Service-Oriented Architecture

A Field Guide to Integrating XML and Web Services

Thomas Erl

PRENTICE HALL
PROFESSIONAL TECHNICAL REFERENCE
UPPER SADDLE RIVER, NJ 07458
WWW.PHPTR.COM

Contents

Chapter 11

CHAPTER
13

Thirty best practices for integrating Web services

Web services introduce technology layers that reside over those already established by the XML platform. Therefore, the best practices for XML provided in the previous chapter also apply to service-oriented environments. Additionally, the contents of this chapter can be further supplemented by extracting best practices from the design and modeling strategies in Chapter 6.

NOTE
All best practices in this chapter are italicized and enclosed in quotation marks.

13.1 Best practices for planning service-oriented projects

13.1.1 Know when to use Web services

"First define the extent to which you want to use Web services before developing them. Services can be phased in at different levels, allowing you to customize an adoption strategy."

If you know that Web services will be a strategic part of your enterprise, then you need to start somewhere. A single application project, for instance, provides a low-risk opportunity for taking that first step. You will be able to integrate Web services to a limited extent and in a controlled manner. The key word here is "limited," because you do not want to go too far with a non-standardized integration effort.

Additional reasons to consider Web services include:

- The global IT industry is embracing and supporting Web services. By incorporating them sooner, your team will gain an understanding of an important platform shift that affects application architecture and technology.
- Use of Web services does not require an entirely new application architecture. Their loosely coupled design allows you to add a modest amount of simple services, without much impact on the rest of the application.
- If you are considering or already using a service-oriented design or business model, you definitely will need to take a serious look at Web services. The benefits of incorporating service-oriented paradigms within your enterprise can motivate the technical migration to a Web services framework.
- Many current development tools already support the creation of Web services, and several shield the developer from the low-level implementation details. This eases

the learning curve and allows for a faster adoption of Web service-related technologies.

13.1.2 Know how to use Web services

"Limit the scope of Web services in your production environment to the scope of your knowledge. If you know nothing, don't service-orient anything that matters."

Although the concept behind Web services has a great deal in common with traditional component-based design, it is still significantly different. Adding improperly designed Web services to your application may result in you having to redevelop them sooner than you might expect.

If you are delivering serious business functionality, you should hold off until you are confident in how Web services need to be integrated. Limit initial projects to low-risk prototypes and pilot applications, until you (and your project team) attain an adequate understanding of how Web services are best utilized within your technical environment.

13.1.3 Know when to avoid Web services

"Even though Web services are becoming an important part of the IT mainstream, you should begin incorporating them only where you know they will add value."

If you don't think that Web services will become a part of your enterprise environment anytime in the near future, then it may be premature to add them now. Technologies driving the Web services platform will continue to evolve, as will the front- and back-end products that support them.

Additional reasons to consider avoiding Web services in the short-term, include:

- The base Web service technologies (SOAP, WSDL, UDDI) are fairly established and robust, but vendor support can vary significantly for second-generation specifications. You may be better off waiting for certain standards to receive industry-wide support.
- Though development tools that support Web services will auto-generate a great deal of the markup, they will not assist in optimizing your application design. Having your developers simply attach one or two Web services to an existing application, without a real understanding of the technology behind them, could lead to a convoluted and weakened architecture.
- Some tools add proprietary extensions that will create dependencies on a vendor-specific platform. The long-term implications of these extensions need to be

understood fully before too much of your application relies on them. Otherwise, opportunities for future interoperability may be compromised.

- Incorporating Web services may simply not be a requirement for autonomous application environments. Web services become a much more important consideration when taking interoperability requirements into account.

13.1.4 Moving forward with a transition architecture

"Consider a transition architecture that only introduces service-oriented concepts, without the technology."

A low-risk solution is an option if your focus is to gain experience with service-oriented technologies and concepts, and you don't have any pressing business requirements that rely on the proper delivery of Web services. You can start your transition by first delivering your application the way you normally would have, and then simply adding application proxies, or a custom designed facade (wrapper) to the functionality you'd like to expose via a service interface.

A benefit to this approach is that you can generally revert back to the traditional component-based model without too much impact to your overall application design. If your project requires a risk assessment wherein the usage of Web services is classified as a significant risk, this can become the basis for a contingency plan.

If you're just toying with the idea of introducing Web services into your application design, but aren't really sure to what extent it makes sense to do so, then you can also consider starting with a feasibility analysis. This will allow you to measure the pros and cons of the Web services platform, as they relate to your development project and your technical application environment.

Alternatively, you could avoid Web services altogether, and still build your application with a future SOA migration in mind. The XWIF modeling process in Chapter 6 provides a strategy for designing traditional component classes into service-oriented classes suitable for Web service encapsulation. These same remodeled classes can still be implemented within a non-service environment. The day you are ready to make the transition, you will already be halfway there.

13.1.5 Leverage the legacy

There are often very good reasons to replace or renew legacy environments in order to bring them into a contemporary framework. A service-oriented integration architecture, however, almost always provides an important alternative.

Build on what you have

"Always consider reusing legacy logic before replacing it."

Through the use of adapters and a service interface layer properly designed for functional abstraction, Web services can let you take advantage of what you already have. This can be a good first step to bringing application logic embedded in legacy systems into your integrated enterprise.

