Advertisement:

Service-Oriented Architecture

By Caprio, Griffin

Published in: CODE Magazine: 2005 - May/June
Last updated: February 20, 2019

Service-Oriented Architecture, or SOA, is the newest acronym to become a buzzword among developers, IT Managers, and CTOs.

It seems that everyone is talking about making an SOA and how much it will improve their operations, yet most people are hard-pressed to define not only what an SOA is, but also to quantify what specific value it might provide to their organizations. Many simply assert that their SOA architecture comprises a group of Web Services through which they can expose business logic over the Internet.

A service-oriented architecture is comprised of a number of different services that can be consumed by any number of clients. The only assumption made by either party is that communication takes the form of a well-defined and strictly enforced contract.

Tweet Share

Published in:

> Service providers are components that provide a service that can be used by any number of clients. The only requirement for

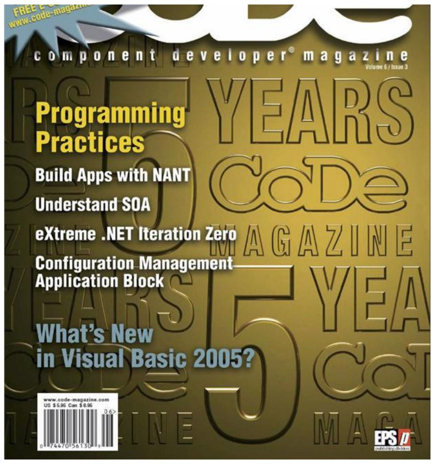
interface.

The purpose of this article is to describe a Service-Oriented Architecture (SOA) in terms of the IT industry. Unfortunately, the term SOA doesn't really have a formal definition. In fact, many people believe that SOA means you have several Web Services exposed to clients. Although this is not necessarily wrong, it's not the whole requirement either.

An SOA is much more than mere Web Services. In fact, Web Services are not a requirement for implementing an SOA; an SOA is about architectural refinement rather than the implementation of one specific technology. This means that an SOA converts your architecture from isolated systems into black box services that can be reused without modification.

Advertisement

To fully implement an SOA, you must move from a monolithic application development model to a publisher/consumer application development model. Developing software to utilize formal contracts and become autonomous removes the dependence on elements outside of the formal contract. Companies


component developer magazine
Programming Practices
Build Apps with NANT
Understand SOA
eXtreme .NET Iteration Zero
Configuration Management Application Block
What's New in Visual Basic 2005?
5 YEARS CoDe MAGAZINE

Filed under:

Architecture Services SOA Web Services

Advertisement:

that is able to provide a large amount of functionality without compromising the core of the system.

The creation of explicit contracts between consumers and providers facilitates a lot of flexibility when accessing the services. Because providers are not directly tied to consumers, contracts allow the service provider to service any consumer, as long as the consumer accesses the service using the defined contract.

Service-Oriented Architectures are Nothing New

SOAs are nothing new. The concepts behind SOAs, providers and consumers, have been around for many years. Most recently, CORBA, DCOM, and RPC-style distributed middleware have been the basis for SOA implementations.

What changed and what is new is the level interoperability and ubiquity that can be achieved. What marred earlier attempts was the presence of propriety interfaces and plumbing, namely the structure of the contract definition and the means by which it was exposed. Typically, the use of previous systems involved several less?than-desirable requirements, including:

- Modifying the network's configuration to open appropriate ports to enable communication
- Extensive, custom two-way integration with every other party involved in the integration
- Large setup costs, both in software and consulting services

These roadblocks have been addressed through XML and SOAP. Aside from making low-level wire communications easier, XML and SOAP have provided a stable base on which to build specifications for other enterprise services, including message security, binary file transmission, transaction and context management, and events. Open consortiums that include many of the big players in software development today, including Microsoft, BEA, and IBM, develop these enterprise service specifications.

> The term SOA has to do with an architectural orientation specifically geared toward providing and consuming services within your application.

Each of these enterprise service specifications begins with the prefix WS-, and as a result, they are collectively referred to as the WS-*, or WS-plunk, specifications. Some of the categories of Web Service specifications, as well as the specifications they contain, are listed in Table 1.

How Web Services are Related to SOAs

It's important to understand that Web Services are not required to create a service-oriented architecture. The term SOA has to do with an architectural orientation specifically geared toward providing and consuming services within your application. However, Web Services are quickly becoming the preferred implementation technology for SOAs.

