Service Oriented Architecture: A Design Style

Edward Calusinski

IBM Financial Services Sector Americas
Chicago, Illinois 60611
epcalus@us.ibm.com

Abstract

Service oriented Architectures (SOA) and Web Services have been widely discussed in the recent past. Often these two terms are used interchangeably to describe how IT systems are designed using a service oriented model. Many people may not see the distinction between Web Services and SOA, nor realize that SOA is not dependent on Web Services. Service oriented architecture has existed for years without using Web Services technologies. Organizations have implemented systems based on the SOA design style using data warehousing, message oriented middleware and distributed object technologies, well before the advent of Web Services. This paper intends to provide a basic introduction to the SOA style of design by defining the term, tracing its origins, and offering insight to the benefits of this design style. We will then examine three distinct technologies and how the SOA style of design can be used when implementing a solution. During the examination we will identify the strengths and limitations of each technology and offer alternatives to compensate for the limitations. We will conclude by offering some final thoughts, areas of further research and the future of this design style.

1. Introduction

The Service Oriented Architecture (SOA) style of design has influenced enterprise architects for quite some time. But what exactly is the service oriented architecture style of design? To understand the SOA design style we first must understand the terms that comprise SOA, namely architecture, service, and service oriented. The term architecture is a widely used term in various industries, including the information technology industry. Table 1 outlines the various definitions for the term architecture:
Table 1 – Defining the Term Architecture

As table 1 demonstrates, the term architecture can be applied to several contexts. The context of civil engineering can be seen in definition one and two. Definition five describes the term in the context of computer science. At the surface, definition five might appear to be the most appropriate definition for SOA given the common context of computer science. However, it is definition three that best describes architecture as it relates to SOA. Architecture in the SOA context is a style and method of design. The term service also has many definitions depending on the context. In the SOA context the term service can be defined as a function that is well defined, self contained and independent of any other service [5]. With this knowledge of the definition for a service, we define the term service oriented as the act of designing with the intent of expressing a feature or function as a consumable service.

Up to this point we have provided a definition for architecture, service and service oriented. Now we are able to offer a definition for Service Oriented Architecture:

Service Oriented Architecture is an architectural design style that expresses computing features as a collection of services. The services are described external from the service implementations; they are stored in a service repository and are consumable.

The exact origin of the SOA style of design is unknown. Mowbray describes the use of object oriented architecture paradigms [1] as a means of separation between architecture, interfaces and implementation. Seevers devised a methodology for the identification of service based enterprise architectures [2]. Gartner used the term Service Oriented Architectures in 1997 when describing an architectural principle in order to enable CORBA [3] and DCOM [4] programming model coexistence in the enterprise. The Object Management Group designed CORBA as a set of services. Still, based on our research, no undisputable “inventor” of SOA has emerged. The absence of a clear creator of the SOA design style may be attributed to the nature of design style evolution. The
computing industry has seen an evolution of design styles for the past forty plus years. Figure 1 show an approximate chronology of the popular design styles of computing systems.

![Diagram showing the chronology of computing architecture design styles.](image)

Figure 1 – The Chronology of Computing Architecture Design Styles

Regardless of the exact origin, SOA as a design style has existed for some time and has benefited from the lessons learned of its predecessor design styles. Design styles are built upon a set of principles that govern its use. These principles are the characteristics that provide the intrinsic behavior for the style of design. Systems that adhere to a style of design are expected to exhibit the principles of that design style and can realize the benefits those principles offer. The principles of the SOA design style are:

- Modularity
- Encapsulation
- Loose Coupling
- Separation of Concerns
- Composable
- Single Implementation

By designing solutions that follow the SOA style of design, these systems can expect to benefit from these principles. Even though designing solutions for the sake of using a modern style of design might seem interesting, real business value needs to exist to justify its use. The SOA style of design can offer the business community the following benefits:

- Increased business responsiveness
- Increased business flexibility
- Ability to transcend organizational boundaries
- Reduces product development cycle times
• Exposes commodities in business processes
• Increased opportunities for revenue

Along with the benefits to the business community, the IT community can benefit from the SOA style of design as well. SOA can provide the following benefits to the IT community:

• Build services once and use often
• Services are built by contract
• Promotes process consistency
• Allows for localization of function and standardization of cross cutting concerns
• Standardized integration
• Reduces solution complexity

With the understanding of the principles of the SOA style of design and the benefits to the business and IT communities, an IT Architect is able to determine the applicability of SOA as a design style when designing a solution.

