software development The lifecycle starts here.

Stairway to Heaven

May 2002

Virtually every enterprise software company has announced support for XML-based Web services in their product suites. While the benefits of a service-oriented architecture are tempting, it's too early to abandon your EAI or middleware solutions.

By Gregor Hohpe

Web services may forever change the way we think about enterprise applications and architectures. Language-, platform- and location-independent processes will easily and politely traverse enterprise firewalls using HTTP and SMTP. They will be self-describing and loosely coupled, meaning that they can spontaneously federate and negotiate terms without human intervention. Sound too good to be true?

These concepts aren't new—they've existed in the form of Remote Procedure Calls (RPCs) and Interface Definition Language (IDL). Web services are simpler, though, depending on the standard protocols that make the Internet possible. While the promise of ubiquitous integration remains to be delivered, Web services have gained enough vendor attention to create a snowball effect: Broader support triggers more momentum, which in turn entices vendors to invest more in Web services.

A Collection of Services

Services are well-defined, self-contained and universally available business functions that respond to service requests from "consumers." At the enterprise level, this concept leads to the notion of a service-oriented architecture (SOA). Service-oriented architectures model the enterprise as a collection of services that are available across the enterprise. Monolithic business applications such as Enterprise Resource Planning (ERP) systems dissolve into a set of self-contained services that perform specific business functions. These services can be invoked over standard protocols to ensure their availability across the enterprise and beyond.

What Is a Service?

- Well-defined
- Self-contained
- Sits idle until request comes
- Does not depend on client context
- No deployment required

Once this infrastructure is in place, developers need no longer build new applications from scratch; rather, they assemble them from services published by internal or external providers. All services are accessed via standard protocols that render them location-independent and are managed through a central registry. Consumers look for desired services in the registry and then retrieve details about a specific implementation from the service's registered description. Finally, the consumer and the service negotiate a suitable interface and transport mechanism to perform the desired function.

A service-oriented architecture has three layers:

Infrastructure Services include security, management and monitoring, administrative functions, data logging and exception handling, as well as registration and discovery.

Business-Neutral Services include service brokers and notification, scheduling or workflow services.

Business Services vary based on the business domain of the application and can include credit card validation, address verification and inventory check.

It's clear that Web services have many properties of a supporting infrastructure for an SOA. They provide a common mechanism for service invocation via SOAP (Simple Object Access Protocol). Based on XML, SOAP is language- and platform-independent, and supports transport protocols that can establish a connection within the enterprise or over the Internet. The registration and discovery of services is managed through Universal Description, Discovery and Integration (UDDI)–compliant registries and Web Services Description Language (WSDL) descriptions.

But while Web services address the basics of a service-oriented architecture, many infrastructure services (security, transaction management, systems management, interface semantics and contracts) and business-neutral services (workflow services, notification services or service brokers) are not yet defined.

This Way to SOA

Though this lack of standardization will slow the pace, a stair-stepping transition from an application-based architecture toward a service-oriented architecture will occur. The process will likely consist of a sequence of plateaus, each with its own technology adoption curve triggered by a new set of standards and implementations.

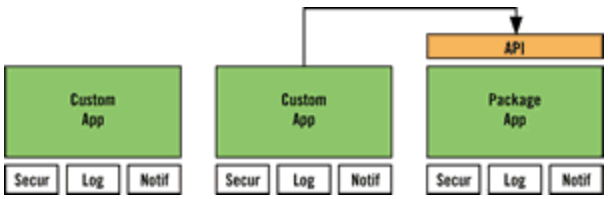
The four plateaus of adoption are:

- integration as an afterthought,
- Web services façades,
- managed Web services and, finally,
- the paradigm shift.

Currently, we're on the Web Services Façade plateau, which is easy to achieve but realizes only a handful of the possible benefits. While the hype engine is running at full speed, we're likely to see little tangible progress over the next months until the next plateau is within reach.

Stage 0: Integration as an Afterthought

The current enterprise landscape consists largely of a collection of completely self-contained custom or packaged applications. Packaged applications may expose functions in a low-level application programming interface (API), allowing some point-to-point integration. Usually, this integration is accomplished by a direct call from one application to a function defined in another's API. Each application uses proprietary security, logging, notification, persistence or other infrastructure services.



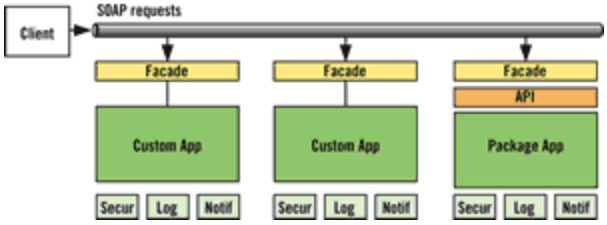
[click for larger image]

The current enterprise landscape consists largely of a collection of completely self-contained custom or packaged applications. Packaged applications may expose functions in a low-level application programming interface (API), allowing some point-to-point integration.

