# What is SOA anyway?

*Getting from hype to reality*

*By Arnon Rotem-Gal-Oz*

Service Oriented Architecture or SOA for short has been with us for quite a while. Yefim V. Natiz, a Gartner's analyst, first talked about SOA back in 1996. However it seems that only in the recent year or so SOA has matured enough for real systems based on the SOA concepts to start to appear – or has it? There is so much hype and misconceptions surrounding SOA that we first have to clear them all up before we can explain what SOA is – let alone identify who really uses it.

This aim of this paper is to try to clear some of the fog surrounding SOA and provide a clear definition of the term

## 1. Service Oriented Ambiguity

Martin Fowler best described the confusion surrounding SOA in his Bliki in a post called Service Oriented Ambiguity. Martin says that the question "what is SOA ?" is impossible to answer because SOA means different things to different people. While it is true that the term has been overly used, at least few of these "definitions" are plainly wrong. Let's take for example the definition that says SOA is exposing methods through Web-services.

### a. SOA == Web Services?

A common misconception about SOA is that using web-services technology makes whatever you are using SOA. The core reason for that is the poor naming choice for methods that are exposed through http which were named web-**services**. For example one of the popular books on SOA "Service oriented Technologies: a Field guide to integrating XML and Web-Services" by Thomas Earl gives the impression that SOA equals WS* (even though that buried in the book, there's an explanation hat WS* is not the only possible technology for SOA). There are many other sources that give the same impression.

Nevertheless, Just a bunch of web-services (JBOWS as Joe McKinderik named them) does not an SOA make – in fact that's isn't really different from Remote Procedure Calls (RPC) using any other technology be that CORBA, DCOM or anything else. The way I see it saying that SOA is using web-services is just plainly wrong and not away to define SOA.

### b. SOA == EAI?

Another point Martin mentions is that a widely used definition for SOA is "EAI without all the expensive EAI vendors locking you in". EAI emerged as an attempt to solve the Enterprise integration spaghetti. While it managed to solve the problem of connecting applications and

transferring data between them through a semi-standard way. I agree with Sandeep Arora who said that basically EAI failed to deliver its promise as it is

- Data centric and not process centric.
- Can't keep up with business process change.
- Does not address the business process.
- And that EAI solutions are technically very complex, need specialized skills and are very expensive to maintain.

Whether Arora is right or even if EAI is the best thing since sliced bread - If we do EAI using web-service technology to get "EAI without the expensive EAI vendors locking you in" (which I also seen called SOI – or Service Oriented Integration) – we are again not doing anything new, we are just using a new technology to achieve an existing way of thinking or architectural approach.

### c. Ambiguity – is everything lost?

Martin claims that the SOA acronym is beyond saving and that the (sometimes) good ideas that are all called SOA need to have their own name and independent life. However the way I see it the examples Martin provides are "just" misuses of the term. I think that the fact that someone uses a term wrongly does not invalidate the correct term. Nevertheless, SOA troubles do not end here – since the hype also looms behind the corner to dilute the SOA term as well.

## 2. SOA Hype and Myths

I am sure many of you have seen that happen before – You are working on v0.9 version of project, making nice progress. Then, almost out of nowhere, steps out this marketing wiz-kid or wonder sales person who meets a client and tells him all about the wonderful features of the product (you were planning for v3.5), signs a deal and leaves you pick up the pieces o somehow try make all that work. Hype is like the power of thousand such wiz marketers working against all of us – making it almost impossible to live up to these "common truths" which are actually hollow promises.

Let's just take a quick look at few of the more common hype-originated myths surrounding SOA

### a. SOA will make reuse easy

The claims that SOA will increase reuse and/or will make it easy is very common- so much that it has been adopted by serious technical people – for example you can see an article entitles "SOA: Separating myth from reality" (!) by Mark Potts who is a CTO in one of HP's divisions which states: *"One of the major benefits of the set of architectural principles that define an SOA drives organizations towards a greater level of reuse and consistency. The amount of achievable reuse increases over time once newer services are designed using existing services in an organization…* "

Well, as object orientation thought us reuse is harder the larger the components we want to reuse. Since larger components have a more context they carry with them which make it hard to reuse them elsewhere. SOA is not different. Services should encompass a meaningful business capability, if you build it right you can **use** it to solve your business needs and you can integrate it to fulfill your business processes. The chances are however, that you will **not be able to reuse** elsewhere. SOA is more about business agility and the ability to change rather than reusing services out of their original context.

There is some substance to the reuse claim, if we take a broader definition of the term reuse. As an organization moves to SOA, it can make the transition an evolutionary process - and still use existing assets in the transition phase. The evolutionary process approach, contradict another SOA hype generated myth which states that SOA requires an all or nothing approach.

