

Service Oriented Architecture: Final Paper

Marc Strom
(stro0447@umn.edu)

Kyle Estes
(kestes@gmail.com)

Kirti Kesavarapu
(kesa007@umn.edu)

Brian Shipe
(ship0035@umn.edu)

Introduction

Service oriented architecture is a new technology which has just come about in the past few years. It is a new way to design software promoting the ideas of reusability, long life span, and easy updating. This paper will focus on the history of where service oriented architecture has come from, how exactly service oriented architecture works, practical examples, and a comparison of service oriented architecture to other architectures.

History of SOA

Although it is difficult to define the history of Service Oriented Architecture (SOA) in terms of when it was created and who founded it, SOA's history can be defined in terms of the impact it has had on industry practices and thinking. SOA itself is based on the Server/Client paradigm, which is a method for computing that distributes the computing between a client computer and a server computer. In the Server/Client paradigm, the server computer is usually responsible for performing the majority of computation and data storage, while the client is a lighter weight application that provides a graphical user interface (GUI), among other things. In SOA, there exist the concepts of a service provider and a service consumer. The service provider runs on the server and client applications are allowed to consume services from the server. In this manner, SOA represents an evolution, not a revolution, in architecture ("Service-oriented architecture").

Theory and Concepts of SOA

In describing the conceptual view of Service-Oriented Architecture, one would expect a model that is representative of any and all implementations of it. This would provide the minimal concepts needed at a high enough abstraction level to facilitate a specific SOA solution, independent of the problem domain and of the technologies, standards, and implementations. This allows for growth, reuse, and interoperability as the model does not include these aspects that are ever changing. This description is often called a reference model and the Organization for the Advancement of Structured Information Standards has proposed one such model for SOAs.

For example, think of a reference model for a restaurant. One imagines that a restaurant needs the concepts of an eating area, a cooking area, and a hygiene area. A specific architecture of this would entail a dining room, a kitchen, and restrooms. Supposing this restaurant also has a game room for children to enjoy, this would also qualify as a restaurant as it includes the realization of the minimal concepts needed, but has a different architecture. This shows the value that a reference model has as it can describe something in a general form to allow differing kinds of specific architectures.

The SOA reference model is concerned with seven main concepts: service, visibility, interaction, real-world effect, service description, policies and contracts, and execution context. These are grouped into three categories according to the functionality that they bring to SOA. The first one is the category of service which has the stand alone concept of service. The second category is the dynamics of services which contains the concepts that are critical to the operation and use of the service. This contains concepts of visibility, interaction, and real-world effect. The last grouping is that of the concepts that support those in the dynamics of services. This contains the ideas of service description, policies and contracts, and execution context.

Before jumping into these concepts it is beneficial to understand the basics of how services are used. Services are provided by service providers and used by service consumers. The consumers discover a provider that can fulfill its needs through a service description and interact with it through the service's interfaces to result in a real-world effect. While there are other factors involved, this is basic enough to understand the underlying concepts that will be detailed. It should be noted that each participant's actions are private and not known to any party outside, unless these actions affect some shared resource.

The heart of service-oriented architecture is the notion of the service itself. So what is a service? Simply, a service is just a means to bring needs and capabilities together. A service allows its capabilities to be accessed through its service interfaces, but can only be used within the constraints that its policies and contracts, along with the consumer's, have specified. These interfaces only detail how to access the service's capabilities as the consumer does not need to know how it is implemented, with the exception that only enough implementation information is needed to use the service. As a result of using a service, a realization of some real-world effect occurs. This could be a result of returned information based on a request, a change to the shared state of some defined entities, or a combination of the two.

The process of going from a consumer having needs to a provider fulfilling said needs through its services is facilitated by the concepts within the dynamics of services. The first step would be for the consumer to identify a service that is capable of satisfying its needs. This leads to the concept of a service having visibility.

Visibility is the basic idea that the service provider and service consumer need to be able to "see" each other in order to interact. Consider a basic C program "Hello World" that prints a string to stdout. If you view the library `stdio` as a service and your program as the consumer, there needs to be a means for the two to know about each other. This is carried out through the use of the statement `#include <stdio.h>`. Within the concept of visibility there are three subcomponents that need to be addressed as they act as preconditions for visibility. These are the notions of awareness, willingness, and reachability.