This architecture was the state of affairs before the advent of mainstream enterprise application integration (EAI) tools and Web services. Silos contain valuable data and functionality, but little or no integration is present.

Stage 1: Web Services Façades

The first step in adoption of Web services is to "wrap" existing applications with a Web services façade. Depending on the internal architecture of the applications, the services can be of fine or coarse granularity. In most cases, the façade must manage additional functions such as application-level security in order to provide a self-contained service to the Web services layer.



[click for larger image]

The first step in adoption of Web services is to "wrap" existing applications with a Web services fa çade. This architecture resembles early EAI implementations, but provides the added benefit of standard protocols.

A client application (either a user interface or another back-end system) can now access functions across application packages through a common protocol. This architecture resembles early EAI implementations, but provides the added benefit of standard protocols as opposed to proprietary EAI vendor implementations. On the other hand, current EAI suites include a number of well-integrated tools that are not yet available for Web services deployments.

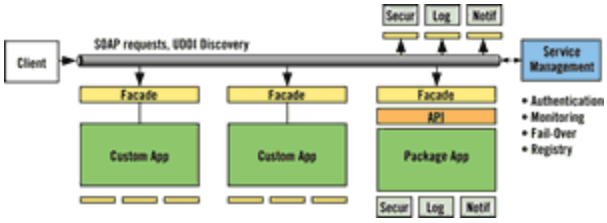
While this step is relatively easy to implement, it lacks many of the features that define a successful enterprise integration:

- No standard translation of data types and structures. Two applications define a "Customer" business object, but each has different notions of address fields; for example, one may have a numeric zip code field, while the other uses a text string. As a result, the client applications must transform the data, increasing complexity and reducing potential for reuse.
- No central service management. This would include monitoring, load balancing, usage statistics and security.
- Proprietary approaches. Each application still implements its own versions of common services such as security, logging, notification and persistence.

Many vendors that claim Web services support provide functionality at this limited level. While compatibility with the emerging Web services standards can be claimed, limited benefit is derived from such a solution. The key driver for implementing such a solution today lies in capability building within the organization. Also, implementing a Web services pilot project using Web services façades would help verify vendor claims by deploying vendor tools within the context of a limited, but real implementation.

Stage 2: Managed Web Services

Implementing some of the common services that make up the lower layer takes application integration beyond simple façades. These infrastructure or business-neutral services (say, for notification or logging) can then be shared by multiple applications. This shift typically requires changes to the application code or architecture. In most cases, package applications are not designed to enable the replacement of underlying services. As a result, this architecture remains a hybrid in which some applications leverage common infrastructure services while others access their own internal services.



[click for larger image]

In most cases, package applications are not designed to enable the replacement of underlying services. As a result, this architecture remains a hybrid in which some applications leverage common infrastructure services while others access their own internal services.

Another critical component of this stage is a central service management that takes care of infrastructure

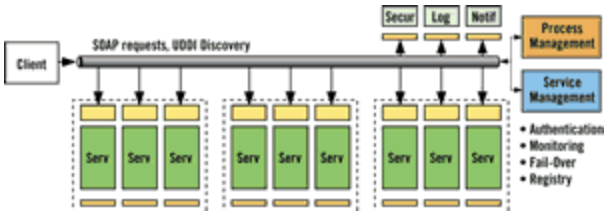
tasks such as monitoring and fail-over to the Web services that are deployed enterprise-wide. This central management provides a common access point to systems management functions across heterogeneous applications. It can also run message broker functions that assist in the translation between application-specific data formats.

These common services and management functions begin to promise tangible benefit for enterprises in which much of the IT budget is spent on operations and monitoring.

Still, the benefits are largely limited to the technical side. Business rules and processes are hidden in multiple applications and can be altered only by making code or configuration changes to the applications. Therefore, this type of administration has to remain the domain of trained IT staff, burdening them and preventing rapid changes to the enterprise business logic.

Stage 3: Paradigm Shift

Businesses will achieve a true service-oriented architecture when they dissolve their monolithic applications into self-contained, well-defined business services. Object-oriented and component-based applications soften the transition. Some application vendors have already announced this type of support, but service-oriented releases of these products cannot be expected for at least another year.



[click for larger image]

In a true service-oriented architecture, all business services use a common set of business-neutral services for logging, notification and security.

In this service-oriented architecture, all business services use a common set of business-neutral services for logging, notification and security. The ability to extend, for example, single sign-on capability across custom and packaged applications greatly simplifies the interaction between services.

Operating on top of the business services layer are centrally managed business processes, a critical component that aims to shift maintenance from the technical into the business domain. While EAI vendors currently offer business process modeling tools and will design some of them to support Web services, new efforts such as IBM's Web Services Flow Language (WSFL) are seeking to standardize this crucial component of the SOA concept.

