Service-Oriented Architecture (SOA) vs. Component Based Architecture

Helmut Petritsch

Index

Introduction

SOA is a buzzword and topic for many discussions in nearly every professional journal and conference. Opinions differ from “some technical rubbish” to “the technology of the future”. The problem about these different opinions partly depend on another problem: What precisely is SOA? Whatever it is and will be, it seems to become the next step in the evolution of software architecture.

Probably SOA is mostly linked with Web Services: They are mostly used to open existing architectures and systems on specific points and allocate them via HTTP or HTTPS. This procedure features mainly one thing: An easy way of B2B connection.

Because of the loose coupling of Web Services are optimal for reuse. This idea results in the (old) idea, of building a “legobox” of interoperable, reusable services. All we need next is an architecture that allows to composite the existing Services. And this architecture could be SOA.

But wait: Module, composition, reusability: Isn’t that an old hat? What’s the difference to Enterprise Java Beans or component based architecture? Isn’t that the same idea? It probably is. But there are a few details, which make a big difference. This paper has been written to figure out these differences.

Definitions

Component-based architecture:

A component is a software object, meant to interact with other components, encapsulating certain functionality or a set of functionalities. A component has a clearly defined interface and conforms to a prescribed behaviour common to all components within an architecture.[1]

The goal of generative and component-based software engineering is to increase productivity, quality, and time-to-market in software development thanks to the deployment of both standard componentry and production automation. One important paradigm shift implied here is to build software systems from standard componentry rather than "reinventing the wheel" each time. This requires thinking in terms of system families rather than single systems. Another important paradigm

[1] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/

shift is to replace manual search, adaptation, and assembly of components with the automatic generation of needed components on demand. Generative and component-based software engineering seeks to integrate domain engineering approaches, component-based approaches, and generative approaches.[2]

A component model is probably used for the developing and executing of components. This model defines a framework, which defines structural requirements for connection- and composition options as well as behaviour-oriented requirements for collaboration options to the components. Beyond that a component model provides an infrastructure which implements frequently used mechanism like persistence, message-exchange, security and versioning. The idea is to build exchangeable software units through clearly defined interfaces. Different manufactures offer platforms like DCOM, JavaBeans, Enterprise JavaBeans, and CORBA.

Service-oriented architecture (SOA)

There are a lot of definitions of Service Oriented Architecture, some of which are:

A service Oriented Architecture is a set of components which can be invoked and whose interface descriptions can be published and discovered. [3] (W3C)

SOA is an architectural style whose goal is to achieve loose coupling among interacting software agents. A service is a unit of work done by a service provider to achieve desired end results for a service consumer. Both provider and consumer are roles played by software agents on behalf of their owners.[4]

Service-oriented architecture (SOA) is an architectural software concept that defines the use of services to support business requirements. In an SOA, resources are made available to other participants in the network as independent services that are accessed in a standardized way. Most definitions of SOA identify the use of web services (using SOAP, WSDL and UDDI) in its implementation; however it is possible to implement SOA using any service-based technology.

Unlike traditional object-oriented architectures, SOAs are comprised of loosely joined, highly interoperable business services. As these services are interoperable via different development technologies (such as Java and .NET), the software

[2] http://www-ia.tu-ilmenau.de/~czarn/generate/engl.html
[3] http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/
[4] http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

components are very reusable. In this way, SOA shares fundamental similarities with CORBA.[5]

The variety and inaccuracy of the definitions of the Service-oriented architecture makes it hard to say if SOA is only accessible with Web Services or if for example interfaces for Enterprise JavaBeans can also be “invoked, published and discovered.” These possibilities will not be excluded to consider similarities and varieties of the Service-oriented architecture and component based architecture (like the referred Enterprise JavaBeans) in more detail.

Web Services can be provided for general use like public web sites and published on the “yellow pages of Web Services”, the registries, working with technologies like UDDI (Universal Description, Discovery and Integration).