An SOA is typically implemented using XML and SOAP, but it can just as easily be accomplished using binary servers and e-mail servers.

The presence of Web Services within an architecture does not mean it is an SOA, just as the lack of a Web Service does not mean the architecture is not an SOA.

Service-Oriented Architecture: A Real Life Example

Documents are everywhere. Every process has some form of documentation to go along with it and that documentation needs to be managed. Managing these documents is a chore using traditional options include filing cabinets, manila envelopes, and even binder clips.

It has also long been true that people have looked toward electronic document management as a way to decrease costs and reduce errors when managing documents. Instant access to the electronic versions of their documents is also a must. Although this seems to be a rather simple idea, in practice, it's less than ideal, oftentimes creating new, distinct problems.

One of the biggest problems with electronic document management systems is getting non-electronic documents into the system. For example, if you have documents from a client who doesn't use the same electronic document management system as you, you still need to get his documents into your system efficiently and without pushing requirements back onto the source of the documents.

efficiently or replicating a workflow for a document, for this example I will just concentrate on the inbound aspect of document management. Namely, how to move documents into your system and how to use an SOA help you do this.

A Dilemma

While working for a document management startup, I came across a problem. The system in use was built to use a simple desktop scanner to send single document uploads from the users' desktops to their servers. Although this worked quite well initially, they were constantly being asked if they could handle larger document volumes. Clearly, I needed some way to scale up the document intake, as a single sheet scanner was not sufficient.

> If thinking that an SOA requires Web Services is the biggest misconception, believing that SOAs are a new concept is the second biggest misconception.

The company provided a service that they intended to sell to many different clients, and every client had different requirements for submitting documents. Some of these variable requirements included the source of documents (fax, e-mail, and high-speed scanners) to frequency of input (daily, periodically, and on demand).

The first solution attempted was a set of integration applications created to map each document entry point within the main system. Unfortunately, this resulted in an N times M solution, where N is the number of document

To solve this, I took a step back to take a reflective look at the problem and how it related to the existing system. The problem was not about how to integrate with each client, but rather the more abstract problem of how to input documents into the existing system efficiently.

The Solution

Upon rethinking the problem, the solution became apparent. Inserting documents into the system was a service that should be provided. Once a document input service was defined, the only remaining problem was developing consumers of that service. These consumers could be built by my company or by the client.

This breakthrough came when I thought of the Web site as a separate isolated application from the larger document management system. The question became how to get documents into the system no matter who was the supplier of documents. For example, the system needed the ability to remotely upload documents using nothing but a Web browser.

That's when I realized that we could create a service that processes a package of documents. The package of documents could come from any source, and as long as the package conformed to what the service was expecting, we could process documents from any source. This simple re-architecture was the first step to creating an SOA. Now, instead of writing a direct pipeline into the system for every client, and another for every document delivery type, we just had to be able to create a package of documents and place it where the service could process it.

Creating a package of documents allowed unlimited flexibility in the aspect of processing that was the most volatile: the client's desired document input requirements. The package also allowed us to stabilize the least volatile aspect of the system, namely the insertion of documents into the system.

By separating the delivery and representation of the documents from the processing of those documents into the system, we were able to standardize the expected groups of inbound documents.

An SOA Example Implemented

To implement the architecture, we defined a flexible suite of services, many of which were published publicly. These services included:

- A Web Service that accepted a packaged group of documents, including a metadata manifest describing the package contents
- A Windows Service that processed each package of documents, regardless of the origin of the package
- .NET Remoting objects that would also accept packages of documents, for faster processing. This service was used in the event that a client used a .NET architecture.
- A Web Service that could be used to track the status of documents submitted.

XML was used to construct the manifest contained within the packages of documents. Initially, the system only supported documents in PDF format, but in successive

This meant that no matter where the documents originated, they could be packaged up, created in the relevant manifests, and submitted as a package to the servers. Some of the clients created to accomplish this included:

- A Document upload Web page, allowing users to submit documents using nothing but the Web site
- A bulk uploading client that monitors directories and file shares on the client's computer. The client then packaged those documents and submitted them to the servers.
- An e-mail server integration client, allowing users to e-mail documents directly to the system.
- A fax server integration client, allowing users the ability to fax directly into the system.
-
- In the latter two client examples, those client components allowed our clients to have documents submitted directly into the system without intervention.
-

It's important to remember that the insertion of documents into the system was but one concern. There were other vertical slices that benefited from a provider and consumer model, including:

- Outbound document transmission
- Document workflows
- Document indexing and management
- Address book servicing

Architecture Do's and Don'ts

This article gives a brief overview of what an SOA is and also examined a small example of a SOA in use today. You might be asking yourself if an SOA is right for your environment and, if so, how you can implement it.