Awareness is the idea that one party needs to know about the existence of the other parties for an interaction to occur. This requires that a service description and set of policies be available for the consumer to know about the existence of and capabilities of the service. For the service to be known to a consumer, it may push its description and policies on a consumer, such as the case for television ads. Conversely, consumer may pull service descriptions searching for suitable services. This is analogous to a company having contractors making an offer for a certain project. Awareness is also the means to which a consumer can decide if the services capabilities match their needs.

Willingness is needed as a consumer and provider may be aware of each other, but may not be willing to interact. Suppose a consumer discovers the service and initiates an interaction, but the service refuses to cooperate and an interaction never occurs. This may be the desired behavior in the context of defending against a denial-of-service attack. The willingness to interact, however, is not to be confused with the willingness to perform some requested action. Think of basic human interaction, one may be willing to interact with their friend, but if their friend asks them to wash their car and clean their house they may reject both requests, but are still willing to interact with them.

The last aspect of visibility is the somewhat obvious one of reachability, in that the consumer and service provider need an actual means to communicate. Without such a means an interaction is impossible as the two have no visibility of each other.

Once visibility has been satisfied and the preconditions met, the next step is to actually interact with the service. This is the act of a consumer performing actions against the service.

Often times this is done through the use of message exchanges, but may also be done through other means, such as the alteration of state of a shared resource. For a consumer to understand what is entailed in interacting with the service, it must reference the service description. Primarily, it consults the service description to find out what information can be exchanged and how it may interact with the service. These models called the information model and behavior model, respectively.

The information model specifies what information may be exchanges as well as the syntax and semantics of the said information. The syntax describes that structure of the information, such as the data types like strings or int, where the semantics is the meaning of the data. For instance, when describing an address, the syntax of the city could be a string, but the street and state may also be represented as strings, so some distinction must be made to understand it. Thus, the idea of a street, city, and state needs to be enforced. The information model is critical in services that are outside the boundary of ownership of the consumer where the data being transmitted is completely unknown.

The behavior model represents the actions that can be performed against the service, responses to those actions, as well the process of interacting with it. As an example, a database service may have the actions that can be performed be presenting security authorization, running queries, and updating records. These actions need to be done in some order for an interaction to be successful. Much like the information model, the behavioral model can be decomposed, where the behavioral model is broken into the action model and process model.

The action model represents the possible actions that may be invoked against the service as well as the expected behavior of those actions. Since internal actions are private, this behavior may not be explicitly included in the action model, but its implied effects will be. The process model regards the temporal aspects of how to interact with the service. This, however, is not explicitly defined within the reference model, as it may include aspects that are not strictly part of SOA, but is nonetheless and important aspect that must be defined at least to address the interactions with the service itself.

The final step is the real-world effect, or the reason that the consumer is using this service. Consider a flight reservation service. The real-world effect is that of a reserved seat on that specific flight. This effect must be done through some state change in a shared resource as all internal actions that are performed by any participant are private. Through the accumulated changes to this state, a real-world effect is realized. These accumulated changes are closely tied with the execution context of the interaction (to be described), which essentially provides the environment of the interaction.

With the dynamics of the service having been described, the concepts that support them should be addressed for a deeper understanding of all that is SOA. If one recalls, the service description is one such concept that supports the functionality of the service.

The service description lays out what the services capabilities are and is the means to which a consumer can decide if the service is suitable to fulfill its needs. The service description should contain information about the service's existence and reachability, the set of functions that it provides, the policies and constraints that it runs under and that it will comply, to some extent, with the policies and constraints that consumer imposes, as well as what messages may be exchanged (information model) and how to interact with the service (behavior model). These may not need to be explicitly included through the use of links or references to other descriptions that include them, such as links to broad policies, in order to take advantage of reusability.

The second concept that supports the dynamics of a service is that of the policies and contracts that constrain the service. Policies represent some constraint on the use,

deployment, or description of some owned entity of a participant. One example is that a consumer requires that all data be encrypted. A contract is just an agreement between two or more parties. These concepts are analogous to the policies and contracts that relate to those that are seen in everyday life and can be mapped as such, so no further details will be provided.

