

From SOA to SaaS

The driving forces behind today's software architectures



David is an Enterprise Architect and may be reached at david@houilding.net.

"SERVICE-ORIENTED ARCHITECTURE" (SOA) and "Software as a Service" (SaaS) are two driving forces in software architectures today. In this article, I explore how to grow a local SOA into a federated SOA distributed over the Web to both use and deliver SaaS. In the process, I examine how "mashups" relate to SOA over the Web and how, if used inappropriately, they may circumvent some of the benefits of SOA and SaaS.

SOA

Service-oriented architecture has established its value in enterprise systems as an architecture for integration, consolidation, and reuse. This has provided a path to integrate and consolidate legacy systems onto a consistent platform for growth and agility going forward. The building blocks of the SOA are services that all adhere to the pattern in Figure 1.

Ensuring that services adhere to a strict pattern and set of access protocols enables growth in the numbers of services in an SOA without increasing its complexity. This in turn has improved reuse, reduced redundancy, and improved maintainability and flexibility of such systems.

One key aspect of an SOA is that all clients are thin in that they contain no business logic. Rather, business logic is located in services where it can be reused across many different types of clients ranging from web browser UI clients, to IVR clients, to system-level B2B external component-type clients.

Services in an SOA may be accessed using a variety of protocols. For the purpose of this discussion, I focus on web services where clients connect with services using SOAP/HTTP and the interfaces of our services are specified using Web Services Description Language (WSDL).

WSDL is agnostic of implementation language. In the SOA example I present here, the service implementation is in .NET C#.

Figure 2 shows an overview of the application architecture of my SOA example. The Mapping Service is responsible for providing mapping-related services, including geolocation to lookup a

latitude/longitude for an address. The following code, from MappingService.asmx.cs (available electronically; see "Resource Center," page 5), illustrates this:

```
public LatitudeLongitudePoint geocode(
    string street, string city,
    string state,
    string zip, string country )
```

The Address Service is responsible for managing addresses, and provides methods such as *add*, which may be used to add a new address (see AddressService.asmx.cs):

```
public Address add( Address address )
```

In the process of adding, the address geolocation is done to assign a latitude/longitude to the address. The Address Service depends on the Mapping Service for this geolocation as in Figure 2, and as shown by the following code (see AddressService.asmx.cs) from the implementation of the *add* method in the Address Service:

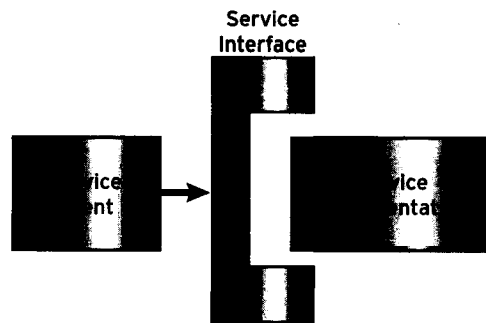


Figure 1: Service pattern.

continued from page 32

```
LatitudeLongitudePoint =
  new MappingService().geocode(
    address.Street, address.City,
    address.State, address.Zipcode,
    address.Country );
address.Latitude = point.Latitude;
address.Longitude = point.Longitude;
```

The Add Address ASP Web Page is responsible for enabling users to enter new addresses, and then add them. Once the address is added, the latitude/longitude coordinates associated with the address are displayed.

SaaS

Software as a Service has established itself on the Web through the provision of APIs; for example, Yahoo Maps (developer.yahoo.com/maps). Here, I focus on SaaS delivered as an API, accessible remotely over the Web at a system level where the consumer is a nonvisual system component. SaaS has enabled new systems to focus on their core value add, while essentially outsourcing supporting services.

This in turn has yielded benefits, including speeding time-to-market and

reduced licensing, maintenance, and administration costs.