Recently Web Services has popularized the term SOA. The computing industry often uses the terms Web Services and SOA interchangeably when describing either a design style or an implementation technology. It is important to know that Web Services is not the SOA, but rather a technology that follows to the SOA style of design. As mentioned in this paper, SOA and its underlying principles have existed much longer than Web Services. Only recently has the Web Services SOA gained so much attention. This is due in part to the significant industry support for Web Services from organizations like IBM, Microsoft, BEA, Oracle, HP and others. It should be noted that as SOA gets popularized by the promises of web services the term SOA continues to morph. the term SOA but this paper will focus on SOA as a style of design.

In the remainder of this paper will examine several technologies that can support the SOA style of design. We will examine how each technology addresses the key characteristics of the SOA design style as well as the limitations each technology imposes.

This paper is organized as follows. Section 2 introduces the technical concepts of service oriented architecture. Section 3 outlines two different SOA models in use today. Section 4 explores the Web Services approach to SOA and section 5 outlines a design that incorporates multiple SOA models. The paper concludes with a wrap up section and outlines various follow-on topics.
2. Service Oriented Architecture

A Service Oriented Architecture (SOA) is a collection of services that:

- are made available
- may be invoked
- offers a clear separation between the implementation and how the service is described

A service is a function that is well defined, self contained and independent of any other service [5]. A service can be a business function such as account open, an IT function such as logging or it can be a user interface function such as a graphical weather map display. Services can contain other services or invoke services in a chaining fashion. The value that is gained by identifying and implementing functionality as services is not for the sake of implementing technology, but rather to promote the principles of the design style and benefit from the values it can offer. A principle of SOA is that there is only one implementation of a service either at runtime, build time or design time [6]. This is a key characteristic because it aids in reducing the effect of having multiple services addressing the same functionality. A service oriented architecture design is not limited to large organizations. Small and medium size organizations can benefit from a well designed solution that follows the SOA style of design. Any organization whose objectives match the principles of the SOA style of design, is a potential candidate organization for implementing a SOA.

A service oriented architecture design must contain several key elements. Figure 2 portrays the basic SOA model and the set of elements that are required for this design style.

The key elements of the SOA are:

- Service Provider
- Service Requestor
- Service Registry

The service providers are systems that offer a computing function that is made available to service requestors. The common way a provider’s service is made available is through a service registry. The process of making the service available to the service registry is called publishing. The registry is typically some form of a repository that contains a listing of all of the services available and the necessary information to invoke the service. The repository usually contains a service description or some metadata about the service. The service description provides the necessary information for invoking the service and any other conditions for its use (API’s, message structures, interface definitions, qualities of service, etc).
The service requestor is the portion of the system that is interested in invoking the services of the service provider. The requestor is responsible for obtaining or finding the service description from the service registry and correctly binding with the service provider. Each SOA design might implement each of these elements differently, but in order for the design to be considered a SOA design, these elements must exist.

When creating a service, several key characteristics are suggested that will enable the service to be utilized effectively. Brown et al. [8] outline these characteristics:
Coarse-Grain
Interface based Design
Discoverable
Single Instance
Loosely coupled
Asynchronous

These characteristics are essential to the design of a service. Each of the characteristics can be traced back to one or more SOA principles which provides integrity to the principles and the characteristics.

**Coarse Grain:**

The level of granularity is a statement of functional richness for a service. The more coarse grain a service is, the richer the function offered by the service. Services are typically coarse grain business functions such as a mortgage application. This is coarse grain because the operation mortgage application might result in the execution of multiple finer grain operations such as get credit report, verify customer identity and employment verification. The value of a service in this context is the reduction in complexity for system developers when executing the necessary steps for applying for a mortgage. The service is identified in meaningful business terminology and allows for the consistent processing of mortgage application operations. The underlying details of the mortgage application process are hidden from the developer, which allows more efficient software development. In the SOA model, a service requester must bind to the service provider and initiate a service request. Coarse grained services typically use a message passing paradigm to exchange information. The message paradigm is used because a coarse grain service usually requires a large dataset in order to complete the request. Also services are designed to be stateless and as a result, all necessary information required to satisfy the request must be contained in the original service invocation. This is in contrast to traditional RPC models where a series of APIs can be invoked and the state is preserved between invocations. Message passing is also preferred because services are inherently distributed. Multiple small interactions in a distributed environment incur excessive latency and can adversely affect system performance.

**Interface Based Design:**

Interface based design allows for the development of a service that is independent of the implementation platform, language and logic. Inability to decouple the service interface from the service implementation causes the developer to be intimately aware of changes made to the implementation. The benefits of interface based design have been proven in the industry by the use of Interface Definition Languages (IDL) such as OMG’s IDL[11].