Compared to replacing a legacy environment altogether, leveraging existing systems is extremely cost-effective, and the process of integration can be relatively expedient. (This option also acts as a good reference point for judging the ROI of a proposed replacement project.)

Understand the limitations of a legacy foundation

"Define functional capacity boundaries around legacy applications, and do not integrate beyond."

There are challenges with bringing previously isolated applications into the interoperability loop. Although doing so can immediately broaden the resources shared by your enterprise, it can also severely tax a legacy environment not designed for external integration.

As long as you understand the boundaries within which you can incorporate legacy application logic, leveraging what you have makes a great deal of sense. Incidentally, typical EAI solutions (service-oriented or not) are based on the same principle of utilizing adapter architectures to include various legacy environments. Many mitigate the impact on legacy platforms through the use of intelligent adapters.

13.1.6 Sorry, no refunds (Web services and your bottom line)

"Budget for the range of expenses that follow Web services into an enterprise."

Web services are expensive. That is, *good* Web services require a great deal of work to ensure they truly are "good." Each service you develop can potentially become an important part of your overall IT infrastructure. Not only can services expose legacy applications and various types of business (and reusable) functionality, they can represent and even enable entire business processes.

How a service is designed requires a solid knowledge of the business model within which it will operate, as well as the technologies upon which it will be built. Services that will form (or intend to participate in) a future SOA will also need to be in alignment with the design strategy and accompanying standards that are part of the overall SOA implementation plan.

If you custom-develop services to add on to existing legacy environments, costs will typically be lower than if you start your integration by investing in enterprise service-oriented middleware products. Development costs can be especially moderate when using existing development tools that support the creation of Web services.

Also, because Web services open the door to new integration opportunities, the quality of the interface they expose is very important. Despite being classified as a loosely coupled technology, once heavily integrated into an enterprise, many dependencies upon service interfaces can still be created. Changing an interface after it has been established can be a costly (and not to mention, messy) task, especially in environments that utilize service assemblies.

Doing it right, however, will reap tangible benefits. Integration effort within a relatively standardized SOA will drop significantly. Hooking new and legacy systems into an established Web services-enabled architecture will generally require a fraction of the effort and cost than traditional point-to-point integration projects. There are definite and measurable returns to be had on your investment. It therefore pays to get it right the first time. To get it right the first time, though, you certainly will have to pay.

13.1.7 Align ROIs with migration strategies

Though many organizations have already invested in XML architectures, a move to a service-oriented design paradigm or a full-scale SOA often needs further justification. This is especially true when large investments have already been made in (non-service-oriented) EAI projects with which an organization is already quite content.

ROIs open eyes

"ROIs for service-oriented architectures can provide valuable information, beyond that required to justify the project."

Regardless of whether you are asked to justify the use of Web services or have already decided to implement them in your environment, putting together a realistic ROI can be an enlightening experience.

In addition to providing "evidence" that predicted cost savings resulting from the use of Web services will be realized, properly researched ROIs can give you a clear idea as to how long it will take for these benefits to be attained.

Iterating through ROI and migration strategy documents

"Keep revising the ROI as new information becomes available."

It is common for the results of an ROI to shape an enterprise migration strategy. Flip that thought around for a second, and consider using a migration strategy as input for the ROI.

The nature of the research that tends to be performed for migration strategies is more focused on technology and implementation, rather than high-level organizational benefits. An intelligent strategy for integrating a service-oriented architecture can lead to much greater cost benefits than an ROI originally predicted. So, even if you used an ROI to justify your migration project, you can typically refine that ROI (and often improve the justifications) using the contents of your migration strategy.

Confused yet? Have a look at the diagram in Figure 13.1.

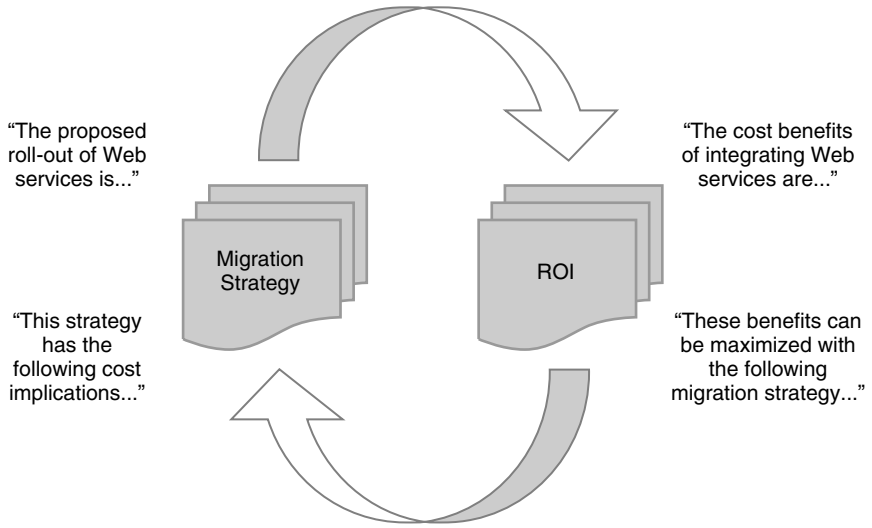

Figure 13.1
An iterative cycle between a migration plan and an ROI.

The initial SOA migration may be the most expensive part of an enterprise-wide initiative however, the scope of your ROI will likely go beyond the migration phase. Further revisions to an ROI will improve the accuracy of its predictions as they relate to subsequent phases in a long-term program.

