Published on **ONDotNet.com** (http://www.ondotnet.com/)
http://www.ondotnet.com/pub/a/dotnet/2003/08/18/soa_explained.html
See this if you're having trouble printing code examples

Service-Oriented Architecture Explained

by Sayed Hashimi
08/18/2003

Introduction

SOA (service-oriented architecture) has become a buzzword of late. Although the concepts behind SOA have been around for over a decade now, SOA has gained extreme popularity of late due to web services. Before we dive in and talk about what SOA is and what the essentials behind SOA are, it is a useful first step to look back at the evolution of SOA. To do that, we have to simply look at the challenges developers have faced over the past few decades and observe the solutions that have been proposed to solve their problems.

The Problem

Early programmers realized that writing software was becoming more and more complex. They needed a better way to reuse some of the code that they were rewriting. When researchers took notice of this, they introduced the concept of modular design. With modular design principles, programmers could write subroutines and functions and reuse their code. This was great for a while. Later, developers started to see that they were cutting and pasting their modules into other applications and that this started to create a maintenance nightmare; when a bug was discovered in a function somewhere, they had to track down all of the applications that used the function and modify the code to reflect the fix. After the fix, the deployment nightmare began. Developers didn't like that; they needed a higher level of abstraction.

Researchers proposed classes and object-oriented software to solve this, and many more, problems. Again, as software complexity grew, developers started to see that developing and maintaining software was complex and they wanted a way to reuse and maintain functionality, not just code. Researchers offered yet another abstraction layer to handle this complexity -- component-based software. Component-based software is/was a good solution for reuse and maintenance, but it doesn't address all of the complexities developers are faced with today. Today, we face complex issues like distributed software, application integration, varying platforms, varying protocols, various devices, the Internet, etc. Today's software has to be equipped to answer the call for all of the above. In short, SOA (along with web services) provides a solution to all of the above. By adopting a SOA, you eliminate the headaches of protocol and platforms and your applications integrate seamlessly.

Key Components of SOA

The first step in learning something new is to understand its vocabulary. In the context of SOA, we have the terms *service*, *message*, *dynamic discovery*, and *web services*. Each of these plays an essential role in SOA.

Service

A service in SOA is an exposed piece of functionality with three properties:

1. The interface contract to the service is platform-independent.
2. The service can be dynamically located and invoked.
3. The service is self-contained. That is, the service maintains its own state.

A platform-independent interface contract implies that a client from anywhere, on any OS, and in any language, can consume the service. Dynamic discovery hints that a discovery service (e.g., a directory service) is available. The directory service enables a look-up mechanism where consumers can go to find a service based on some criteria. For example, if I was looking for a credit-card authorization service, I might query the directory service to find a list of service providers that could authorize a credit card for a fee. Based on the fee, I would select a service (see Figure 1.1). The last property of a service is that the service be self-contained.

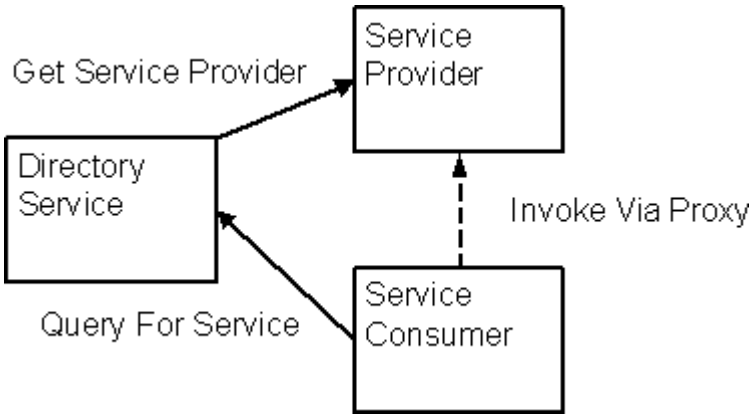

Figure 1.1. Directory service

Message

Service providers and consumers communicate via messages. Services expose an interface contract. This contract defines the behavior of the service and the messages they accept and return. Because the interface contract is platform- and language-independent, the technology used to define messages must also be agnostic to any specific platform/language. Therefore, messages are typically constructed using XML documents that conform to XML schema. XML provides all of the functionality, granularity, and scalability required by messages. That is, for consumers and providers to effectively communicate, they need a non-restrictive type of system to clearly define messages; XML provides this. Because consumers and providers communicate via messages, the structure and design of messages should not be taken lightly. Messages need to be implemented using a technology that supports the scalability requirements of services. Having to redesign messages will break the interface to providers, which can prove to be costly.

Dynamic Discovery

Dynamic discovery is an important piece of SOA. At a high level, SOA is composed of three core pieces: service providers, service consumers, and the directory service. The role of providers and consumers are apparent, but the role of the directory service needs some explanation. The directory service is an intermediary between providers and consumers. Providers register with the directory service and consumers query the directory service to find service providers. Most directory services

typically organize services based on criteria and categorize them. Consumers can then use the directory services' search capabilities to find providers. Embedding a directory service within SOA accomplishes the following:

1. Scalability of services; you can add services incrementally.
2. Decouples consumers from providers.
3. Allows for hot updates of services.
4. Provides a look-up service for consumers.
5. Allows consumers to choose between providers at runtime rather than hard-coding a single provider.

Web Service

Although the concepts behind SOA were established long before web services came along, web services play a major role in a SOA. This is because web services are built on top of well-known and platform-independent protocols. These protocols include HTTP, XML, UDDI, WSDL, and SOAP. It is the combination of these protocols that make web services so attractive. Moreover, it is these protocols that fulfill the key requirements of a SOA. That is, a SOA requires that a service be dynamically discoverable and invokeable. This requirement is fulfilled by UDDI, WSDL, and SOAP. SOA requires that a service have a platform-independent interface contract. This requirement is fulfilled by XML. SOA stresses interoperability. This requirement is fulfilled by HTTP. This is why web services lie at the heart of SOA.

Conclusion

Even though we have decades of experience in software development, we have yet to solve the mysteries of software complexity. As complexity grows, researchers find more innovative ways to answer the call. SOA, in combination with web services, is the latest answer. Application integration is one of the major issues companies face today; SOA can solve that. System availability, reliability, and scalability continue to bite companies today; SOA addresses these issues. Given today's requirements, SOA is the best scalable solution for application architecture.

This article was an introduction to SOA. There is a lot more to SOA than the simple introduction that I have given here. In several future installments, I will discuss the core concepts behind SOA in much more detail. For example, an upcoming article will discuss messages with more breadth. There is a lot more to a message than XML. For example, how do you design synchronous vs. asynchronous messages? How can you ensure reliability? How do you handle security within your messages? These topics and more will be discussed next.

Sayed Hashimi is an independent consultant based in Jacksonville, Florida.

Return to ONDotnet.com