**Discoverable:**

The ability for a service requestor to discover a service provider is critical for the wide adoption of a SOA design. If the no one knows that a service exists, it is unlikely that the service will ever get utilized. The service registry plays the key role in the discovery of
services and the delivery of the service description. It is important to understand that the registry needs a mechanism that enables potential service requestors to find a desired service, as well as provide a mechanism for delivery of the service description. How a registry is implemented is left to the designers of the SOA solution. It would be completely legitimate for a SOA solution to implement the registry as a published document that provided a textual description of a service, for example a telephone number and email addresses, for those wishing to receive the service description via postal mail or email. More elaborate registry implementations might provide for a fully automated query and delivery mechanism for the service requestors to utilize.

**Single Instance:**
The single instance characteristic helps reinforce the notion that only one implementation of a service should be running. Also, since state is not preserved between invocations, the ability to scale is greatly increased.

**Loosely Coupled:**
Loosely coupled services offer the ability to separate concerns. Dijkstra describes in [12] the importance of separation when programming in complex environments. The separation of concern principle allows for a cleaner handling of the complexity as well as a means to achieve adaptability and scalability of software systems. A major benefit of a SOA design is the separation of concerns to reduce the complexity of today’s enterprise systems. Loose coupling enables a system that can separate the concerns of application features into separate independent pieces. This separation provides a mechanism for a service to call another service without being tightly bound. Message passing is the predominate manner for exchanging information between services, which is further supported by the loose coupling characteristic.

**Asynchronous**
Asynchronous communication is not required, but is consistent with the other service characteristics such as coarse grain services and loose coupling. It would be completely legitimate to design a SOA solution that used synchronous styles of computing and still achieve coarse grained services and loose coupling. A later section of this paper will describe how the distributed object SOA design offers a synchronous communication style. Asynchronous communication offers the benefit of scaling through asynchronous behavior and queuing techniques. Brown et al.[8] identify unpredictable network latency and high communications costs that contribute to slower response time in a SOA solution due to the distributed nature of services. Asynchronous behavior and queuing allows a service to issue a service request while being able to continue processing until a response is returned by the service provider.

Figure 3 illustrates a hypothetical banking example of a mortgage application service. This example illustrates the business services required for executing the mortgage application process. As you can see, there are several layers of services within the design. The mortgage application service requestor initiates a service request to the mortgage application and transmits the mortgage application document for processing. The mortgage application service requestor is completely unaware of the steps necessary to complete the mortgage application process or how each step is designed and
implemented. The figure also demonstrates that the mortgage application service utilizes the capabilities of other services in the overall architecture. Each service within the overall architecture maintains the key characteristics of a service for a SOA design. Also notice that a service accepts requests from the requestors as message documents, but in some cases, such as verify insurance service, alternative technologies are used to fulfill the request.

![Diagram of a mortgage application service](image)

**Figure 3 – Example SOA of a mortgage application**

With a better understanding of the service oriented architecture model and its key characteristics, the next section will examine two different technologies that can utilize the SOA style of design. The examination will expand on the mortgage application example by exploring its use with a message oriented middleware SOA design and a distributed object computing SOA design.
3. Examples of Service Oriented Architectures

In this section we are going to examine how the SOA style of design can be utilized by two different technologies in message oriented middleware (MOM) and distributed object computing.

Message Oriented Middleware Service Oriented Architecture Design

Message oriented middleware allows for a general purpose message exchange between a client application and a server application by utilizing message queues [13]. MOM in essence offers system to system connectivity through an intermediary layer. This intermediate layer provides message queues by which applications communicate with one another, while offering a set of services such as guaranteed message delivery, transactional control, loose coupling between applications and priority based messages. Further advancements of the intermediate layer have been categorized as enterprise application integration and include the advanced concepts of message brokers [14] and process management systems [15]. Figure 4 outlines how the MOM SOA design might look for the mortgage application example presented earlier.

![Diagram of Mortgage Application SOA Design](image)

Figure 4 – MOM SOA Design Mortgage Application Example
At the heart of the MOM SOA design is the Service Hub. Architectural designs that are implemented using EAI technologies often follow the hub and spoke design pattern. The hub-and-spoke design pattern allows for a logical centralized collection point for enterprise services. Service requests enter the hub and a participating spoke application services the request. The EAI services within the hub provide the routing logic along with a series of other capabilities commonly associated with EAI technologies. In this hub and spoke design, requests are placed in the service input queue and the hub provides the necessary routing of the request to the appropriate service provider. For example, the mortgage application request is placed in the service input queue for servicing. The hub determines that the request should be routed to the mortgage application service for processing. Once the mortgage application service accepts the request for processing, it submits requests to the service input queue for identity verification, credit reporting etc. As mentioned earlier, a design that is consistent with the SOA style must satisfy each of the SOA characteristics. Table 2 outlines how the MOM SOA design addresses each characteristic.

