

Service-Oriented Architecture, Where do we stand?

Mehran Nikoo, Dunstan Thomas Consulting
<http://consulting.dthomas.co.uk>



Service-Oriented Architecture (SOA) is expected to dominate IT industry in the upcoming years. It all began when people started to use the term “software as a service”. In “software as a service” model, you design your software so that it can be used by other systems. Then people can subscribe to your service and use it by demand. It is like subscribing to a Cable TV service. As long as you are connected to the service, you can use it whenever you want.

Well, for a long time, most of us developers have been writing modular code so that it can be used by other software. The difference is the user base and audience of course. Back to our Cable TV example, we are talking about the length of wire. If your Cable TV technology allows you to broadcast the programs only in a building where you make the programs, you cannot find lots of subscribers. The larger the user base, the better for you as the service provider.

These days the ideal user base for software is the Internet, but writing such software is a challenging task and is some cases not possible at all. In this article we are going to see how successful the concept of SOA has been. I will highlight some issues that need to be considered when implementing a solution based on SOA. We are also going to see how .NET framework can help developers to implement successful SOA-based solutions.

I assume that the reader has a basic understanding of at least one component-based technology like COM+, CORBA or RMI. A basic knowledge of web services and .NET will be advantageous.

What is SOA?

As its name suggests, it is about “Architecture”. So we are talking about a high-level infrastructure that is based on best practices and patterns for creating service-oriented solutions. Maybe you are using it already, but you are not aware of it.

Is it a new concept?

Software Engineers have been always told to write software that has *high cohesion* and *low coupling*. Giant software vendors have built technologies that allow developers to achieve this goal. COM+, CORBA and RMI are some attempts to provide a solution to this problem. So it is not a new concept. It is rather another attempt to achieve the goal of writing software with high cohesion and low coupling.

“Why another attempt? Haven’t we succeeded yet?” one might ask. Introduction of component-based technologies soon changed to a war between their vendors as each of them wanted their solution to dominate the market, a dream that never came true. Lack of interoperability between systems reduces the level of cohesion between components, which is not good.

The concept of “high cohesion, low coupling” can become too vague when it comes to evaluating a solution or technology. Also one can easily mix up the idea of cohesion with coupling! To prevent such confusion, you can use the characteristics of SOA as a way of evaluating the quality of software in regards to cohesion and coupling. Although those concepts are not exactly the same, but systems that are based on SOA will show the characteristics of a system with “high cohesion and low coupling” and vice versa.

Characteristics of SOA-based solutions

Let's have a look at some characteristics of software, which is based on SOA:

- The client of the service should not have to know about implementation details of the server.
- Location of the server should be transparent to the client. It is only at run-time that client will know the location of the server.
- Software should be able to interoperate with other software running on other platforms.
- Multiple versions of server software should be accessible at the same time. Remember that clients are not necessarily located in a single department or organisations and you cannot expect them all to upgrade their software at the same time.

There are also implicit requirements that we expect from any solution:

- Performance
- Security (access control, encryption, temper proofing)
- Availability
- Reliability: This is especially important when it comes to transaction processing,

Web Services serve SOA

Evolution of XML was followed with the introduction of web services. A web service is the best solution to implement a SOA as far as the user base is concerned. You will be relying on HTTP, which is the most accessible protocol on the Internet. By looking back at characteristics of SOA-based solutions, you will see that web services satisfy most of the requirements. But they compromise the following factors for the others:

- Performance: XML which is the building block of web services is an overhead as compared to efficient native mechanisms to represent data. So web services can be easily challenged because of their performance.
- Reliability (Transactional Integrity): If you are providing a service that should be part of a transaction, how can you maintain the integrity of transactions over the Internet using HTTP? For example in Microsoft world you had the option of using Distributed Transaction Coordinator (DTC) to achieve this.
- Security: How do you authenticate users over the internet and authorise them to use the service?

Another negative point about web services is the level of support that developers get from existing IDEs. When you use a class in existing IDEs, you have access to Meta data for the class. It allows IntelliSense feature to provide assistance to developer. Also compiler can detect any errors in the code at compile time, and not the run time.

So unless all of the above issues are resolved, there is a very good argument why web services should not be used everywhere. You should have very good reasons to implement a web service interface for your software. Even if you do, you should consider providing a more efficient way of connecting to your service if many internal clients in your organisation will be using the service. For example you can consume your service using COM+ inside your organisation, and provide a web service interface for clients that cannot use COM+ interface because they are behind a firewall or running on another platform.