ROIs for Web services can also be easier to justify than for other XML-based technologies. The benefits tend to be more tangible, because the interoperability enabled by the service integration layer results in immediately recognizable savings.

13.1.8 Build toward a future state

"Design a service-oriented solution to accommodate its probable migration path."

The built-in features of a Web services framework are there, whether you choose to use them or not. The foremost benefit of any service-oriented environment is the intrinsic

potential for immediate and future interoperability. You can take advantage of this potential by designing application architectures for integration, even when not immediately requiring integration.

Fostering integration requires a change in design standards, application architecture development, and the overall mindset of the project team. For example, you can facilitate local requestors as well as future remote requestors by providing both coarse- and fine-grained interfaces based on standard naming and service description conventions.

To an extent, you can consider this new approach as building integration architectures, regardless of whether they will be integrating anything immediately. Perhaps a better suited term would be "integrate-able architectures."

Chapter 14 fully explains this design approach by providing future state environments for service-oriented integrated architectures and EAI solutions.

13.2 Best practices for standardizing Web services

13.2.1 Incorporate standards

"Consider standards for Web services as standards for your infrastructure."

In the corresponding section of the previous chapter,[1] I used an analogy about obeying traffic laws in order to highlight the importance of standards when integrating XML within a development project. Let's alter that analogy to define an approach for standardizing the integration of Web services.

A city's commuting infrastructure is almost always standardized. A traffic sign on the East side generally communicates a message (stop, yield, merge) the same way on the West side. You, the driver, can go from any point A to any point B with the confidence of knowing that the rules of navigation are being expressed consistently. Take your car outside of the city boundary, though, and that might change.

As with any enterprise application development project where you have different units of developers building different parts of the system, standardizing how each part is designed is important for all the traditional reasons (robustness, maintenance, etc.). In the service-oriented world, though, the real benefit is in establishing a standardized application interface. In an enterprise, this can potentially translate into a standardized system for navigating:

1. The "Incorporate standards" best practice in Chapter 12.

- application logic
- integration architectures
- corporate data stores
- parts of the enterprise infrastructure

Back to our analogy: A different city, let's say in another country, will have a compatible driving platform (paved streets, intersections, traffic lights), but there will be new signs with new symbols, and often a different approach to driving altogether.

Navigating through non-standard environments will always slow your progress and introduce new risks. When developing services within your enterprise, you are establishing infrastructure through which developers, integrators, and perhaps even business partners may need to navigate in the years to come.

Simply adding a common platform for data exchange is not enough to ensure a quality service-oriented environment. You need more than streets and intersections to guarantee a safe and consistent driving experience.

13.2.2 Label the infrastructure

"Consider naming conventions as a means of labeling your infrastructure."

When assembling the pieces of an integration architecture you can end up with a multitude of interdependent components, each a necessary link in your solution. Since your environment will consist of a mixture of legacy and contemporary application components, you will already be faced with inconsistencies.

Contemporary integration solutions, however, are based on the concept of legacy abstraction. Introducing new architectural layers, such as those provided by Web services and adapters, allows you to hide the inconsistent characteristics of legacy environments. You, in fact, are given the opportunity to customize these new application endpoints. If you take a step back and look at the collection of potential endpoints that exist in your enterprise, you essentially are viewing infrastructure.

When working on a project, it's easy to label the components of an application arbitrarily. A name is something quickly added, so that you can move on to more important functional tasks. The benefits of naming standards are often not evident until later in the project cycle, when you actually have to start plugging things together. That's when introducing a naming convention can become especially inconvenient.

For instance, imagine yourself as an application architect in the midst of a development project. Surveying the environment in which developers are deploying Web services,

you recognize that it is riddled with inconsistent endpoint names. You convince your Project Manager that the solution should adopt a naming convention in order to increase consistency. The project team revisits the relevant component and service interfaces they created, and your PM watches in horror as this wonderful solution begins to crumble to the ground.

The names used to identify public component and service interfaces act as reference points for other components and services. When you change a name, you therefore need to change all references to it. As a result, renaming all your solution's components and services turns into a major project in itself, during which all further development is halted.

Finally, a week later, all references seem to have been updated and the solution is online again. But… it isn't working quite as well as before. The odd error, the odd communications problem — there are still some references hidden somewhere that need to be changed. So, the solution undergoes another round of testing, the remaining references are updated, and things finally return to normal.

You call up your PM (who's been away on stress leave) and let him know everything is up and running again and your components and services have new names! After a long silence, he calmly says that he will be returning soon. He hangs up, turns back to his therapist, and finishes discussing the fantasy where *you* are the PM who has to explain the cost and time overruns to the project stakeholders, and he is the annoying architect whose only concern is that things have pretty names.

Using a naming convention will not only improve the efficiency of administering your solution, it will make the migration and deployment of new integration projects much easier. Naming conventions reduce the risk of human error and the chance that a simple adjustment will lead to your solution mysteriously breaking down. As powerful and sophisticated as enterprise solutions are these days, it can still take only one broken link to bring them to a grinding halt.

One more thing

Don't bury naming conventions amidst other standards documents. I highly recommend you place them in a separate document that gets distributed to every member of your project team. This document will act as a both a navigation and development aid that can assist developers, administrators, and many others involved with a project.