<table>
<thead>
<tr>
<th>SOA Characteristics</th>
<th>MOM SOA Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grained</td>
<td>Mortgage Application service is a coarse grained view of the mortgage application process. The underlying process steps are abstracted from the caller.</td>
</tr>
<tr>
<td>Interface Based Design</td>
<td>An XML message based system that utilizes message queues for integrating. The message structure offers the interface constructs.</td>
</tr>
<tr>
<td>Discoverable</td>
<td>Not outlined in figure 4, service discovery would be facilitated through an application repository web interface or provided via an application librarian.</td>
</tr>
<tr>
<td>Single Instance</td>
<td>The notion of queuing provides an inherent single instance view to the client. The implementation of each service would need to adhere to the single instance policy.</td>
</tr>
<tr>
<td>Loosely Coupled</td>
<td>MOM offers a natural loose coupling through the use of queues and by implementing a message passing paradigm for service invocation.</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>MOM and queuing is inherently asynchronous.</td>
</tr>
</tbody>
</table>

Table 2 – MOM SOA Design Characteristic Summary

The SOA design style prerequisite characteristics are nicely satisfied in the MOM model. By utilizing a message passing paradigm, the MOM SOA design is afforded the benefit of satisfying the interface based design in the form of message structures. The message structure provides the application designer with the specification for message construction. This brings up an interesting architectural issue in the MOM SOA design.
Can messaging truly be considered an interface based design? When examining the definition of interface based design, several characteristics emerge. Using the CORBA IDL as the reference for interface design, the value offered by an IDL is that it is an industry standard, platform and language neutral definition language for interfacing components. IDL offers the ability for tools to generate code based on its design and can be placed in a repository for design and run-time support. When examining how the MOM SOA design might use messaging as its interface design, eXtensible Markup Language (XML) [16] emerges has the leading candidate for the implementation. XML offers a language and platform neutral definition for message structures, it is an industry standard and it is supported by numerous tools for generating execution code. By using XML, the MOM SOA design offers a clean approach to solving the interface definition and message protocol problem. Figure 5 provides an example of a possible Mortgage Application XML message.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!--MOM SOA Example XML message-->
<SOA>
  <ServiceType>MOMSOA</ServiceType>
  <ServiceName>MortgageApplication</ServiceName>
  <QualityOfService>
    <SecurityModel>Highest</SecurityModel>
    <NotificationMechanism>Email</NotificationMechanism>
  </QualityOfService>
  <DocumentType>Loan</DocumentType>
  <Application>
    <ApplicantInformation>
      <Name>John Smith</Name>
      <StreetAddress>123 Main St</StreetAddress>
      <City>Chicago</City>
      <State>IL</State>
      <ZipCode>60611</ZipCode>
      <SocialSecurityNumber>123-22-4455</SocialSecurityNumber>
      <EmploymentInformation>
        <SelfEmployed>No</SelfEmployed>
        <EmployerName>IBM</EmployerName>
        <Position>IT Architect</Position>
        <YearsEmployed>10</YearsEmployed>
      </EmploymentInformation>
    </ApplicantInformation>
    <PropertyInformation>
      <StreetAddress>123 Main St</StreetAddress>
      <City>Chicago</City>
      <State>IL</State>
      <ZipCode>60611</ZipCode>
      <County>Cook</County>
      <EstimatedValue>450,000</EstimatedValue>
      <InsuranceInformation>
        <InsuranceCompany>ValueInsurance</InsuranceCompany>
        <AgentName>Bob Agent</AgentName>
        <AgentPhoneNumber>312-555-1212</AgentPhoneNumber>
      </InsuranceInformation>
    </PropertyInformation>
  </Application>
</SOA>
```

Figure 5 – Sample MOM SOA XML Message for Mortgage Application Service
As figure 5 shows, the XML based message structure provides a clean separation of concerns between the various pieces of data required for the mortgage application process. Service based information is isolated from applicant information and property information. This type of construct helps facilitate the decomposition of information into the other message types required in the overall SOA solution. Since all service requests are managed by the EIA services layer of the services hub, XML standard compliant tooling can be utilized for the routing and transformation of the mortgage application information.

The MOM SOA design does have drawbacks. MOM is often considered proprietary, because very few standards exist to govern its advancements. This issue might affect the extensibility of a solution that uses the MOM SOA design. The MOM SOA design also does not have a standard discovery service. For those solutions that require dynamic service discovery and invocation, a custom built discovery service would be required. The implication of custom building a discovery service is that the newly created solution would be proprietary as well.

Distributed Object Service Oriented Architecture Design

Distributed object computing has been, and continues to be, a significant design approach when designing enterprise solutions. Technologies like Common Object Request Broker Architecture (CORBA), J2EE and .Net offer designers and developers an elegant component based approach to solve application integration issues in the large. Object technologies have matured significantly over the years from being a viable design choice for small stand alone applications, to being a viable enterprise wide design alternative for distributed applications. Orfali and Harkey describe distributed objects as encapsulated chunks of code that are accessible from anywhere in the enterprise [17]. Components are the primary constructs by which a distributed object SOA design is built. “Components are standalone objects that can plug-and-play across networks, applications, languages, tools and operating systems.” [17]. CORBA, J2EE and .Net can be view as distributed component architectures and thus well suited for the SOA model. From this point forward we will treat objects and components equally. Most distributed object architectures provide a set of object services that enable objects to work in a distributed manner. Services like transaction services, persistence services, security services and directory services provide the necessary infrastructure components for true distributed object behavior. This is an important feature of distribute object computing because these services are some of the services necessary for a service oriented architecture.