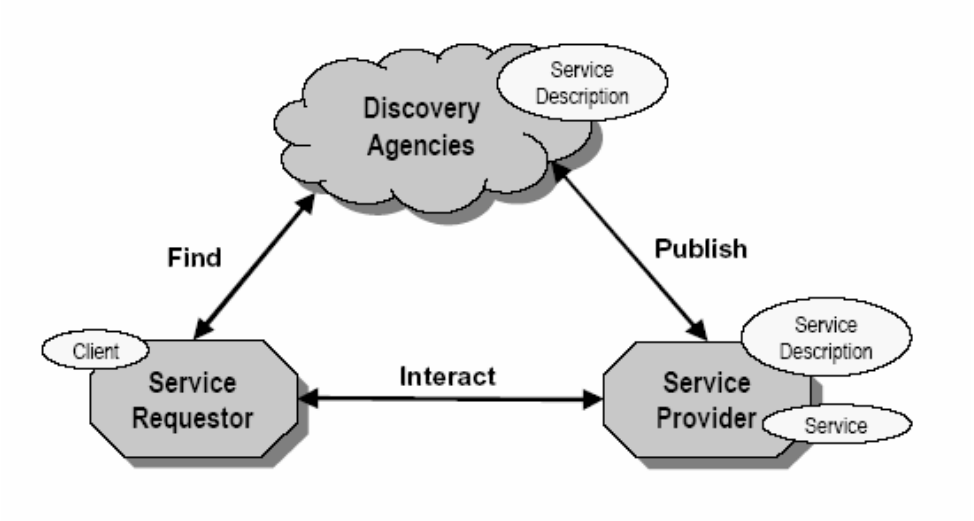

Figure 1: Finding and interacting with public Web Services

Figure 1 shows the principle of the usage of public Web Services. The Service provider publishes the Web Service at a discovery agency. The potential service requestor searches for a service at the discovery agency, acquires the URL of the required service, gets the WSDL file, builds the client and uses the provided service.

[5] http://en.wikipedia.org/wiki/Service_Oriented_Architecture

Evolution vs. revolution

Component based architectures and Service-oriented architectures seem to have the same goal: To provide a foundation for loosely joined and highly interoperable software architecture, enabling efficient, error-free software development.

Nearly all evolution in recent years had this intention: To develop a type of architecture, that allows loose coupling and high reusability of its components. These attributes should allow more efficient, faster, error-free software production. In more abstract terms, one evolutionary step enhanced the previous step and helped to get closer to these objectives.

That's why I am calling the step from Component-based Architecture to Service Architecture an evolution and not a revolution.

An additional example, why this step is an evolution step and why there are no clear and absolute boundaries between Service-oriented- and component based architecture is the following.

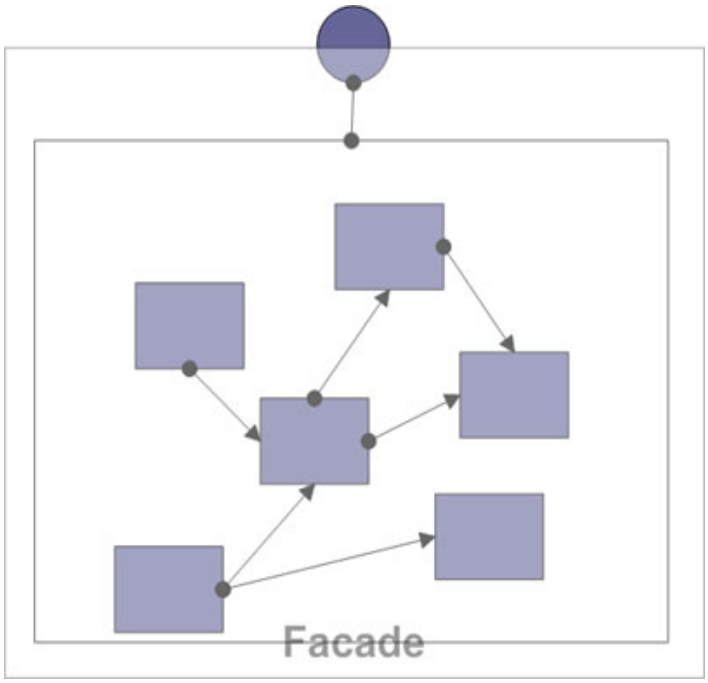

Figure 2: The Façade pattern

Although with component based software development exits a bunch of technologies for object distribution, it's still not possible to distribute fine-grained objects without causing a measurable impact to at least some of the non-functional requirements. Since local method invocation is still much faster than a remote one, only coarse-

grained objects should be exposed to the network. However, it's hard to reuse coarse-grained objects, so the reusable business logic should remain fine grained.[6]

A solution to this problem is the façade pattern: We do not want to publish the fine grained entities to the client, so we have to provide a coarse view of them. On the other hand, we do not want to change the interface of the entities, so we have to provide an additional element that provides a distributable view of the system.

The façade provides such a view to the system the clients can work with. Various façades achieving different demands can be designed. This pattern is similar to a fundamental idea of Web Services: To open the system on specific, precisely defined points for clients or business partners.

This example should demonstrate that there are no clear boundaries between the two architecture types; a lot of characters of SOA come from component based architectures or from hereon constitutive design patterns.