13.2.3 Design against an interface (not vice versa)

In the previous section we introduced a best practice that promoted the use of a naming convention for labeling enterprise endpoints. When modeling a service-oriented

framework, we actually get to provide a complete description of these endpoints. The standardization of these descriptions, therefore, becomes very significant (see Figure 13.2).

"Consistently describing service interfaces establishes a standard endpoint model. This results in a standardized service-oriented integration architecture that can be positioned as part of enterprise infrastructure."

To ensure consistency in endpoint design, the common development process for a Web service needs to be reversed. Instead of building our application logic and then expressing this functionality through an appropriate service interface, we need to make the design of that interface our first task.

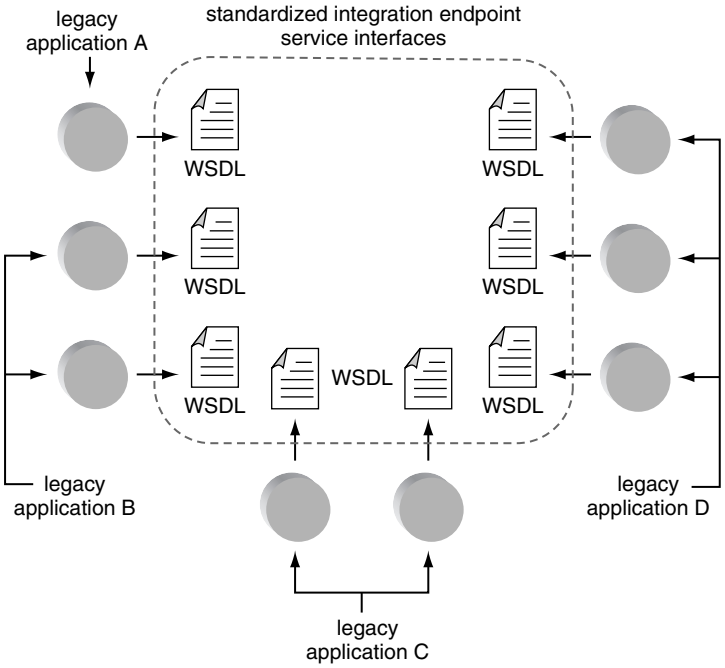

Figure 13.2
Standardized integration endpoint services establishing a service-oriented integration architecture.

This is where the fore mentioned naming conventions are incorporated with your enterprise-wide interface design standards. With a fundamental knowledge of what Web services will be encapsulating, you can create a generic, consistent interface with operation characteristics that comply to a standard model. With that in place, you can then build the back-end logic. (For a step-by-step process on how to analyze and design service interfaces using this approach, visit Chapter 6.)

The best way to ensure consistency across the interfaces within your Web services framework is to make a resource responsible for the interface design. The role of the service interface designer is explained next. Essentially this person is responsible for both the modeling of Web services as well as the design of SOAP messages.

13.2.4 Service interface designer

Designing a Web service is a separate task from its actual development. A service interface designer is responsible for ensuring that the external interface of all Web services is consistent and clearly representative of the service's intended business function. The service interface designer typically will own the WSDL document, to which developers will need to supply the implementation code. The service interface designer can also be in charge of all SOAP documents to ensure a consistent message format as well.

Typical responsibilities:

- WSDL documents
- SOAP message documents
- interface clarity
- interface extensibility
- interface standards and naming conventions

Typical prerequisites:

- a background in component design
- high proficiency in WSDL and SOAP
- a proficiency in business analysis
- a good understanding of the organization's business scope and direction

13.2.5 Categorize your services

"Use service models to classify and standardize service types."

Every Web service is unique, but many end up performing similar functions and exhibiting common characteristics, allowing them to be categorized.

This guide refers to service categories as service models. A number of service models are described throughout this book, each with a specific purpose and a list of typical characteristics. Use these as a starting point, and customize them to whatever extent necessary.

Here are some examples of how using service models can be beneficial:

- you can apply specific design standards to different service models
- the model type instantly communicates a service's overall role and position within an architecture
- models can be aligned with enterprise policies and security standards
- service models can be used to gauge the performance requirements of service-oriented applications

13.3 Best practices for designing service-oriented environments

13.3.1 Use SOAs to streamline business models

"Service-oriented designs open up new opportunities for business automation. Rethink business models to take advantage of these opportunities."

If you find yourself amidst the technology surrounding Web services, don't lose sight of one of the most significant benefits this new design platform can provide. By offering a more flexible, interoperable, and standardized model for hosting application functionality, SOAs provide an opportunity for you to rethink and improve your business processes.

For instance:

- A service-oriented architecture within your organization will increase the interoperability potential between legacy systems. This will allow you to reevaluate various business processes that rely on multiple applications or data sources.
- An array of generic business and utility services will provide a number of ways to automate new parts of your business centers.
- Services can integrate with EAI solutions to deliver new business processes that, in turn, integrate existing business processes.

To learn more about service-oriented business modeling, see Chapter 14.

13.3.2 Research the state of second-generation specifications

As more and more legacy application logic is expressed and represented within service-oriented environments, the demand increases for Web services to support a wider range of traditional business automation features.

The IT community responds to these demands by improving and sometimes replacing technical specifications. The feature set of the Web services framework continues to grow, driven both by standards organizations and major corporations, many of which

collaboratively produce specifications that address new functional areas for Web services to utilize.