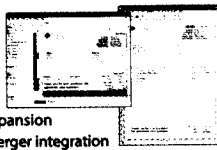
In an Enterprise SOA, where all services are implemented locally and none are outsourced over the Web, implementations are full in the sense that they contain all business logic, or may be adapters that can delegate to a local third-party framework, library, or product. For example, in a local enterprise architecture, the Mapping Service implementation may simply delegate to some third-party product also running locally. (To see a list of such products, search the Web for "geocoding software products")

total output control



DynamicPDF™ ReportWriter v4.0 for .NET - New Version!
Our easy-to-use tool integrates with ADO.NET allowing for the quick, real-time generation of PDF reports.

- GUI report designer • Event driven
- Recursive sub-reports • Bar Codes
- Use PDF templates • Encryption
- Linearize/Fast web view
- Automatic record splitting and expansion
- Full DynamicPDF Generator and Merger integration



DynamicPDF™ Merger v4.0

Our highly efficient library for manipulating PDF documents is available natively for .NET, COM/ActiveX* and Java.

- Intuitive object model • Merge • Stamp • Append • Split
- Place, rotate, scale and clip pages • Form-fill • Add form fields
- Preserve outlines and annotations • Encryption and decryption
- Linearize/Fast web view • Includes the Generator API

DynamicPDF™ Generator v4.0 - Free community editions

Our award winning and highly efficient API for real-time PDF creation is available natively for .NET, COM/ActiveX* and Java.

- Intuitive object model • JavaScript • Security and Encryption
- Linearize/Fast web view • HTML text • PDF/X-1a • ICC Profiles
- One and two dimensional bar codes • Interactive form fields
- Flexible document templates • 48 ready to use page elements

*See our web site for current COM/ActiveX product features

DynamicPDF™ components will revolutionize the way your websites and applications handle printable output. Available natively for .NET, COM/ActiveX and Java.



DynamicPDF
www.dynamicpdf.com

ceTe Software leads the industry in highly efficient and easy-to-use libraries for real-time PDF document creation and manipulation.

ceTe software
INFINITE POSSIBILITIES

800.631.5006 +1 410.772.8620

The Convergence of SOA and SaaS

Fortunately, the service interface/implementation pattern in Figure 1 enables a convenient architectural path to align these two powerful trends and leverage their synergies.

As more services become available over the Web as part of the SaaS trend, service implementations can change so that instead of implementing all logic by themselves or delegating to some local third-party product, they instead delegate to some remote service delivered by some third party.

For example, Yahoo Maps includes an API that you can use to geocode an address (developer.yahoo.com/maps/rest/V1/geocode.html). To make use of this geocoding service, you can simply switch the implementation of your Mapping Service and change the geocode method to delegate to the Yahoo geocode web service instead of processing such requests in our local system (see `MappingService.asmx.cs`):

```
XmlSerializer xs =
  new XmlSerializer(typeof
    ( ResultSet ) );
WebRequest request =
  WebRequest.Create( uri );
WebResponse response =
  request.GetResponse();
ResultType result =
  ( (ResultSet) xs.Deserialize
    ( response.GetResponseStream()
    ) ).Result[ 0 ];
point =
  new LatitudeLongitudePoint
    ( Double.Parse
      ( result.Latitude.ToString() ),
      Double.Parse
      ( result.Longitude.ToString() ) );
```

This change to your implementation from using a local geocoder to using the remote Yahoo Maps API does not change the WSDL interface for the Mapping Service, so