Figure 6 shows how the distributed object service oriented architecture might be designed for the mortgage application example. As you can see, the familiar business services for the mortgage application are expressed as distributed components. The service requestors interface with the service providers by constructing service request objects and passing them to the service provider components. The service providers then internalize the request objects and service the requests. This design is consistent with the principles of a SOA design. The client requesting the service is oblivious to the actual implementation, they are only aware of the necessary interfacing contract between the requestor and provider. In the distributed object model, the distributed object bus provides a great deal of the infrastructure services necessary to implement the SOA design. The bus contains the directory service and the services required to bind two
components and transport the request objects. The bus also contains the other advanced services like transaction, security, and persistence. These services become increasingly more important when designing solutions for the enterprise that follow the SOA style of design. Ideally the distributed object infrastructure vendors will provide these services and adhere to industry standards in their implementations. CORBA and J2EE vendors do offer these services and adhere to their respective standards. Microsoft offers the same set of distributed object services with their .Net offering, but this solution is more proprietary in nature.

Table 3 provides the distributed object SOA characteristic summary. The distributed object SOA design, much like the MOM SOA design, offers a nice mapping to the characteristics required for a SOA solution. The component nature of the distributed object SOA design provides a coarse grained representation of the services offered. Achieving enough coarseness in the service representation is a key driver why a component model is used. The J2EE standard uses the JavaBeans and Enterprise JavaBeans [18] for its component model and .Net utilizes COM (COM, DCOM, COM+ and ActiveX) for its component model. CORBA utilizes the CORBA components.
specification [19] for its component model. Each of the distributed object technologies utilize an IDL for describing the services in a neutral manner and this IDL can be stored in the service repository for querying purposes. The service repository provides the discoverability of services and would be typically built using the directory service component of the distributed object architecture bus. The directory service can be a proprietary service implementation or built upon industry standards. Regardless of the implementation, the requestors and providers must agree upon the specification used when interfacing with the directory service.

<table>
<thead>
<tr>
<th>SOA Characteristics</th>
<th>Distributed Object SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grained</td>
<td>Component based design of the Mortgage Application services offers a coarse grained view of the mortgage application process. The underlying process steps are abstracted from the caller.</td>
</tr>
<tr>
<td>Interface Based Design</td>
<td>Distributed Object Architectures (DOA) use an Interface Definition Language for interface based design.</td>
</tr>
<tr>
<td>Discoverable</td>
<td>Not outlines in figure 6, service discovery would be facilitated through a directory service for the DOA. CORBA, J2EE and .Net offer directory services for their architectures.</td>
</tr>
<tr>
<td>Single Instance</td>
<td>Component based design in the DOA can facilitate the single instance policy. It is up to the implementers to adhere to the policy.</td>
</tr>
<tr>
<td>Loosely Coupled</td>
<td>IDL offers a loose coupling between design and implementation details and describes the message passing mechanism.</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>The component designers must provide some provisions for asynchronous behavior. Distributed object architectures are typically synchronous in nature.</td>
</tr>
</tbody>
</table>

Table 3 – Distributed Object SOA Characteristic Summary

The single instance characteristic can be satisfied through the use of components and by instituting a policy for designers and developers to follow. For example, an EJB Session bean is a Java component that can facilitate the single instance model by creating the session bean as stateless bean. However there is nothing to prevent the designer or developer from implementing the session bean as a stateful bean, thereby violating the single instance characteristic. By utilizing IDL, the distributed object SOA design can achieve the loosely coupled characteristic. The IDL provides the description of the contract between the service requestor and service provider. The contract will dictate what the message structure should look like when communicating between requestor and provider. In the case of the distributed object SOA design, the message structures are the service request objects. IDL offers a clean separation between description of the service