Here are some guidelines to keep in mind when architecting your system in order to make it an SOA-friendly environment:

- Don't concentrate on the implementation specifics of who or what will use your system. This means that you shouldn't look too much at internal clients or external clients. Settle on extensible options, while keeping your system defensive on the type of data it consumes.
- Do base interactions on well-defined contractual formats, like XML schemas.
- Don't make assumptions about who or what will be using your service. Your system could be originally used from a Web site, but could easily be required to be accessible from a Windows client or Web Service.
- Do think about your software as autonomous and stateless. Base service contracts on accepting everything needed to fulfill a service request.
- Don't believe that you can think of every possible use for your system. Future requirements could bring new interactions. For example, don't create one service to provide a filtered view of certain data. Instead, create two services, one to provide the data, and the other to

- Do separate your application into tiers. This enables the creation of services at any tier. For example, tiers allow you to expose business logic, as well as data, as services.
- Don't think that the initial UI used to interact with your system is the only way that you will want to interact with your system. If you have a Web application, don't think of the whole system in terms of the Web site, but as a individual system, with one possible front end being the Web site.

Conclusion

Building an SOA is not an easy task. It takes a lot of time and patience to be able to make sound and consistent decisions that keep the larger SOA vision paramount. Architects and developers must resist the urge to circumvent the provider and consumer model in favor of quicker, short term fixes.

Team members must also migrate existing service-based thinking from purely external interactions to other internal aspects of development, including component assembly and interface design. Applying the consumer and provider model to these constructs is complex, and is the subject of future articles.

Above all, remember to build flexibility into your software. This does not mean building in hooks and extension points into your applications. Rather, reduce the dependencies and assumptions of your applications. Reducing the dependencies of your application on current requirements increases the possibility that your software

Service Consumers

Service Consumers are components that "know" how to consume a service. Service Consumers can also act as Service Providers themselves. Service Consumers can consume any number of services and can also aggregate the results of those services, publishing them as another service.

XML and SOAP

XML is a standard for representing text in a structured plain, text format. Because XML is a standard, parsers exist for almost every platform.

SOAP is a messaging standard based on XML. It defines a message construct to be used in the exchange of XML messages between services. Mirroring a real message, it contains constructs for envelopes, addresses, and message bodies.

What is HTTP REST?

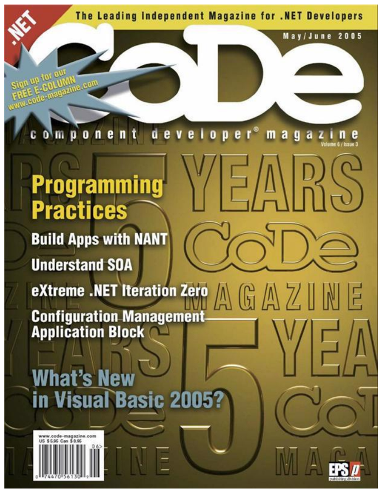
HTTP REST is an alternative Web Service implementation, based on HTTP and the four HTTP verbs: GET, POST, DELETE, and PUT. It is a simpler architecture, requiring little more than HTTP support and XML processing. HTTP REST Web Services are an alternative to the slightly more complex SOAP based Web Services.

Table 1: WS Specifications and their locations

Category	Specifications
Messaging Specifications	SOAP, WS-Addressing, WS-Transfer
Security Specifications	WS-Security, WS-Trust, WS-Federation
Transaction Specifications	WS-Coordination, WS-AtomicTransaction
Business Process Specification	BPEL, BPEL4WS
Metadata Specifications	WS-Policy, WSDL, WS-Discovery

Architecture Services SOA Web Services

This article was published in:


.NET
The Leading Independent Magazine for .NET Developers
May/June 2005
CoDe
component developer magazine
Sign up for our FREE E-COLUMN www.code-magazine.com
Programming Practices
Build Apps with NANT
Understand SOA
eXtreme .NET Iteration Zero
Configuration Management Application Block
What's New in Visual Basic 2005?
5 YEARS CoDe MAGAZINE

Have additional technical questions?

Get help from the experts at CODE Magazine - sign up for our free hour of consulting!

Contact CODE Consulting at techhelp@codemag.com.