"Approach the choice of each second-generation specification as a strategic decision-point."

If you are building serious service-oriented solutions, you will be working with second-generation specifications. Before you begin creating dependencies on the features offered by one of these standards, you need to ensure that:

- it is sufficiently stable
- it is supported by your current platform vendors
- there is no emerging specification poised to take its place
- support for the standard is (or will be) provided by middleware or development products you are considering

Don't make the mistake of classifying the selection of these specifications as a purely technical decision. It is a strategic design decision that will have implications on your architecture, technology platform, and design standards. (To stay current with Web services standards, visit `www.specifications.ws`.)

13.3.3 Strategically position second-generation specifications

"Design your SOA with a foreknowledge of emerging specifications."

Regardless of whether you are planning to incorporate the features offered by some of the newer second-generation Web services specifications, you should make it a point to research the feature set provided by these standards. This will allow you to identify those that may be potentially useful.

Whichever ones you classify as being significant or relevant can be positioned within your future-state enterprise architecture. This is a key step in evolving a service-oriented environment.

It is also important that you make this information publicly available to your project teams. Architects will approach the design of application logic differently with a foreknowledge of how the role a future technology may affect their application designs.

13.3.4 Understand the limitations of your platform

While traveling the roads that lead to an SOA, you are bound to run into the odd pothole or roadblock. As key industry standards continue to mature, so does the feature set required for Web services to become fully capable of representing and expressing sophisticated business logic in enterprise environments.

Until the second generation of Web services specifications is fully evolved, however, there remains a rather volatile transition period, as many middleware and development platforms compensate for the absence or immaturity of these standards by supplying solutions of their own.

Proprietary extensions (the potholes)
"Define and work within the boundaries of your development platform."

Several platforms supplement core Web services standards with proprietary extensions. Often these new features will be based on draft versions of specifications expected to become industry standards. The extent to which standards are supported, however, can vary.

When considering the use of vendor-specific extensions, make sure you understand how they are being implemented, and what dependencies they impose. Keep in mind that if you commit to using them, you may need to migrate your services away from these extensions at some point in the future.

In the meantime, however, they may very well address your immediate requirements, while allowing you to proceed with a service-oriented application design.

Exclusive proprietary extensions (the roadblocks)
"If development platform boundaries are too restrictive, reconsider the platform."

Some development platforms provide extensions to Web services at the cost of requiring that all requestors of the service be built using the same technologies. This not only defeats the purpose of designing service-oriented applications, it ties you to a platform that offers little more than traditional component-based environments.

Within a controlled environment, these features may be attractive. If open interoperability is one of your future goals, though, it's time to make a U-turn.

13.3.5 Use abstraction to protect legacy endpoints from change

The service interface layer can give you a great deal of flexibility in how you continue evolving integrated legacy applications. Since the integration layer acts as an intermediary between previously tightly bound legacy applications, a level of decoupling is achieved. This makes the Web service the only contact point for either legacy environment.

Use abstraction to improve configuration management
"Alter configuration management procedures around the abstraction benefit introduced by the service interface layer."

A side benefit to the loose coupling introduced by the service integration layer is improved configuration management of integrated legacy applications.

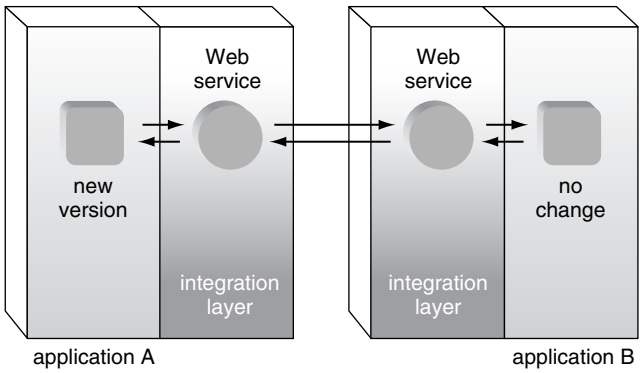

Figure 13.3
Application A is upgraded without affecting application B.

As shown in Figure 13.3, you can upgrade application A without requiring changes to application B. However, depending on the nature of the upgrade, application A's Web service may be affected. Modifying the integration layer, though, tends to be much easier than making changes to legacy logic.

Although this aspect of an SOA isn't the first benefit that comes to mind, it can have major implications on how you administer and maintain applications in your enterprise.

Use abstraction to support wholesale application changes
"Take advantage of the service integration layer by more aggressively evolving integrated legacy environments."

The level of abstraction that can be provided by service-oriented integration architectures can significantly reduce the impact of platform migrations.

Since the two applications displayed in Figure 13.4 only know of service endpoints, you can replace application A entirely without any changes required to application B. Any required modifications to application A's Web service almost always will be less disruptive than changes to application B's integration channel.

13.3.6 Build around a security model

Especially when developing second-generation Web services, incorporating a sound security model is a key part of your design process.