The remaining concept is that of an execution context as briefly described. The execution context, more specifically, is the infrastructure elements, policies and constraints, and process of a specific instantiation of a service. This effectively acts as the path between the consumer and provider. As previously stated, it can be thought of as the environment in which the interaction takes place, in that it considers the interaction as a whole and changes as throughout the life of a specific interaction. As such, it is also the context in which the data is interpreted, which also allows for a consumer to distinguish which instantiation of the same service it can use by inspecting the execution context. With this ability, it is then possible to use multiple services as long the execution context can be transmitted within the message that is being exchanged. This idea of statelessness is encouraged, so as to take advantage of the lack of dependency on a single service instantiation, but one must also recognize that there may be cases where state is necessary.

These underlying concepts of SOA are critical in understanding what is required at a minimal level for such a solution. The intent of the reference model is just that, to capture the essence of SOA and to provide a common understanding so as to be a base point for the definition of more specific reference architectures or the implementation and design of specific architectures. With this knowledge one can deploy SOA solutions regardless of the current technologies and standards.

Service Oriented Architecture Examples

At its most primitive level, the Internet is a collection of documents that can be shared and accessed from remote locations. However, recent advances in technology are requiring more functionality from the Internet. Technologies like AJAX, Flash, and Server Pages are signs that the Internet needs to evolve into more than a collection of accessible documents. The Internet needs to have the capacity to support web applications.

Rising to meet that need are web services. Web services are a combination of the following technologies: HTML, XML, UDDI, SOAP, and WSDL. XML is the core of web services while HTML serves as the presentation layer. SOAP and WSDL are both technologies built on XML that allow web services to function.

Web Services Description Language (WSDL) uses XML to describe the interface to the web service. It indicates where the service is located and the operations that the service provides for use. WSDL was first introduced by Ariba, IBM, and Microsoft to the World Wide Web Consortium (W3C) for describing XML activity for discussion in March 2001 but was formally released in July 2002. Since then, it has been a vital technology in web services.

Closely tied to Web Services, is the Universal Description, Discovery and Integration (UDDI) directory. It serves as a place where business can register and search for web services. UDDI really helps integrate the business processes of different companies to reach their customer base. For example, air line companies could use UDDI to allow customers to check flight rates and make reservations from one central location. Over 220 companies are members of the UDDI community and with some more time, it may grow to be an essential technology.

Unlike applications aimed at the average user's desktop computer, Web Services need

to support all the different operating systems that exist and will exist in the future. To accomplish this, web services use the Simple Object Access Protocol (SOAP). SOAP is primarily a communication protocol. Based on XML, it is a language for describing and sending messages. It is platform and language independent allowing for communication between processes without the limitations that plague desktop applications.

Combining all these technologies, we can get a web application like behavior. For example, if we write an application for converting between units, it can be used as a web service. Using WSDL, we can describe what functions/operations the service supports as well as where the service resides. Also, using SOAP, we can send and receive messages to the service from different applications. If we wish to make this service available to everyone, it can also be registered with UDDI. Together, these technologies take service oriented architecture to the web and create the foundation for what might redefine how users and programs interact with the Internet.

Services are by no means new. They have been essential parts of popular operating systems such as Windows and Unix/Linux. Although they were crucial parts, the operating systems were definitely not designed around these services. With the realization of SOA, the windows operating system is making an effort to expand it's foundation to take advantage of the benefits of SOA. As a component of WinFX, WCF (also known as Indigo) is the new technology that leverages SOA to handle communication between different processes. As it stands on the Windows platform, there are several different ways to communicate between processes such as MSMQ, WSE, and .Net Remoting. WCF serves to integrate the roles of all those solutions into one encompassing technology. With the help of SOA, the solution produced is elegant and effective.

The general idea behind WCF is to integrate services into windows applications using the framework provided. Similar to web services, these embedded services communicate with each other with the use of SOAP. The beauty of WCF is that communication between applications now transcends the limitations of the operating system. Because WCF uses SOAP as its underlying communication framework, applications written on other operating systems can still communicate with the use of messages. Applications don't need to fumble with shared memory or operating system specific implementation of message queues. Like web services, applications can communicate on the same machine as well as between different machines.