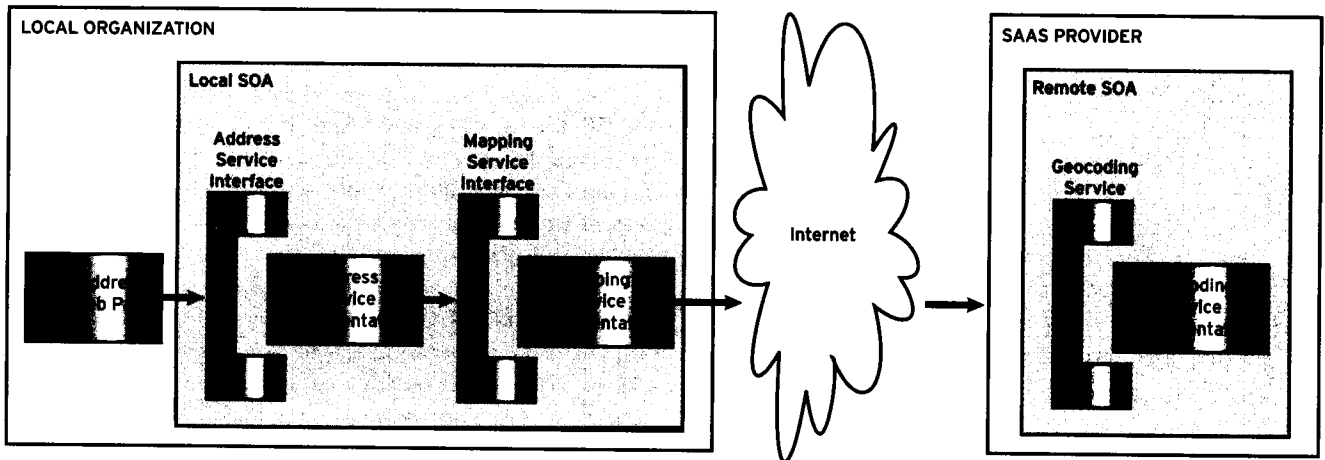


Figure 2: Application architecture.

there is no impact to any of our service clients through this change. This agility afforded by the simple service pattern is extremely important as it enables us to make localized changes to services without changes to the service interface and (more importantly) without causing pervasive rewrites of clients, which are risky and costly.

It is important in an SOA to have a uniform set of protocols for accessing services to improve their ease of use and minimize the complexity of service clients, as well as enable the SOA to grow without adding a lot of complexity due to different access protocols. In my SOA example, all services are accessed using web services with SOAP/HTTP(S). However, note that in the Mapping Service geocode method implementation, I use the Yahoo Maps API with an XML/HTTP REST ("REpresentational State Transfer") protocol. In this way, implementations of my services that delegate out to remote services perform a "glue" function in making remote services with diverse APIs and access protocols accessible locally through a set of services that may all be accessed more easily and uniformly using the same protocol.

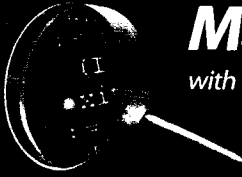
Service Interface Design

You may wonder why we don't just call out to the remote geocoding service from the Address Service. This is because in creating your SOA, you need to decompose groups of functionality into services. Embedding mapping functionality inside an Address Service makes it less accessible and reusable by other clients. Having a separate Mapping Service enables you to reuse just the mapping capabilities directly. For example, if

you wanted to create a map and use geocoding without a full address, you could use your separate Mapping Service directly. Learning how to break down groups of related services for reusability is more of an art than a science, and one that senior architects should play a key role in.

It is also important to design the interfaces of services well so that they don't expose the specifics of their current imple-

mentation. For example, even though the Mapping Service used a local third-party mapping product initially, I didn't expose any specific details of that third-party product through the Mapping Service interface. This let me later change the Mapping Service implementation to deliver its service by delegating to some other API such as Yahoo Maps, without having to change the WSDL interface of the Mapping Service.



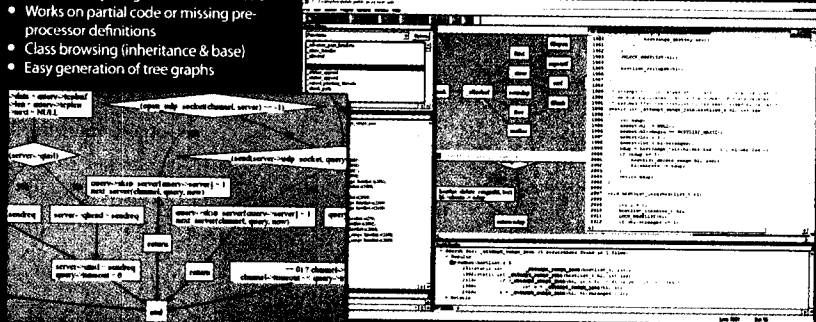
Maintain Your Software with Understand™

You didn't write that mangled code your company picked up in a merger. But it's big, it's ugly, and you need help.