and the actual implementation. Each of the distributed object technologies support an IDL, however they are not compatible with one another. This brings up an interesting technical decision when designing a distributed object SOA solution. If the IDL for the predominate distributed object technologies are not compatible, what happens when two distributed object SOA designs built utilizing different distributed object technologies need to be integrated? Some higher level specification would need to be established and agreed upon in order to facilitate DOA collaboration. A distributed object SOA solution can achieve asynchronous behavior, but it is a design decision when creating the components. The distributed object SOA design approach can offer very robust and scaleable solutions. Each of the three mentioned distributed object technologies offer a rich set of services to help facilitate SOA design. The transaction, security and persistence services are very impressive and they provide a solid foundation by which to build distributed systems. The biggest challenge facing the distributed object SOA design is interoperability between the models and the tight coupling inherent to object computing. Each of the three distributed object technologies offer a full suite of services required when building a SOA solution and there is little incentive to provide rich interoperability between the vendors. The competitiveness between the distributed object technologies dates back to the early 1990’s and very little movement has occurred in offering significant changes to the technologies to support interoperability. Interoperability is a great concern for customers because most organizations have to deal with the issue of integration between two or more of these distributed object technologies. The issue of interoperability has become such a huge issue, that the creation of a standardized approach for dealing with distributed object technology interoperability has been established. This approach is called Web Services.

4. Web Services SOA Design

There is little doubt that Web Services is the most talked about service oriented architecture technology today. Almost all general literature on Web Services describes web services in the context of service oriented architecture. For most people Web Services is the service oriented architecture, but as we have seen in this paper, there are alternatives. In this section we will describe Web Services as a SOA design and address its benefits and drawbacks.

Web Services Overview

Graham et. al. describe the workings of Web Services, “The service provider is responsible for creating a service description, publishing that service description to one or more service registries, and receiving Web services invocation messages from one or more service requesters” [20]. This description matches perfectly to the description of a service oriented architecture. It is clear the when the creators of Web Services were designing its model, the SOA design style was the intended target. Graham continues to describe the Web Services approach as “an application integration concept; it is a set of technologies that provides access to business functionality, such as purchase order processing. Often, the business functionality already exists in the form of legacy transaction processing systems, existing web applications, Enterprise Java Beans, and so on. Web services technology is about access and application integration; it is not an
implementation technology” [20]. Access and application integration is the essence of Web Services. But the last sentence is very interesting when Graham states that Web Services is not an implementation technology. MOM and distributed object technologies are implementation technologies that can be utilized in a service oriented architecture design. So if Web Services is not an implementation technology, then what is it? Web Services is a set of standards that can be applied to solve the application integration problem in an open and consistent manner. This approach is significantly different than the distributed object SOA design and MOM SOA design. Web Services provides the standards by which interoperability can be achieved. Figure 7 shows the Web Services standards utilized in the SOA model.

Service providers describe their services in the Web Services Definition Language (WSDL) [21] format and submit their services to the registry via the Universal
Description Discovery and Integration (UDDI) [22] protocol. Service requestors query the repository via the UDDI protocol and extract the WSDL for those services they are interested in invoking. The Service requestor interrogates the WSDL and determines the necessary service location, message structure and communication protocols for communicating with the service provider. The service requestor binds to the service provider via the established protocol and traditionally uses SOAP [23] for message encoding. Figure 8 shows the Web Services SOA mortgage application example. The Web Services design is very similar to the basic SOA design. In Figure 8, you can see that the familiar mortgage application services are presented as web services.

The mortgage application client queries the UDDI Registry for a mortgage application web service. Upon discovery of the service, the client examines the mortgage application WSDL, composes the mortgage application document, binds to the mortgage application service and transmits the request. The process of utilizing Web Services is quite simple. What is not expressed in figure 8 is how the services are implemented. Earlier Graham et al. indicated that Web Services is not an implementation technology. So what technology can be used when implementing Web Services? Since Web Services is a standard, any
technology that supports the Web Services specification can be used to implement a Web Services SOA solution. Examples include Java, C++, WebSphere, SAP, DB2, Tivoli, and MQSeries. Table 4 outlines how Web Services satisfies the SOA characteristics. The Web Services model does not contain constructs that enable coarse grained design. As indicated earlier, Web Services is not an implementation technology. One technology used to implement Web Services, EJBs for example, must have a coarse grain construct or the responsibility to adhere to the coarse grained principle falls on the service designer and developer.