[6] http://entwickler.com/itr/online_artikel/show.php3?nodeid=97&id=554

Loose Coupling and the possibility of publication

What's new about SOAP?

One of the most significant characteristic of Web Services is their loose coupling. They are not only independent from the infrastructure they are running on. It doesn't make any difference (in contrast to Component-based Architecture) if one service using another doesn't run on the same computer, or doesn't use the same language/operating system. Web Services released with .NET can easily be coupled with Web Services written in Java, as if both were Java or .NET builds.

In addition, they can be exposed to the internet so that everyone can use components he is not the owner of, without taking them away from their owner, or having to copy them. Web Services are designed to be published as far as possible like web sites. They should be like web site for machines and computers.

A popular example is writing a Web Service (or only a client), that combines the two Web Services of Google and Amazon: It could find books on a specific topic at Amazon and provide more information from Google. This example shows one more characteristic of SOA: Services can be combined, that haven't been designed to be combined. Services that have been developed for an appointed reason or function and are released on the web can be used for other purposes than intended. The fact that services can be used for other purposes can be an advantage and a disadvantage. It can be an advantage, if the service is used in a new, innovative way, probably help the provider to reach a new business or new client. It would be a disadvantage if the service is abused harmful for the provider. Not everything should be published, and usage should be regulated by general terms and conditions or the like.

This usage for other purposes than intended is only possible because of loose coupling. Google service doesn't know anything about Amazon service and vice versa, both of them are self-sufficient. Well, with Enterprise JavaBeans there can also be two Components in a Container that know nothing about each other and the interfaces are usable by a client. But loose coupling is not loose coupling, there is no boundary, where something is loosely coupled or is not. A lot of technologies are loosely coupled, but each has a different shade of loose coupling. For example, have you ever come across an EJB developer, using an EJB from another company, without having to get in contact with this company? That's only possible with Web Services.

Why not CORBA for SOA?

The differences between component based architectures, for example CORBA, and Web Services are not as big as it seems from this position. CORBA has standardised interfaces and a standardised protocol too. CORBA opened nearly the same facilities as Web Services and probably failed only because of the attempt to establish a new protocol for CORBA (IIOP), instead of using an existing one. Besides, during the standardisation it has been forgotten to determine the port of the communication. Thereby network administrators had to open a new port for every CORBA communication partner in the firewall. Furthermore CORBA spawned mistrust at network administrators because of the binary stream, impossible to sniff in a reasonable way, passing all security facility straight in the software heart of the company. Web Services in contrast use an established protocol (HTTP, HTTPS) and data format (XML, Schemata).

SOA and SOAP

SOA doesn't mean SOAP. It's possible to develop service oriented with other techniques except SOAP. Only if you have to provide the service for third parties, will Web Services be the best choice. Service orientation doesn't require Web Services, a much better performance can be achieved with other technologies like JMS, EJB or CORBA, because they use a binary protocol with a lower data stream and no need to parse the data. JMS provides additional possibilities for reliable messaging.

So the advantage of Web Services "only" is that they provide services for third parties on the Internet. But especially this point is probably necessary to afford and develop new kinds of business ideas. Essential conditions to reach this goal are companies like Amazon or Google, having the courage to publish such services on the web. Only if other companies follow the example of Amazon and Google, will there be a possibility to develop new products utilising the potentials of Web Services and produce some "real service oriented" software. Otherwise the advantages of Web Services over Component Based Software would not really be used.

Confine SOA vs. components

There is no clear dividing line between Service Oriented Architecture and Component Based Architecture. In principle SOA is the enhancement of Components: The individual services are single components, which can be linked to gain new business logic, new services or a new component. The big difference is the connection

between and the possibilities of offering single services for third parties. For example, EJBs (especially Session Beans) can be designed to offer its business methods like services in a context free way. These services of this EJB can be used by other EJBs or clients. In a big company (or a coalition of parties gaining access to each others EJB's) single departments could offer their services (in the shape of the business methods of the EJBs) for other departments, so that the same effect could be achieved as with services supporting SOAP: The business methods of the EJBs represent the activities one department offers. Other departments could use these services for their belongings and perhaps use them in a way the business method wasn't built for. So, this usage of Enterprise JavaBeans could be seen as service oriented too.

SOA – a miracle cure?

SOA seems to be a good step forward to looser coupled software, higher reusability, faster development and probably a completely new style of software development. The difference between SOA und components seems to consist of two major points:

- Services have to be publically accessible. Models for consumption will probably be developed though not necessarily cost free. But through registries (UDDI) it should be possible to find services like other business partners in the yellow pages.
- Services have to be largely independent from implementation specific attributes. For users and customers it is irrelevant, if the service is released with Java, .NET or Perl. The shared communication is XML based, and as long as no other protocol exits, the protocol will probably be SOAP.