Help is here! Understand helps software developers maintain, measure, document and comprehend source code they didn't write, wrote long ago, or is too large and complex.

What makes us better? Understand quickly parses and manipulates very large amounts of code. One million SLOC projects are common with our users. We also focus on exceptional customer support with rapid response from real engineers. New features and fixes are incorporated into weekly builds.

- Works with K&R C, ANSI C, C++, Java, Ada, FORTRAN, Pascal, Jovial
- Visio export of graphics
- Freely sharable HTML and text reports of analysis information
- Fast on really big projects
- PERL or C API to get data/graphics for custom documentation
- Complexity, Size, OO, other metrics
- Great for exploring code
- Comment extraction for documentation generation
- Easy export of metrics to spreadsheets and databases
- Detailed x-referencing & 1-click visiting
- Info on everything called/used in code
- Works on partial code or missing pre-processor definitions
- Class browsing (inheritance & base)
- Easy generation of tree graphs



FREE Trial Version at www.scitools.com 435.627.2529 Scientific Toolworks Inc.

www.faircom.com/go/?rtp

Chip manufacturing
plants are driven by

**REAL-TIME
PROCESS**
automation systems.

FairCom
database
technology
makes it
possible.

Other company and product names are registered trademarks or trademarks of their respective owners. © 2006 FairCom Corporation

Core Technology

Good service interface design makes all the difference between technical service plugability and practical service plugability. Remember that you can't change your service interface later without forcing a rewrite of clients of that service, which is something you want to avoid by paying careful attention up front to service decomposition and interface design.

Tracking Best of Breed

The other reason you have a separate Mapping Service rather than call out to the remote service from all over your architecture is to localize or isolate dependency on the remote service for agility. Consider a scenario where the Mapping Service proves useful and several clients in the architecture grow to depend on it. To deliver the best service to these clients, over time you want to track the "best of breed" mapping SaaS available. Today, perhaps Yahoo Maps fills this need. However, perhaps in a year, Google Maps comes out with a much better mapping service that we want to use. If you made the mistake of connecting from several points in your architecture directly to the Yahoo Maps API, then later to change to Google would require a "pervasive" rewrite where you have to touch the code in many places to make this change—a risky and cumbersome task.

Rather, by isolating the dependency on Yahoo Maps API behind the Mapping Service interface, you are able to change over to Google Maps with only a simple local change to the implementation of the Mapping Service, avoiding any code changes to the clients of the Mapping Service. This lets you make architectural changes quickly to track best-of-breed SaaS providers, easily and with minimal risk, making the architecture more flexible and business that depends on it more agile.

Mashups

"Mashups" is a term typically used to describe the integration of services at the web page UI level to create rich web applications with lots of functionality behind them delivered by several different web sites. For example, you could create a web page that has maps, news, weather, and other information all drawn from different remote web application services and integrated with a custom layout look-and-feel. In this case, the SaaS is delivered at the point of the UI. This goes against the grain of an SOA in that it locates logic in the UI to access services where it is only useful to the UI client and cannot be reused by other types of clients.

For example, consider a case where you have nonUI clients such as business systems that want to make use of geocoding services to lookup latitude/longitude coordinates for a given address. If you put your logic to access remote geocoding services in a web page UI, this is not accessible to your nonvisual business-system client components. However, if you provide your geocoding access through a Mapping Service accessible as a web service using SOAP/HTTP, then you support its use for all your clients, both UI and nonUI.

In my SOA example, the Address Service depends on the Mapping Service. This is an illustration of the important concept in SOA that services can depend on each other, and new higher level,

more abstract services can build on and reuse or delegate to lower level, more foundational services. This concept is important in the convergence of SOA and SaaS because it illustrates how local SOAs will merge to form federated SOAs and new higher level, more abstract services may be developed that depend on other lower level foundation services that already exist.