Preparing for SOA

Well, I mentioned some negative points for web services, but they are resolvable. For instance performance issue can be addressed by introduction of faster processors but it may take a while to get rid of all those issues. Although you may decide not to move to a service oriented architecture yet, but you need to prepare for it anyway. It means that you should start the way you think about and implement components. Here is a list of things to consider when architecting software:

- Interact with the servers using a chunky interface rather than a chatty one. In other words, try to combine different methods calls as a single method call as much as possible. If you are sending a message over the Internet, every byte counts and it is better to avoid the overhead associated with extra bytes in the header and footer of each message, as well as the time required for name resolution and routing.
- Try not to keep too much state on your servers. You can delegate this task to the clients instead. This is as if you are working for an organisation, which has lots of clients and each client has got lots of documents and folders. If you want to keep all that data, you will need a very big warehouse, which costs you money. So it is better to ask the clients to keep those documents themselves. The alternative is to store the state in a database or another machine which is dedicated to this task. Back to the above example, you may decide to outsource this operation to a third-party company, and use a ticket to identify and retrieve information about your customers.
- Have very well defined interfaces for your services. When you move to the web services world, you will need to publish the interfaces to your clients.
- Join the asynchronous world. Many of the servers are serving the clients in an asynchronous fashion. So it is better for the clients to use the same approach. It is not a good idea to block the execution of the client for receiving a response when the server even has not started the processing. Try to change your view of request/response paradigm to a message based system where client sends a message to the server, and server will send a message back to the client sometime later.
- Think of an alternative way of authentication and authorisation. Security mechanism is different in web services world. Do not rely on platform-dependent security mechanisms since it will be of no use when you are interoperating with another platform. There is a security extension to the web services basic profile called WS-Security, which specifies the way security can be implemented in the web services world. We will have another visit to this subject later in this article.
- Choose a platform that allows you to keep existing versions of the same component in parallel. This way you can have a separate interface for each version, which allows you to provide your service to various clients using different versions of your service.

Component technologies like COM+, CORBA and RMI have been a success since there are so many systems out there using these technologies. Introduction of web services addresses some issues which are not addressed by those technologies, but web services technology is not replacing any of those technologies in the short term. So you should consider using them in parallel to get the best of both worlds.

.NET and SOA

Currently .NET has the broadest support for web services profiles. Web services basic profile consists of XML Schema 1.0, SOAP 1.1, WSDL 1.1 and UDDI 1.0. This profile is supported by both .NET 1.0 and J2EE 1.4. Microsoft released Web Services Enhancements 1.0 (WSE) in 2002, which adds the following profiles to the list of supported profiles in .NET:

- WS-Security: Describes enhancements to SOAP messaging to provide quality of protection through message integrity, message confidentiality, and single message authentication.
- WS-Routing: WS-Routing is a simple, stateless, SOAP-based protocol for routing SOAP messages in an asynchronous manner over a variety of transports like TCP, UDP, and HTTP.
- DIME: Direct Internet Message Encapsulation (DIME) is a lightweight, binary message format that can be used to encapsulate one or more application-defined payloads of arbitrary type and size into a single message construct.
- WS-Attachments: Defines an abstract model for SOAP attachments and based on this model defines a mechanism for encapsulating a SOAP message and zero or more attachments in a DIME message.

WS-Security and WS-Routing are published specifications whereas DIME and WS-Attachments are IETF Internet drafts at the moment the work is in progress.

In addition to the web services, you can also use .NET Remoting to implement SOA-based software. It allows the clients to connect to a server, without knowing about details of implementation of the service. You can specify the location of the server using a configuration file on the client. You can also use side-by-side versioning feature of .NET, which allows you to publish multiple versions of the same component at the same time. Specifying binary format for messages results in a better performance as compared to web services and you can still use HTTP, so that firewalls do not become a concern.

The major drawback for using Remoting is that it is platform dependent, meaning that both client and the server need to have .NET framework installed.

Visual Studio .NET is a great time saver for developers. As I mentioned before, having an environment which provides IntelliSense when consuming a web service can be a great step forward. Visual Studio .NET 2003 allows developers to locate a web service by searching the local machine, UDDI servers on a local network or global UDDI directory. Once you add the reference to the web service to your project, .NET builds a proxy for the web service and you can use the web service as if it was a local class on your machine.

Another benefit of .NET is its built-in serialisation mechanism for classes like DataSets that are usually used to transport data. You can easily return a DataSet from a web service, without being concerned how to serialise and deserialise the data. It also allows you to customise the serialisation process easily if you want to improve performance.

Windows Server 2003 which will be released by in less than a month has built-in support for Web Services. By native support Microsoft does not mean just an upgrade for IIS to run Web Services. Windows Server 2003 benefits from a device driver names HTTP.SYS which serves HTTP requests. Since device drivers run in kernel mode rather than user mode, it will greatly improve performance. Windows Server 2003 has a built-in UDDI server too, which means that you can register and publish your web services on your own UDDI server.

References:

<http://www.microsoft.com/windowsserver2003/evaluation/overview/technologies/iis.msp>

<http://msdn.microsoft.com/webservices/building/wse/default.aspx?pull=/library/en-us/dnwssecur/html/wssecauthwse.asp>

<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/progwse.asp>

http://java.sun.com/blueprints/guidelines/designing_webservices/plattechs.pdf