
Software as a Service
E.g. Route optimizer software that uses the data to generate quickest routes
Maintained in cloud
Data as a Service
E.g. Addressing data, geodata or personal data (perhaps a list of client information)
001000010

SOA overview

Side note about Web-oriented architecture

Note

According to Gartner's definition of Web-Oriented Architecture (WOA) from 2008:

> WOA is an architectural substyle of SOA that integrates systems and users via a web of globally linked hypermedia based on the architecture of the Web. This architecture emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints:
>
> 1. Identification of resources
> 2. Manipulation of resources through representations
> 3. Self-descriptive messages
> 4. Hypermedia as the engine of application state
> 5. Application neutrality

Resource identification could be done by uniform resource identifier (URI), resource manipulation by HTTP, application state by links and self-describing messages could be identified by Multipurpose Internet Messaging Extensions (MIME) types.
If we familiar with REST than we can feel this is essentially a rehash of REST except for the fifth point on "application neutrality". After some back-and-forth discussions on the REST-discuss list [WOAvsREST], it turned out that "application neutrality" meant using generic media types such as the Atom Syndication Format for representations. Media types should be able to convey the type of representation, and by using generic types, applications lose describability.
It is unfortunate that Gartner chose a vague term like "application neutrality" to describe this. But on the positive side, they are at least talking about REST. Furthermore, Gartner archived the document in 2011 because some of its content may not reflect current conditions and said that its more easier to speak about Web APIs.
[GartnerWOA] *Tutorial: Web-Oriented Architecture: Putting the Web Back in Web Services*. Web page.
[GartnerBlogWOA] *WOA: Putting the Web Back in Web Services*. Web page.
[GartnerSOAgovernance] *Magic Quadrant for SOA Governance Technologies*. Web page.
[WOAvsREST] *Yahoo! groups: Gartner note on WOA*. Web page.
[RESTthesis] *Architectural Styles and the Design of Network-based Software Architectures*. Web page.

Prev Up Next
The MVC pattern - useful but not a silver bullet Home Representational State Transfer (REST)