In summary, to best leverage SOA and SaaS, you want to do your mashing up or accessing SaaS from behind services, and not in UIs. This approach does not prevent all the benefits of new rich web UIs that leverage powerful mashup tools such as AJAX. It simply means that your AJAX-enabled web UIs talk to your local service interfaces, which may in turn talk to remote services, and your web pages can focus on presentation rather than connecting to remote services.

Challenges

The convergence of SOA and SaaS is not without its challenges and risks.

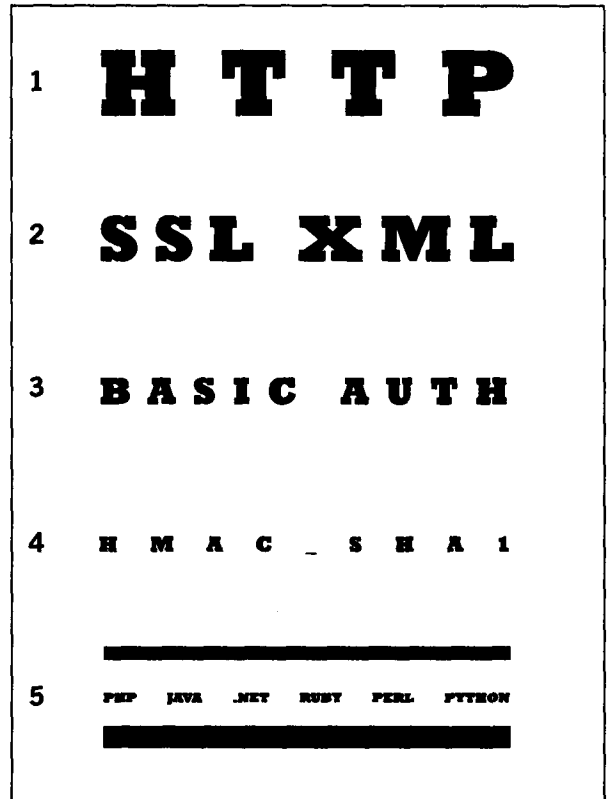
In delegating to remote services over the public untrusted Internet, you often need to be concerned with security. This risk can be mitigated effectively with a combination of application and communications security. Application security typically includes username and password login to the remote service before use, while communications security may be implemented with SSL; for example, HTTPS.

Licensing is another challenge for SaaS. Many of today's SaaS services are offered free. However, for SaaS to grow to its full potential it must be funded, and some providers of services need to get revenue from the use of their services to offset the cost of ongoing hosting and development of new software services. Early forms of such licensing are emerging that involve signing up at web sites associated with the SaaS provider, providing billing information over the Web, receiving a key in response, and sending this key with requests for services. This approach enables the SaaS provider to accurately bill for their service usage. This model of usage moves SaaS toward a utility pay-as-you-go model.

Network and service reliability is yet another challenge with SaaS. When deciding to delegate geocoding over the Web, one has to consider the scenario of network or service outages that could compromise our local Mapping Service and break its clients. Some strategies to mitigate these risks may include redundant network connectivity, so if one channel fails, another will be available. Similarly, behind a service such as the Mapping Service, you may have the capability to delegate to several remote geocoding services. If one goes down, this failure is detected and the Mapping Service implementation automatically fails over to an alternate remote service. This ensures continuity of service to clients of the Mapping Service.

Conclusion

SOA has proven itself in architectures today as a tool to integrate, consolidate, and reuse services. Concurrently, SaaS has established itself as a successful method to deliver business services across the Web. Together, these driving forces may be leveraged as discussed in this article to deliver federated SOA over the Web, and powerful new business capabilities. ■■■



Got 20/20 Vision?

Help sellers integrate Google Checkout and earn a \$25 bonus and 0.5% of their Checkout sales.

<http://checkout.google.com/seller/drdoobbs>

Google
Checkout

© Copyright 2007. All rights reserved. Google and the Google logo are registered trademarks of Google Inc.