Bumpy Road Ahead

Service-oriented architectures promise seamless integration of all business functions across the enterprise. This service-level integration can provide significant benefits over retrofitted EAI solutions that are implemented as an afterthought. However, a number of obstacles must be overcome before this seamless integration can become a reality.

The Challenges Ahead

- Evolving standards
- Immature tools
- Semantic mapping
- Network reliability
- Performance
- Application ownership
- Learning curves

Web services standards are amorphous. Many are still working drafts and are likely to evolve further. This could result in rework of existing code or incompatibilities between multiple vendor solutions that comply with different versions of the "standard." Higher-level services such as workflow and security have not been standardized at all.

Development tools are still immature. Most are early releases, built on evolving standards. Many provide proprietary extensions to compensate for patchy standards. This, too, will likely require rework once the standards are completed.

Semantic mapping across applications and businesses is the key to process integration. Consortium-led efforts such as ebXML and RosettaNet are attempting to define common nomenclatures and ontologies that can be used as a lingua franca, but most of these definitions are still constrained to narrow vertical industries and will face increasing challenges as they broaden their scope. In the early 1990s, before enterprise application integration tools arrived, many fell prey to the "Enterprise Object Model that never happened." The key is to start small, focusing on integrations that promise immediate benefit.

The reliability of internal and external network connections becomes paramount in any distributed environment. In a highly interconnected service-oriented architecture, even a partial network failure wreaks havoc on a business. In order to maintain system uptime, the infrastructure has to be highly redundant and robust-and upgrading it may be costly.

Web services also face performance challenges. Transport protocols like HTTP are connectionless and carry performance penalties. XML parsers are getting faster, but still carry more overhead than native data formats. Smart caching and the correct granularity of services are key to high-performance Web services deployment.

And those are just the technical challenges. Organizationally speaking, SOAs blur the boundaries of application ownership. Today's IT departments are grouped around specific applications. In a service-oriented architecture, IT must take a global view.

We may also need to revise software development processes and best practices for in-house development and vendor management to verify internal and external compliance.

Finally, it takes time to learn new and evolving technologies. The endless list of acronyms can be daunting. New tools will make services integration easier, but a thorough understanding of the underlying technologies remains critical during the transition to an SOA.

Your Mileage May Vary

Given the technical and organizational challenges, you should carefully choose your path to service-oriented architecture. It will take time for tools and processes to mature to the point where they can support a SOA.

You may be tempted to launch an enterprise-wide effort right now to build a Stage 3 service-oriented architecture. Any such "fast lane" effort is almost guaranteed to fail. The progression to a service-based, seamless integration requires phases of collective learning. Along the way, many vendor concepts and offerings will become evolutionary relics, replaced by fitter species. Stage 3 will not become a reality until Stage 2 has stabilized and the product offerings have consolidated.

Many of the toughest challenges the transition to SOA will face are not technical. In fact, compared to the standardization, organizational and political challenges, some of the underlying technologies are actually quite simple. This will entice vendors to introduce new products and toolkits at a rapid pace, sometimes precipitously, each claiming to have solved the Web services dilemma du jour. Eager clients will race heedlessly to grab the goodies, neglecting the collective learning process. The age-old balance of knowledge and technique will become a crucial component in the Web services adoption process.

Kicking the Tires

Today, Web services are still a technology, not a business solution. A pilot implementation is the best way to prepare for the upcoming services revolution. Many package and tools vendors claim Web services support at various levels. A limited real-life implementation is the only means of separating fact

from fiction and to assess the value of specific tools in a production environment.

A technology pilot should be as simple as possible, but comprehensive enough to test the remedies promised by vendors. The best candidates for Web services deployment are coarse-grained, stateless services that must be accessed from more than one system.

Initial Web services deployments will face many of the challenges EAI solutions have attempted to address, foremost among them semantic mapping and duplication of information. For example, in the case of credit card validation, does the service receive all the necessary fields (credit card number, expiration date, billing zip code and address) or simply a customer ID? How can this information be represented in a canonical format that is valid across the enterprise? How do applications translate from an internal format into the canonical format? How can the growing set of WSDL descriptions be managed?

Initially, a pilot implementation is likely to generate more questions than answers. Don't despair—the whole point is to generate questions to ask vendors and drive the development of internal standards and guidelines.

Bon Voyage

Service-oriented architectures will grow up gradually, with defined plateaus driven by the adoption of another architecture layer. Initially, immature standards and tools will slow down the process, but the ultimate benefits will be impressive.

Enterprise integration with service-oriented architecture is no slam-dunk. While the benefits are tempting, at this time, it's too early to replace existing EAI or middleware solutions with Web services. First, let's focus on peace-ful coexistence among multiple technologies to prepare for a gradual transition.