### b. SOA requires a big-bang approach

Consultants and technology vendors would have you believe that you either have to fully embrace SOA or it just wouldn't work. I don't think so. I agree with Sohel Aziz that it is actually the other way around, if you want to succeed with SOA you need a pragmatic step-by-step approach.

When you think about it, a Full Enterprise SOA initiative would mean replacing **all** the enterprise's systems. Do you really think the business will just freeze or halt while the SOA initiative is underway? Not quite. Rather, a comprehensive SOA initiative is like building a new interchange on a highway. You need to provide ways for the traffic to continue to flow (business to continue to operate) while work is underway. More so, you will make as much progress as you can without disturbing traffic.

Also SOA characteristics (which I'll expand more about later) can even allow you to take an incremental way for removing legacy systems by exposing existing functionality using new SOA interfaces before you actually replace the underlying system. Using SOA for

### c. SOA will make integration easy

Another SOA myth is that it will make integration easy. SOA puts a lot of emphasis on the interface , which makes it "easy" to just make services talk to each other. On top of that SOA is (usually) based on standard protocols like XML, WS*, http and the like, so it is easy to just tie services together and integrate them.

However, Fred Brooks told us ages ago.: "*The essence of a software entity is a construct of interlocking concepts: data sets, relationships among data items, algorithms, and invocations of functions. This essence is abstract in that such a conceptual construct is the same under many different representations. It is nonetheless highly precise and richly detailed. I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation. We still make syntax errors, to be sure; but they are fuzz compared with the conceptual errors in most systems".* SOA does not change these basic rules. To properly integrate services you need to model their contracts in a way that will make them usable for multiple business processes – this

is not a small feat and that is where you'd spend most of the time – this is still hard and SOA does not solve it. Once you solve that, SOA does help you lift some of the burden of doing the actual integration - but that's just making integration easier and not making it easy.

### d. From myths to reality

Yes, it seems SOA is indeed in dire straits. In fact, I've almost managed to convince myself that Fowler is right and SOA **is** beyond saving. However as mentioned at the beginning over the past year or two we start to see systems –The reasons for that is that in fact we start to see some consensus on what SOA is and isn't. It seems that today there are two prevalent ways to look at SOA– from a business point of view and from a technical one.

## 3. And then they were two

Looking beyond the hype and misconceptions it seems we can narrow the field into two points of view for the SOA concept and both of them seem to be valid– one from the business perspective and another from the technical one.

### a. Service Oriented architecture

At the enterprisearchitecture level, it is always about the business. This is not a bad thing, on the contrary, the enterprise architecture perspective should be focused on the business needs in order to make sure IT serves the business and not vice versa.

The emphasis from the business perspective is on "service orientation". Consider for example the SOA definition from Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology - Eric A. Marks, Michael Bell : *"SOA is a conceptual business architecture where business functionality, or application logic, is made available to SOA users, or consumers, as shared, reusable services on an IT network. "Services" in an SOA are modules of business or application functionality with exposed interfaces, and are invoked by messages."*

Looking at other "business oriented" definitions of SOA we can see they follow the same reasoning. In a nutshell, they can be summarized as follows: from the business point of view SOA is about analyzing the business to identify business areas and business processes. Followed by defining services to represent these "areas" .Services expose their capabilites through message interfaces. The services can then be choreographed or orchestrated to realize the business processes. The goal of SOA is to increase the alignment between business and IT and achieve business agility – the ability to respond to changes quickly and efficiency.

And then there's the other face of SOA – this time from the technical point of view

### b. service oriented Architecture

While the on the business side of the fence the emphasis is on" Service Orientation" or SO - on the technical front the emphasis is on the A of SOA – Architecture. True, there isn't a single unified definition for SOA; however, just like the many definitions of software architecture,

there are several characteristics that are more common and frequent than others. Looking at definitions of SOA such as the ones from Wikipedia, O'reily's, JavaWorld , Windley, Microsoft etc. you can see that SOA is commonly thought of as an architecture or an architecture style that builds on loosely coupled, interoperable and composable components or software agents called services. Services have well-defined interfaces based standard protocols (usually web-services but most definitions mention that it is not the only possible implementation) as well as QoS attributes (or policies) on how these interfaces can be used by Service Consumers. SOA definitions mentions the basic communication pattern for SOA is request/reply but many definitions also talk about asynchronous communications as well.

If we look at the business and technical approaches for SOA we can see that there's still hope to achieve convergence as there are some common grounds - the next section will try to do just that.