Developers of WCF were careful to keep a few important things in mind when designing the technology. First of all, services only interact by sharing their interfaces and do not pass entire classes between each other. Along similar lines, services in WCF are designed to be autonomous. While the client and a service may agree on an interface, they are independent of each other. This means that they can be implemented using different languages, runtime environments and may operate on different operating systems. Last but not least, the program boundaries are established very clearly. Unlike previous technology like DCOM which tried to hide the fact that remote objects were involved, WCF does not hide the fact that tasks are being delegated to a service. By following these tenants and more, WCF supports service oriented development.

Creating a WCF service is very simple. To begin with, we need to create a service class using a CLR based language like C# or VB.NET. Because of how WCF is designed, the service needs to have a host application to run inside. Finally, since a service is useless without providing access, a service needs to provide one or more endpoints to allow clients to utilize the service. These three components are part of WCF development only and service developed under a different framework may be constructed using different techniques. However, as long as the services use SOAP, they can communicate with WCF across

machines and operating systems.

Service Data Objects

With the advent of computing, data was stored and access from simple files stored on the harddrive. With time, databases have become increasingly popular for storing data on a larger scale. As with any technology, several versions of database management systems (DBMS) have been released and each of them supports a different interface for access. With further development, XML became a data description language that contained the data as well as the schema contained within the document. With only a few decades behind computing, we already possess so many types of data sources and an absolute possibility of more to come. However, the industry is struggling to provide a common interface to accessing data between all the different sources. Technologies such as Java and .NET have created their own versions of uniform data access, but they are developed specifically for each of those technologies.

IBM and BEA have introduced a proposal for a new technology called Service Data Objects (SDO) that transcends the limitation of a specific development language. SDO contains a collection of services that handle data retrieval and storage from different sources. For example, one SDO service is written to handle data exchange between an application and an XML data source while another SDO service handles exchange between an application and an Oracle database. All the services return the data as a datagram which can be used and modified by the application. Using the datagram format, the application can also apply changes back to the original data source. The service reinterprets the data from the source as a datagram when retrieving and converts a datagram into the source's native format when writing data to the source. Such an architecture can prove useful for dealing with change as new data formats are introduced into the industry and as existing ones are modified. SDO is still in the proposal stage and a first draft is being developed by several companies. If successful, SDO could eliminate the headache of dealing with multiple data sources with the effective use of SOA for change management.

SOA vs. Other Architectures

Service-Oriented Architecture (SOA) is used because of its very loose coupling between separate objects and data. SOA uses object oriented programming techniques to achieve this loose coupling. Loose coupling means that each part of the program is self-reliant and that minimal assumptions are made regarding the interfaces. A self-reliant process is one that is given data, runs processes on the data, and/or stores the results. It does this without the help of another process. The developer using this process does not care how the process manipulates or stores the data as long as the correct data is given when requested. The advantage to developing software in this style is that if any underlying algorithms are changed as an update to the object, the interface with the object will not change. The developer will continue to use the process in the same manner regardless of what changes occur in the implementation of the process. This concept allows for a very long lifespan of the process. The lifespan of the systems using this process will also increase. (Morgenthal)

Another important aspect of an SOA system is that new processes are created with the idea they will be reused in another system later. Any processes not specifically created for a given system have most likely already been designed and fully tested for use from another system. The advantage to this type of design is that it is very easy to implement this type of system. The processes which are being reused have already been fully designed and tested. These processes just need to be used in the current system. The only parts which need to be

designed in this system are the new processes which have not been implemented yet. Overall using the concepts of SOA will decrease the time needed to build the system and increase the time the system can be used without a major overhaul. (Morgenthal)

The downside to the SOA design style is that with the very loose coupling the abstraction becomes more complicated. If two processes, A and B, are loosely coupled, accessing data in Process B from Process A can become an expensive task. Older technologies did not run into problems such as these. Most systems were designed and were very tightly coupled. All of the different parts of the system rely heavily on each other to store, process, and/or retrieve the data. The advantage to systems designed using this methodology is that they are very efficient in running, due to the fact that the data is very accessible at any part in the system. Each part of the system is streamlined specifically for this system. Another advantage to this type of system is that the information is very secure since each system is designed specifically for the type of data being stored. The disadvantages to this type of system is that individual parts of the system are not reusable and if any changes are made, a large part of the system will need to be updated. To implement any new technologies which become available, the system will need to be given a complete overhaul. This fall back will drastically reduce the lifespan of a given system. (Morgenthal)