<table>
<thead>
<tr>
<th>SOA Characteristics</th>
<th>Web Services SOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coarse grained</td>
<td>Web Services has no provision to force coarse grained service development. Good design practice is required.</td>
</tr>
<tr>
<td>Interface Based Design</td>
<td>Web Services utilizes WSDL for its interface definition.</td>
</tr>
<tr>
<td>Discoverable</td>
<td>As shown in figure 8, the UDDI registry provides the discovery services for the Web Services SOA.</td>
</tr>
<tr>
<td>Single Instance</td>
<td>Web Services model does not have an enforcement mechanism of the single instance principle. It is the responsibility of the designer and implementer to adhere to the single instance principle.</td>
</tr>
<tr>
<td>Loosely Coupled</td>
<td>WSDL offers a loose coupling between design and implementation details and describes the necessary message passing mechanism.</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>The Web Services specification provides the capability to communicate asynchronously.</td>
</tr>
</tbody>
</table>

Table 4 – Web Services SOA Characteristic Summary

Interface based design is easily satisfied in the Web Services model through the use of WSDL. WSDL provides all of the necessary information about the service. This includes the location of the service, what is required to communicate with the service, how to communicate with the service and non-functional capabilities offered by the service. WSDL is a standards based definition language that is universally accepted. The adoption and standardization of WSDL is one of the most critical steps in the use of Web Services. Figure 9 shows an example of WSDL for the mortgage application example. At first glance, you can see that WSDL is based on XML and as such, inherits all of the benefits of XML. A closer examination reveals the major sections of the WSDL. Web Services are defined using six major elements:

- **types**, which provides data type definitions used to describe the messages exchanged.
- **message**, which represents an abstract definition of the data being transmitted. A message consists of logical parts, each of which is associated with a definition within some type system.

- **portType**, which is a set of abstract operations. Each operation refers to an input message and output messages.

- **binding**, which specifies concrete protocol and data format specifications for the operations and messages defined by a particular portType.

- **port**, which specifies an address for a binding, thus defining a single communication endpoint.

- **service**, which is used to aggregate a set of related ports.
<?xml version="1.0" encoding="UTF-8"?>
<definitions name="MortgageApplicationBean"
xmlns="http://schemas.xmlsoap.org/wsdl/
xmlns:xsd= http://www.w3.org/2001/XMLSchema"
xmlns:xsd1= http://mortgageapplication.soa.ibm.com/


<import location="Property.xsd" namespace="http://mortgageapplication.soa.ibm.com/

<import location="Insurance.xsd" namespace="http://mortgageapplication.soa.ibm.com/

<import location="Applicant.xsd" namespace="http://mortgageapplication.soa.ibm.com/

<import location="Employment.xsd" namespace="http://mortgageapplication.soa.ibm.com/

<! -- Message Definitions - - >
<message name="ApplicationStatusRequest">
  <part name="Appnumber" type="xsd:string"/>
</message>
<message name="ApplicationStatusResponse">
  <part name="result" type="xsd:string"/>
</message>
<message name="ServiceRequestRequest">
  <part name="m" type="xsd1:MortgageApplicationDocument"/>
</message>
<message name="ServiceRequestResponse">
  <part name="result" type="xsd:string"/>
</message>
<! -- Port Type Definitions - - >
<portType name="MortgageApplicationBean">
  <operation name="ApplicationStatus" parameterOrder="Appnumber">
    <input message="tns:ApplicationStatusRequest" name="ApplicationStatusRequest"/>
    <output message="tns:ApplicationStatusResponse" name="ApplicationStatusResponse"/>
  </operation>
  <operation name="ServiceRequest" parameterOrder="m">
    <input message="tns:ServiceRequestRequest" name="ServiceRequestRequest"/>
    <output message="tns:ServiceRequestResponse" name="ServiceRequestResponse"/>
  </operation>
</portType>
<! -- Service Description - - >
<service name="MortgageApplicationBeanService">
  <port binding="binding:MortgageApplicationBeanBinding" name="MortgageApplicationBeanPort">
    <soap:address location="http://localhost:9080/MortgageApplicationWeb/servlet/rpcrouter"/>
  </port>
</service>
<! -- Binding Description - - >
<binding name="MortgageApplicationBeanBinding" type="interface:MortgageApplicationBean">
  <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <operation name="ServiceRequest">
    <soap:operation soapAction="" style="rpc"/>
    <input name="ServiceRequestRequest">
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://tempuri.org/com.ibm.soa.mortgageapplication.MortgageApplicationBean"
        parts="m" use="encoded"/>
    </input>
    <output name="ServiceRequestResponse">
      <soap:body
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://tempuri.org/com.ibm.soa.mortgageapplication.MortgageApplicationBean" use="encoded"/>
    </output>
  </operation>
</binding>
</definitions>

Figure 9 – WSDL Example for Mortgage Application Service