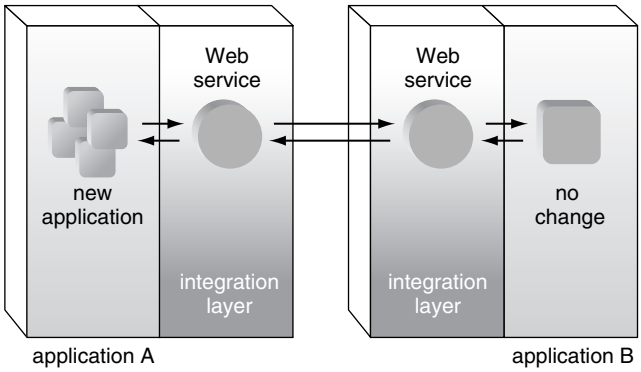

Figure 13.4
Application A is replaced without affecting application B.

Security requirements define boundaries, real boundaries

"The functional application design needs to be built upon the security model, (not the other way around)."

Putting together a design, and perhaps even building a preliminary version of your application without serious consideration for the underlying security model is a common mistake. It's like going out on stage for your performance, and then, before you can finish, getting pulled back with one of those long hooks. (It's the security requirements pulling that hook, in case you didn't get that.)

Web services security models are unique, complex, and multi-dimensional. There are many factors to consider that will result in firm boundaries that will shape and scope the remaining parts of your application design.

Define the scope of the security model

"A key part of a standardized service-oriented enterprise is a service-oriented security (SOS) model."

When we talk about a Web services security model, it can mean a number of different things. A model can represent the security rules and technologies that apply to an application. It also, however, can be standardized within the enterprise.

The enterprise SOS model displayed in Figure 13.5 establishes a standard security platform that includes policies and the technologies that enforce them. As shown in Figure 13.6, this model can be implemented within a dedicated security services layer that also becomes an application architecture standard. (For more information, see Chapter 11.)

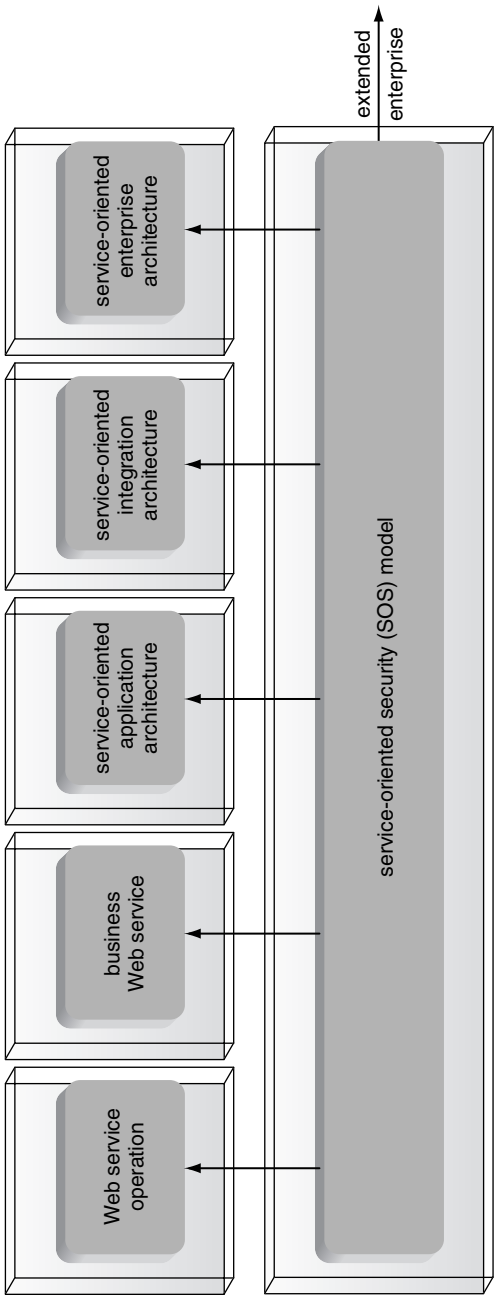

Figure 13.5
The service-oriented security (SOS) model.

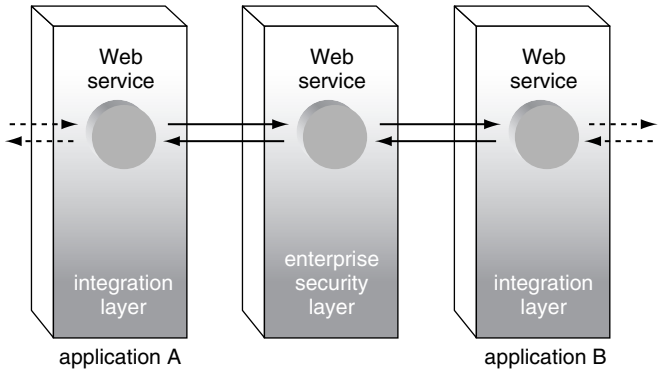

Figure 13.6
A separate security layer can implement the SOS model.

The required use of a SOS model can impose significant restrictions upon application designs. This is offset, however, by the fact that its use also alleviates application development projects from having to deal with many of the issues relating to security. Most of the decisions will have already been made; all the project team has to deliver is an application that conforms to SOS standards.

13.4 Best practices for managing service-oriented development projects

13.4.1 Organizing development resources

"Group development teams around logical business tasks."

A common mistake in development projects that incorporate a limited SOA is to have one team deliver the Web services, and the remaining team(s) develop the balance of the application. From a resourcing perspective this approach makes sense, because you have each team working with technologies that match their respective skill set. It can, however, create a disconnect between developers responsible for building parts of an application that deliver a logical unit of business functionality.