The vision so far involves a building set of services and developing a new architecture, providing an easy way of merge available services. But this will raise a bundle of problems, which didn't use to exist with "traditional components".

Gained Web Services mean bad performance

As a rule following the paradigm of service oriented architecture most Web Services will partly use other Web Services, which perhaps themselves use other Web Services. But this chain must not get too long, because too much to fine graded services will reduce performance and are costly to service. If non-public services are used, they can be integrated with other communication technologies like JMS. This means, you have to design a JMS-Web-Service bridge which has to transport security- and transaction issues.

A jungle of standards doesn't make it easy

But performance is only one factor that has to be regarded. Another criterion could be affording a high flexibility and reusability, which asks for a loose coupling.

To reach a maximum degree of loose coupling, Web Services shouldn't call each other mutually. The standard BPEL (Business Process Language) achieves this challenge for Web Services. BPEL is a meta-like language, defining the actions of

calls from different services per XML. The Web Services themselves should be defined by a WSDL interface, which should be as self describing as possible, to be used by BPEL in an easy way. But this raises the next problem: it's not possible to use JMS with BPEL. In addition a service locater (something like UUDI) should be developed; security issues have to be integrated, for example with SSL or SAML. Single Sign-on could be released with the help of the Liberty Alliance, transactions with the help of WS-Transactions. The must to evaluate, test and use all these technologies is supposed to be a reason for many companies to retard or avoid the usage of Web Services. To give Web Services a realistic chance there has to be an "all-in-one" solution to all these problems.

Finding the right Web Services

Finding an existing Web Service is a non-trivial job. Technologies like UDDI afford the basic necessities. But it seems that this is still one big problem of Web Services: The publishing and finding of Web Services in an efficient and competitive way. One might also write undiscoverable services oneself; but that's against the idea of SOA: If you have to write all the services yourself, there's no real advantage to component based solutions, like with a stateless session EJB.

Are there guarantees for Web Service users?

Using Web Services means to become dependent on these Web Services. If there's nothing like a contract or the like a company might have to face major difficulties if one ore more Web Services aren't can't be reached any more.

Apart from this worst case, there may also be smaller problems. Interfaces (and thus i.E. WSDL-files) can change, the service probably provides slightly different services; the code has to be adopted manually which causes additional costs.

Quality of Service of foreign applications

For commercial web services the web services used have to be analysed with regard to the quality of service. Non-functional attributes like performance, reliability, security, and manageability have to be detected. If possible, there should be metrics to decide if a foreign service satisfies the needs of one's own software and company.

Data overhead and lower performance though XML/SOAP

The usage of XML as data format not only leads to platform independence. In contrast to binary data transport mechanism, XML has a higher need of data to transfer, resulting in lower performance and higher usage of network and internet traffic. SOAP is the de facto wire protocol for Web Services. But SOAP performance is degraded because of the following:

- Extracting the SOAP envelope from the SOAP packet is time-expensive.
- So is parsing the contained XML information in the SOAP envelope using an XML parser.
- With XML data not much optimization is possible.
- SOAP encoding rules make it mandatory to include typing information in all the SOAP messages sent and received.
- Encoding binary data in a form acceptable to XML results in overhead of additional bytes added as a result of the encoding as well as processor overhead performing the encoding/decoding.[7]

But not only data overhead results from the usage of XML: The parsing of the XML messages takes more time than serialising and deserialising from data sent in binary format over the network.

[7] http://www-106.ibm.com/developerworks/library/ws-quality.html: SOAP and performance

Conclusion

An exact differentiation between Service oriented architecture and component based architecture is hard to make, because opinions on what “SOA” exactly is and how it will develop differ.

If SOA is seen as a new type of architecture that defines the how-to of assigning interfaces in a servicing way so that these services can be used in a context free way, it doesn’t differ significantly from existing component based frameworks like Enterprise JavaBeans.

If the definition of SOA includes the usage of technologies like WSDL, UDDI, and SOAP (and its potential successors), SOA differs in several ways from the “old” component based architecture. With these technologies software can be built in a completely new way. Software developers can use foreign, external “components” in the form of Web Services. Web Services can be used in contexts that weren’t considered at the time they were built.

But SOA is not the solution to all problems linked with software development. There are a lot of problems: Ranging from finding the required services, providing acceptable performance, security, realising transactions up to maintaining one’s own service, even if foreign, integrated services have changed or are closed.

There are a lot of problems to resolve, but there are a lot of possibilities too. It will depend on Sun or other larger companies, to develop an overall solution, containing solutions to all of these problems.