This example shows that WSDL offers a very clean and concise approach for interface definition as well as offering all of the necessary features of a robust interface definition language. By utilizing WSDL, the Web Services SOA design is forced to expose it services as interfaces and therefore is able to adhere to the interface based design characteristic. The Web Services SOA design offers a registry service for discovering services. The most common implementation for a Web Services registry is the UDDI registry. This is by no means the only implementation mechanism, but UDDI does represent the standards based approach to service discovery. Web Services does not inherently offer a facility to ensure that the single instance characteristic is followed. The single instance characteristic can be achieved in the Web Services SOA design by enforcing a common name space when designing applications as well as ensuring that implementers produce services that are reentrant or stateless. The Web Services model and WSDL provide an elegant approach to loose coupling through its use of message based passing and the interface design which decouples the implementation from the description. WSDL provides even greater levels of loose coupling by providing transparency when dealing with binding issues. The binding description in WSDL provides a nice abstraction to the developer in the areas of transport protocols, communication behavior and message encoding when invoking services. The Web Services SOA design allows for asynchronous communication by either using the one-way style of communication or by using the request-response style with the service provider returning a response before the completion of the service. In figure 9, we defined the services as request- response. This can be seen in the port type definition section. The client can invoke the service request operation and pass in a ServiceRequestRequest message and in return, the service request operation will pass back a ServiceRequestResponse message. In the actual implementation, the service provider will accept the request, generate an application number, place it in the ServiceRequestResponse message, pass the message back to the client and then the service can continue processing the service request asynchronously. With the application number, the client can query the service for status of the mortgage application by calling the ApplicationStatus operation.

5. Bringing It Together

We have demonstrated that the service oriented architecture design style can be implemented by utilizing several different technologies. Each of the three techniques described in this paper provides their own advantages and disadvantages. When designing a solution that follows SOA design style, no single technique can satisfy all situations. Many factors can influence the design of a solution, and as such, each design will likely be unique. In this section we will examine a proposed way to approach the design of a solution that leverages the benefits of the three technologies discussed thus far.

The MOM SOA design offers a design that is well suited for applications that are heterogeneous in platform, that utilize multiple programming languages or that are self contained. This is because the style of integration is based upon the simple construct of message queues. To be considered as a service provider in the MOM SOA design, all the system needs is the ability to accept messages from a message queue and adhere to the
agreed upon message structure. In some cases even the message structure can be variable, because the EAI hub can provide a robust message transformation facility to transform messages from one format to another. In the DO SOA design, a rich set of infrastructure services are offered by the distributed object bus. The directory, security and transaction services found in the distributed object bus provide the essential foundation for building an enterprise class solution based on the SOA design style. The DO is often preferred over the MOM model, because the MOM model lacks a rich set of standardized services. Integration can be more challenging in the DO SOA model when compared to the MOM SOA model. In order to integrate in a pure DO SOA, application support for a specific DO technology is required. With multiple viable DO technologies available today, the integration issue can become costly for the application vendors wanting to support multiple DO technologies. The Web Services SOA design offers universally accepted standards for describing services and the protocols for communication, but suffers from not being an implementation technology.

In examining the strengths and weakness of each SOA, we can see that these three models complement one another quite nicely. We can also envision a very robust SOA solution built using all three models. By leveraging the integration capabilities of the MOM model, blended with the implementation capabilities of the DO model, wrapped with the universally accepted standards for service description and delivery of Web Services, a very robust, extensible, loosely coupled service oriented solution can be designed. Figure 10 shows how you could combine the three different SOA designs in a way that leverages the
best aspects of each. For those services that need to utilize message queues for integration, the EAI hub can provide those capabilities. Services that are developed using a component architecture like EJBs can benefit from the distributed object SOA design. When exposing services externally, utilizing the benefits of WSDL and Web Services offers the best solution. For those services that are newly developed, or can accommodate change, standardizing on the Web Services model provides an extensible option.

6. Conclusion
We have seen that it is possible to develop a solution that follows the service oriented architecture style of design using a number of different technologies. A solution design requires certain characteristics to be considered a SOA design. These characteristics promote the principles associated with the SOA design style. By adhering to this style, business and IT communities can realize the benefits that the SOA style of design has to offer. Depending on the constraints of the architecture and the requirements of the solutions, a MOM, DO, Web Services or combination SOA design could provide the necessary framework for designing the enterprise solution.

The SOA designs presented here are not the only SOA designs used in industry today, but they do represent the more common approaches to SOA design. It should be noted that this paper does not address all the issues that an Architect might face when designing a
SOA solution. Significant architectural concerns like security, systems management, non-functional characteristics etc were not addressed here, but can influence the design considerations for the Architect when designing a SOA solution.
References