When you break an application down into its primary (or even secondary) business functions, you end up with the equivalent of a series of subprojects that collectively make up the application's feature set. Each of these subprojects will typically require the creation of a number of application components, some of which may be encapsulated within Web services. Development teams need to be grouped in accordance with these subprojects, so that the performance and functionality of individual business tasks can be streamlined.

If you have only one or two developers qualified to build Web services, then you should consider sharing them across more than one development team. As long as they are actively participating with their respective teams, they will be able to optimize the Web services to best accommodate each business task.

13.4.2 Don't underestimate training for developers

"Ensure that your developers and designers are capable of applying the right blend of business and technical intelligence to every Web service."

How much do you tip a developer? Well, that depends on how good the *service* is … OK, let's officially end my sad career in IT comedy by moving on to the importance of ensuring that your development (and design) staff has the proper skills to build and integrate Web services. If they are new to this platform, you may not want to just hand them a book and ask them to "go figure it out."

Here's why:

- In many cases, Web services affect the application's business model. The execution of an automated business task will differ within an application, depending on how Web services are utilized. Developers may not be aware of the relationship between the service they are developing and its associated business task(s). This becomes an especially critical issue when building controller and process services.
- Recurring requirements for Web services are that they be generic, openly accessible, and relatively independent. To achieve these characteristics, the functionality within a service needs to be carefully defined. This requires knowledge of the business functions an application needs to deliver now, as well as the level of interoperability it will have to provide in the future. Generally, developers are only focused on immediate project requirements.
- When teams of developers are involved with an application development project, a set of new standards will likely be required to ensure that Web services are implemented consistently. Developers can participate in the creation of these standards; typically, though, standards need to be established by those already proficient with the technologies.
- It is easy to create Web services with contemporary development tools. This can give a developer a false sense of confidence. It is, in fact, easy to create Web services, but only bad ones.

13.5 Best practices for implementing Web services

13.5.1 Use a private service registry

Once Web services establish themselves as a common part of your enterprise, they will begin to evolve, requiring interface upgrades and spawning new generations of services. Pretty soon, it will be difficult to keep track of the many service interfaces, especially since some will always be in a state of transition. (To learn more about private registries and UDDI, read through the tutorial in Chapter 4.)

Centralize service descriptions in a service repository

"Incorporate a private service registry to centralize published service descriptions into one accessible resource."

A private service registry can house the collective descriptions of all your Web services. It acts as the central repository for current service interface information to which anyone interested will go to discover and learn about an enterprise's service framework.

Its use has immediate benefits, including:

- efficient access to service interfaces (no time wasted searching)
- preserving the integrity of published service interfaces ("published" is a state represented by the repository)
- encouraging the discovery of generic and reusable services

Make the use of a service registry mandatory

"Require the use of a private service registry and keep it current. Otherwise it won't be useful."

If people lose confidence in a service registry, it can quickly become the least popular part of your IT environment. If you locate a service interface in a local UDDI registry, and you're not sure it is the latest version, you won't be inclined to use it. Instead, you'll probably phone around until you find the original service developer, from whom you'll get the most recent WSDL file.

If, however, the use of this registry is a requirement that is strictly adhered to, it will become a core part of your administrative infrastructure, supporting development projects as a resource centre for published Web service endpoints.

Assign a resource to maintain the registry

"To make enterprise service registries a functional part of an organization, assign a Service Library Manager."

Private service registries need to provide a high level of availability and dependability. Not only will the registry serve individuals who manually search it for various service details, it may also need to facilitate dynamic discovery. At that point, it could become a critical resource.

The best way to ensure that a registry is kept current and available is to assign ownership of these responsibilities. Maintaining a service registry is a unique job, in that it involves an uncommon combination of skills. Below we provide a description of this role, called the *Service Library Manager*.

Such a resource becomes especially important if your organization opens its registry to external business partners. In that case, the Service Library Manager also needs to manage the authentication and authorization of users from outside of the organization.

Responsible for maintaining the service library and the local UDDI registry, this individual may need to be included in official application design reviews so that proposed service designs can be evaluated and compared against existing and other planned services.

The Service Library Manager will also own the organization's utility services. Any changes required to these services will need to be approved by the library manager, and implemented in such a way that they are sufficiently generic for future use, and do not break existing interfaces already in use by service requestors.

Responsible for:

- service library
- publishing of service descriptions
- maintenance and design of utility services
- review of application designs incorporating services

Typical prerequisites:

- UDDI or a services broker product
- a background in component design
- a good understanding of the organization's business scope and direction

NOTE
In smaller IT environments, you can consider combining the roles of Service Library Manager and XML Data Custodian.

13.5.2 Prepare for administration

An often overlooked aspect of projects implementing service-oriented applications are the subsequent maintenance tasks required to keep these environments going.

"Be prepared for the costs and complexities in administering a service-oriented enterprise."

Increased interoperability results in a higher amount of dependencies between application environments, namely their Web service endpoints. With a high level of integration comes the responsibility of keeping your Web services running smoothly, regardless of what's thrown at them. High usage volumes, error conditions, and other environmental variables need to be anticipated.

Entire product suites are available to maintain Web services, although many are platform specific. If you are deploying Web services without such an environment, administration can eventually become an overly burdensome task. You may want to prevent this from happening by investigating some of the application hosting environments offered by service-oriented EAI solutions.