## 4. SOA defined

In order to be able to converge between the technical and business viewpoints we first need to differentiate between an architectural style and its application – once we define what SOA is we can apply it at an organization level to get an SOA initiative where services will encapsulate business function. However we can also apply SOA on a single project and get services whose content revolves around technical issues like security or management.

We also need to differentiate between design goals such as loose coupling or business alignment and architectural building blocks and constraints like coarse grained services or policy based interactions

Lastly, if we look at definitions of other architectural styles like Client/Server, Layered or REST we can see that we can see that architectural styles are defined in terms of components, their attributes, their relations and the rules or constraints that govern them.

Based on that we can define Service Oriented Architecture as an architectural style for building systems based on interacting coarse grained autonomous components called services. Each service expose processes and behavior through contracts, which are composed of messages at discoverable addresses called endpoints. Services' behavior is governed by policies which are set externally to the service itself. Figure 1 below shows the SOA componets and their relations:

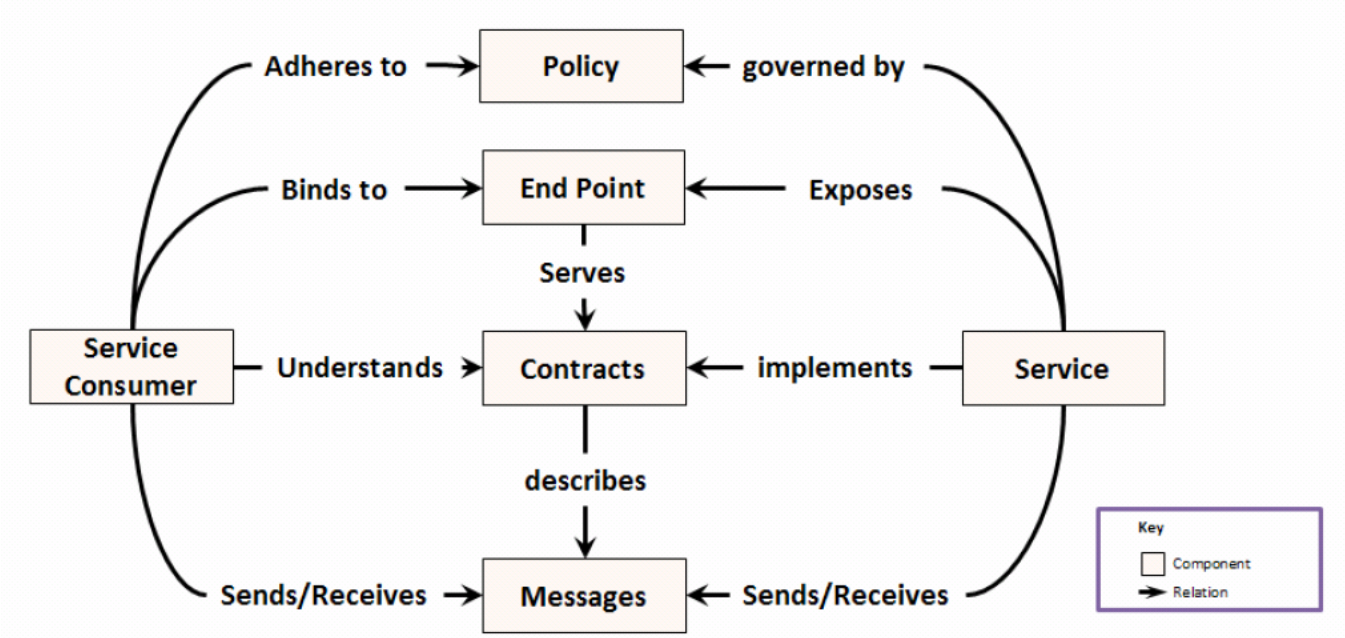


**Figure 1: SOA components** (Arnon, 2006)

Let's take a look at each of these components (excerpt from my upcoming SOA Patterns book)

*Service*

The central pillar of SOA is the service. Merriam Webster defines service as "a facility supplying some public demand". A Service should provide a high cohesion and distinct function. Services should be coarse grained pieces of logic. A Service should implement at least all the functionality promised by the contracts it exposes. One of the characteristics of services is service autonomy. Autonomy means the services should be self-sufficient, at least to some extent, and manifest self healing properties.

*Contract*

The collection of all the messages supported by the Service is collectively known as the service's contract. The contract can be unilateral, meaning a closed set of messages the service chooses to provide. A contract might also be multilateral or bilateral, that is, between a predefined group of parties. The contract can be considered the interface of the Service akin to interfaces of object in object oriented languages.