Object Request Brokers (ORB) is an example of the older technology. An ORB is middle ware software which allows a developer to make program calls from one computer to another via a network. Some ORBs use an Interface Description Language (IDL) to describe the data which is being sent across the network. Given a message, the IDL would describe the data in the message so the computer receiving the message would understand what was being received. Encapsulating this was the concept of a Remote Procedure Call (RPC). This would work using two messages. The first message would request a program to be run using the given parameters and the second message would contain the results of the program. ("Object request broker")

Common Object Request Broker Architecture (CORBA) is an example of an ORB. CORBA would send messages between systems as part of the RPC protocol. It was best suited for a high volume number of transactions needing security such as bank or retail transactions. The CORBA systems are very fast because they are very tightly coupled and designed specifically for the purpose it is serving. A disadvantage to CORBA is that it only works with other systems already using CORBA. (West)

These types of systems, using an ORB type of design, are usually fairly complex, just from the nature of the system. However, the same systems created using SOA methods are much more complex. This is due to the fact that the individual processes and parts of the system are too loosely coupled. Extra work is needed to tie these loosely coupled parts of the system together and make the SOA system more complicated. (West)

An advantage SOA has over an ORB system is that SOA is able to communicate between two programs each written in any language. The ORB systems do have IDLs to help communicate, however an IDL does have to be written to translate one language data into another language. The good part of this is that most languages already have an IDL written for them. The downside is that any new languages will need to have an IDL created specifically for them. SOA already has a structured way to send the data between computers. An example of this the "HTTP GET" statement. A message can be sent from any computer to any other computer using any type of language. The receiving computer is able to understand the data based upon the header of the message. (West)

When designing a system, the biggest question to be answered is re-usability vs. performance. If a system is going to need to run fast for a very specific application, then

CORBA or an ORB designed system is an appropriate choice. The system is not very easily updated or reused, however it will be much faster than an SOA designed system. If the system is going to be constantly updated and speed is not necessary, then an SOA designed system is very well suited. It will be able to be reused in the future and allow for similar applications to be easily designed off of the original. (West)

The repository architectural style is an example of a cross between SOA and ORB systems. In a repository design, all of the data is centrally stored in relation to the processes involved with the data. Each of the processes using the data are loosely coupled, however the processes with the data are tightly coupled together. The advantage to this is that the data can be stored in a very controlled location and be very secure. Also, new processes can be added into the system very easily since individual processes are loosely coupled. The disadvantage to this type of design is that the repository can become a bottleneck when all of the processes attempt to use the database at once. Because of the very tight coupling between the repository and the processes, if any changes do occur in the repository then changes will also need to take place in the related processes. The repository architectural style is best used for database management systems, such as payroll or bank systems. (Bruegge 238)

The repository style is similar to SOA systems since the processes are very loosely coupled. This allows for new processes to be added very easily. This style is also very similar to the ORB systems because of the tight coupling of the data to its process. Each individual process is designed with the specific data in mind. This increases the efficiency of each process being run on the data.

SOA is a great new technology however a few things do need to be improved upon. Since many SOA systems are used over the internet between different languages, there is no good way to secure the data in each message. One solution is to use which has already been implemented is through the use of Web Services Security (WS-Security). This type of protocol incorporates security in the header of the message. An example can be found in the HTTPS protocol. Another problem with SOA is that there is no good way to describe unstructured data in the header of a message. Unstructured data may be binary data recognizable only to the receiving computer, however there is no way to tell the receiving computer what exactly that data is. ("Service-oriented architecture")

Compared to its ancestors such as ORB designed services, SOA is almost a completely opposite approach to design. ORB services are designed around very tightly coupled systems, run very efficiently, and need a certain level of security. SOA services are designed with the idea of being easy to create, update, and reuse for similar systems in the future. The repository style sits somewhere in between both the SOA and ORB design styles, containing both positives of each type of design. The best design choice for a given system is for the pattern which will best fit the final system, not the newest concept. In today's world of needing a well built system which will last for a long time and be reusable, an SOA system is an excellent design strategy.