Either way, administration costs need to be properly represented in project budgets. Otherwise, the support infrastructure required by service-oriented architectures will not be sufficient. This, in turn, can jeopardize the success of the application and those that integrate with it.

Here we introduce the *Service Administrator* role, a resource responsible for the maintenance and monitoring of these hosting environments.

In an environment where many Web services are deployed and utilized, an administration system needs to be in place in order to ensure a reliable runtime hosting environment.

Service administrators need to be proficient in the use of maintenance and monitoring tools. They will be the ones who need to respond to production issues relating to the availability and performance of Web services. This role is similar to that of a webmaster for a Web site. The administrator is required to keep track of usage statistics and look out for (and preemptively avoid) performance bottlenecks.

This position is especially relevant in organizations offering Web services that can be accessed externally. In order to effectively handle unpredictable usage volumes, the administrator must be able to respond quickly when performance trends start heading south.

In EAI environments, this role is often assumed by the same person managing the integration brokers. Typical responsibilities include:

- assessing the need for specific administration tools and servers
- evaluating and perhaps integrating these products
- monitoring the use of individual Web services
- analyzing usage statistics and identifying trends and patterns
- assessing the impact of the Web service use on underlying legacy applications or components
- identifying and tracking dependencies (service requestors) of deployed Web services
- managing version control over Web services
- being involved with version control of legacy applications represented by Web services

Typical prerequisites:

- proficiency in Web services administration products
- fair knowledge of WSDL and SOAP
- good understanding of service deployment techniques and related security settings

13.5.3 Monitor and respond to changes in the service hosting environments

"Be responsive to increased infrastructure requirements."

When Web services represent the endpoints to existing or new integration channels, they can easily become the busiest parts of an integrated environment. This can tax the underlying areas of the infrastructure supporting those services. To avoid performance bottlenecks, it's a good idea to survey your existing infrastructure and identify any part that may need to be upgraded.

In preparing for any new application, there will be obvious areas where upgrades will be required. New servers, more memory, and strategically located routers are all typical requirements needed to support incoming application hosting environments.

To achieve an optimized environment designed to host Web services, though, you frequently need to see them in action first. The communications framework introduced by Web services brings with it new protocols, different types of runtime processing, and an overall shift in where this processing can physically occur.

It is difficult to predict exactly where and to what extent processing requirements will change. Therefore, it is often best not to *fully* upgrade your infrastructure until you

have a good idea of what the actual performance requirements will be. One way of determining this prior to making your Web services available for production use is to perform a series of stress and volume tests.

Another approach is to measure and respond to performance requirements by carefully monitoring production usage. For instance:

1. Phase in the production release of your application.
2. Closely monitor performance and study usage patterns.
3. Respond quickly with hardware upgrades where required.

13.5.4 Test for the unknown

"…but they said that if we build a service-oriented architecture, we'd be freed from the problems involved with connecting disparate technology platforms…" Sure, but you still need to build your Web services framework using a vendor-specific development platform, along with a vendor-manufactured SOAP server.

Products used to establish an environment for service-oriented data exchange may provide various levels of support for various (mostly second-generation) Web services specifications. This can lead to some discrepancies in areas such as WSDL document interpretation and SOAP message header processing.

"To guarantee the level of interoperability promised by Web services, incorporate a multi-client test phase. This precaution is especially important when working with second-generation specifications."

The simplest way to address this issue is to increase the amount of testing each newly created Web service will be subjected to. Your test strategy should require that services be tested with a range of clients representative of potential service requestors.

For instance, you may have one project team creating an application using J2EE, while the other is basing theirs on .NET. Even though neither team needs their application to integrate with the other's, their respective testing phases should still include client requestors based on both J2EE and .NET.

This issue is comparable to the age-old presentation-related problems Web page designers faced when having to accommodate Netscape and Microsoft browsers. There were a number of discrepancies in how HTML was rendered and in how client-side script was processed. Conditional logic often had to be used in order to output different page content.

When first developing a Web service, the effort to fix processing discrepancies is generally negligible. However, if these problems remain undetected until after services have been deployed, then you've got yourself a redevelopment project on hand.

About the Author

Thomas Erl is an independent consultant with XMLTC Consulting in Vancouver, Canada. His previous book, *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services*, became the top-selling book of 2004 in both Web Services and SOA categories. This guide addresses numerous integration issues and provides strategies and best practices for transitioning toward SOA.

Thomas is a member of OASIS and is active in related research efforts, such as the XML & Web Services Integration Framework (XWIF). He is a speaker and instructor for private and public events and conferences, and has published numerous papers, including articles for the *Web Services Journal*, *WLDJ*, and *Application Development Trends*.

For more information, visit `http://www.thomaserl.com/technology/`.

About SOA Systems

SOA Systems Inc. is a consulting firm actively involved in the research and development of service-oriented architecture, service-orientation, XML, and Web services standards and technology. Through its research and enterprise solution projects SOA Systems has developed a recognized methodology for integrating and realizing service-oriented concepts, technology, and architecture.

For more information, visit `www.soasystems.com`.

One of the consulting services provided by SOA Systems is comprehensive SOA transition planning and the objective assessment of vendor technology products.

For more information, visit `www.soaplanning.com`.

The content in this book is the basis for a series of SOA seminars and workshops developed and offered by SOA Systems.

For more information, visit `www.soatraining.com`.