*End Point*

The Endpoint is an address, a URI, a specific place where the service can be found and consumed. A specific contract can be exposed at a specific endpoint.

*Message*

The unit of communication in SOA is the message. Messages can come in different forms and shapes, for instance, http GET messages (part of the REST style) ,SOAP messages, JMS messages and even SMTP messages are all valid message forms.

The differentiator between a message and other forms of communication such as plain RPC, is that messages have both a header and a body. The header is usually more generic and can be understood by infrastructure and framework components without understanding, and consequently coupling to, every message type.

The existence of the header allows for infrastructure components to route reply messages (e.g. correlated messages pattern) or handle security better (see Firewall pattern).

*Policy*

One important differentiator between Object Orientation or Component Orientation and SOA is the existence of policy. If an interface or contract in SOA lingo, separates specification from implementation. Policy separates dynamic specification from static/semantic specification. Policy represents the conditions for the semantic specification availability for service consumers. The unique aspects of policy are that it can be updated in run-time and that it is externalized from the business logic. The Policy specify dynamic properties like security (encryption, authentication, Id etc.) , auditing, SLA etc.

*Service Consumer*

A service doesn't mean much if there isn't someone/something in the world that uses it. So to complete the SOA picture we need Service Consumers. A service consumer is any software that interacts with a service by exchanging messages with the service. Consumers can be either client applications or other "neighboring" services their only requirement is that they bind to an SOA contract.

## 5. Summary

Looking at this SOA definition we can see SOA has a lot of emphasis on interface. Starting from the messages which are the parts of the interface, the contract which is the collection of the messages, the endpoint where the contract is delivered and the policy which governs the behavior of the endpoint. Thus SOA has a total of four different components that deal with the interface vs., for example, OO which only has one. The focus on interfaces is what gives SOA the ability to create loose coupling, composable components, reuse and achieve the various design goals. Another nice attribute of this definition is that we can use as a base for both the technical and the business perspectives of SOA as the common elements of both perspective are used in this definition.

Thus, even though there are a lot of misconceptions and hype surrounds SOA  There is value in the term . multiple definitions does not have to translate to ambiguity if they are just different workings for the same concepts. We do however have to be careful not to be fooled by the hype and misconceptions.  I hope that the definition provided here helps achieve this goal.

## Bibliography


Aziz, S. (2006). *Service Oriented Architecture: Look beyond the myths to succeed.* Retrieved from InfoSys: http://www.infosys.com/services/systemintegration/white-papers/SOA-look-beyond-myths-to-succeed.pdf

Brooks F. (1987, April) “No Silver Bullet: Essence and Accidents of Software Engineering," *Computer*, Vol. 20, No. 4 pp. 10-19 Retrieved from http://www-inst.eecs.berkeley.edu/~maratb/readings/NoSilverBullet.html

Fowler, M. (2005) Service Oriented Ambiguity. Retrieved from http://www.martinfowler.com/bliki/ServiceOrientedAmbiguity.html

He H. (2003) What is Service Oriented Architecture, O’reilly XML.Com Retrieved from http://webservices.xml.com/pub/a/ws/2003/09/30/soa.html

Kodali R. R., What is Service Oriented Architecture? An introduction to SOA, Javaworld.Com Retrieved from http://www.javaworld.com/javaworld/jw-06-2005/jw-0613-soa.html

Marks, E. A; Bell M., (2006 June) Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology, Wiley.

Microsoft (2006 Dec.) Learn about Software Oriented Architecture (SOA), Retrieved from http://www.microsoft.com/biztalk/solutions/soa/overview.mspx

Natis, Y. V. (2003, April 16). *Service-Oriented Architecture Scenario.* Retrieved from Gartner: http://www.gartner.com/resources/114300/114358/114358.pdf

Potts, M. ( 2006, Jan 28). SOA: Separating Myth from Reality,  SOA World Magazine, Retrieved from  http://webservices.sys-con.com/read/175375.htm

Rotem-Gal-Oz A. (2006). *SOA Patterns Chapter 1 (Draft).* Retrieved from Manning Publications : http://www.manning.com/rotem/free.html

Sandeep, A. (2005, Dec. 9). EAI, BPM and SOA. SOA Institute Ord. Retrieved from http://www.soainstitute.org/articles/article/article/eai-bpm-and-soa/news-browse/2.html

Thomas, E. (2004). *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall.

Wikipedia (2007) Service Oriented Architecture. Retrieved from http://en.wikipedia.org/wiki/Service-oriented_architecture

Windley, J. P. (2004). *Service Oriented Architectures.* Retrieved from The Windley Group: http://www.windley.com/docs/2004/SOA.pdf