Future of SOA

One of the reasons that SOA has come into prevalence is because it promotes high reusability and loose coupling of services. Services can be thought of as discrete modules that perform well defined operations on their inputs to produce the output. When developing the services to be used in an SOA, engineers focus on creating highly reusable, loosely coupled services that are very good at accomplishing a small number of tasks. However,

services in this state are hardly ready to be used directly by consumers. For instance, imagine a credit card processing service that is used by an online retailer. The credit card processor is a specialized service that takes in credit card information and performs the necessary operations on that information (e.g., verification, charging the card, etc.). As a person shopping in the online store, you would never expect to have to deal with the credit card processor directly. Instead, you are presented with an interface that, behind the scenes, orchestrates many of these specialized services into something that is useful (e.g., purchasing an item). In SOA jargon, this concept is known as a business process. In researching what SOA is all about, the reader will see terms like “business processes” and “business rules”. Business processes are simply a figure of speech that indicate the organization of loosely coupled, highly reusable, small services into a set that together can perform a meaningful task. Thus, a business process is simply a particular organization of services that meet an application-specific need and has little to do with how business or companies operate.

Also, the rise of SOA to buzzword status in the software engineering community is closely linked to the web, and in particular web services. In fact, while SOA does not require the existence of web services as we know them today, it is hard to imagine SOA developing as it did without the web and web services. The reason is because web services are a perfect example of the realization of SOA. The past few years have witnessed some interesting developments in web services. Recently a small private company in California made the winning bid, \$15.5 million, for 39 different patents concerning web services (Gilbert). Companies like Google, Amazon, and eBay are also providing inexpensive web services to the community. For example, Amazon’s Simple Queue Service (SQS) is a web service that distributed applications can use for intra-communication. The Amazon SQS has a simple application programming interface (API), and can be accessed by any program much like any other queue is accessed. The advantage here is that, using web services, an application that might otherwise be tightly coupled to perform communication is now loosely coupled; each part of the application need only know the location of the queue on the web.

Service oriented architecture is a new technology which has been introduced in the past few years. Still in its infancy stages, much work needs to be done before it is perfected. Since reusability is a key factor in getting systems put together quickly, SOA is an excellent choice. The future is unforeseeable, but with the use of SOA while designing, systems will be much more reliable and have a longer lifespan.

Bibliography

Bruegge, Bernd and Allen H. Dutoit. Object-Oriented Software Engineering Using UML, Patterns and Java. 2nd Edition. New Jersey: Prentice Hall, 2004.

"Common Object Request Broker Architecture." Wikipedia, The Free Encyclopedia. 28 Apr 2006, 14:20 UTC. 1 May 2006, 04:49 <http://en.wikipedia.org/w/index.php?title=Common_Object_Request_Broker_Architecture&oldid=50585203>.

Gilbert, Alorie. *Web services patents fetch \$15.5 million*. 6 Dec 2004. <http://news.com/Web+services+patents+fetch+15.5+million/2100-1038_3-5480341.html>

Morgenthal, JP. *Reuse Versus Performance in SOA*. 22 Sept. 2005. DMReview.com <http://www.dmreview.com/article_sub.cfm?articleId=1037865>.

"Object request broker." Wikipedia, The Free Encyclopedia. 28 Apr 2006, 01:39 UTC. 1 May 2006, 04:48 <http://en.wikipedia.org/w/index.php?title=Object_request_broker&oldid=50515013>.

"Service-oriented architecture." Wikipedia, The Free Encyclopedia. 30 Apr 2006, 11:26 UTC. 1 May 2006, 04:44 <http://en.wikipedia.org/w/index.php?title=Service-oriented_architecture&oldid=50876544>.

West, Kenneth. *SOA vs. Corba: Not Your Father's Architecture*. 6 Jul. 2004. InformationWeek.com <<http://informationweek.webservicespipeline.com/management/22103843>>.