Disadvantages and Pitfalls of Web Services

In the previous hour, we looked at what Web services are and how they can be used to solve problems. However, Web services are not a magic bullet solution for every issue; they do have limitations. In this hour, we’ll discuss those issues.

In this hour, you will learn

- What pitfalls to expect with Web services
- What performance issues affect Web services
- How a lack of standards affects Web services
- How the newness of Web service technology can be a problem
- What staffing challenges exist when going with a Web services solution
Pitfalls of Web Services

No technology is perfect. Although Web services do a great job at solving certain problems, they bring along issues of their own. Some of these pitfalls are inherent to the technological foundations upon which Web services are based, and others are based on the specifications themselves. It is important to know what these issues are so that you can plan for and build around them.

Some of the biggest issues are

- **Availability**—Everyone who uses the Internet knows that no site is 100% available. It follows that Web services, which use the same infrastructure as Web sites, will not be 100% available either. Even if the server is up and running, your ISP might not be, or the ISP hosting the other side of the transaction might not be either. If you need 100% up time, do something else. Because of this situation, it is often necessary to build mechanisms that will retry the transaction or fail gracefully when this occurs. Some of the newer protocols supported by Web services (JMS, for instance) will handle this automatically, but the majority built on HTTP will not.

- **Matching Requirements**—Any time you create a general service that will handle a variety of customers, you will run into specialized requirements. Some customers might require the one extra little feature that nobody else needs. Web services are envisioned as a “one size fits many customers” technology. If your business can’t fit into that model, you should consider other solutions.

- **Immutable Interfaces**—If you invest in creating a Web service for your customers, you have to avoid changing any of the methods that you provide and the parameters that your customers expect. You can create new methods and add them to the service, but if you change existing ones, your customers’ programs will break. This is easy to do until you find that one of your existing methods is returning wrong answers and can’t be repaired because the approach is fundamentally flawed. Early releases of Java contained methods for calculating date and time differences that could not be made to work correctly. They had no choice but to junk the first set of routines and write new ones. This caused programs to quit working, but there was no choice because the programs were getting wrong answers from the old routines. Although this sort of problem occurs in all systems, it is especially true in Web services. You might not know who is using your service, and as a result you have no way to inform those users of the change. In most other systems, because of the tight coupling, there usually is more verbal or written coordination between producers of a service and consumers of it.
Guaranteed Execution—The whole idea behind having a computer program instead of doing a job by hand is that the program can run unattended. HTTP is not a reliable protocol in that it doesn’t guarantee delivery or a response. If you need this kind of guarantee, either write the code to retry requests yourself or arrange to send your requests through an intermediary who will perform these retries for you. Again, newer versions of the specification allow for using protocols such as JMS to resolve this issue, but the majority of services out there still utilize HTTP, which does not.

As you can see, with such limitations, Web services might not be right for your needs. This is just the beginning of the list, however.

Performance Issues

The only performance guarantee that you have concerning the Internet is that performance will vary wildly from one time to the next. Performance-critical systems are not suited to becoming Web services.

Web services rely on HTTP—the same communication protocol upon which Web pages are requested and delivered. HTTP was designed to enable one server to handle hundreds of requests quickly by not maintaining a long-term stateful connection between the clients and the server. Instead, HTTP initiates a fresh connection with the server and maintains it only as long as it needs to transfer the data. Once complete, the connection is terminated and the server is then freed up to process a request from someone else. The server typically will not maintain any sort of concept of state to keep track of from one user’s request to the next, although this can be achieved through the use of session tracking or cookies. This tends to make HTTP very transactional in nature.

Although the HTTP communication transaction enables the server side to handle many clients, it also means that a lot of time is wasted creating and terminating connections for clients that need to perform a large number of calls between the client and the server. Other technologies—such as DCOM, CORBA, and RMI—don’t have this problem because they maintain the connection throughout the application lifecycle.

See Hour 4, “Comparing Web services to Other Technologies,” for more information on the differences between Web services and these other technologies.
The other performance consideration that must be taken into account with Web services is the conversion to and from XML during the communication process. All communication takes place as XML messages encoded in a special format (a SOAP envelope). This conversion takes time. Depending on how complex your data is and how much of it there is, this time could be a serious penalty. For instance, if you try to send a binary image as a method parameter, that data must be encoded into a format that can be represented in XML. This certainly is not as fast as transferring the image across the wire in its original format.

Although this sort of time penalty occurs with other architectures such as RMI and CORBA, it can be especially bad for Web services–based systems because data is transferred as XML text—with a large amount of extra information required in the SOAP envelope. This overhead isn’t too bad when we consider a simple string or number to be encoded. Binary data, such as images, tend to be much larger though and take a lot longer to convert. XML tends to take more bytes to encode data than the equivalent binary representation.

Finally, when writing a Web service, you typically have a choice of many different languages and tools with which to build your solution. It is important that you pick ones that will scale to handle the expected loads adequately.

**Lack of Standards**

Another set of issues is associated with Web services that centers around incomplete or nonfinalized standards. Although there are ways to fix most of these problems, they are unique to each vendor’s implementation. If the guiding principle of Web services is to create an open standardized interchange system for remote program execution, the utilization of any vendor-specific solutions should be avoided. The current Web services specifications and standards are lacking in the following areas:

- **Security and Privacy**—Anything sent over the Internet can be viewed by others. At present, the only approved option for sending sensitive information to Web services is to use a Secure Socket Layer (SSL) program. SSL over HTTP is sometimes called HTTPS. This technology is wonderful in that it does a good job of safeguarding information such as credit card numbers, and it is easy to set up compared to other encryption techniques. The disadvantage is that SSL is slower than unencrypted HTTP transactions and is therefore unsuited to large data transfers. In addition, SSL is much more secure than most sales statistics need. Such a transaction shouldn’t ordinarily be sent as clear text though. SSL encrypts the entire data stream, not just the sensitive parts. Let’s consider the case
in which the data being transferred contains an item’s inventory tracking number, its description, and the credit card number used to pay for it. In most cases, online thieves would not care about the inventory number or the item description, but they would want to steal the credit card number. With SSL, it’s an all or nothing solution. This situation is improving, however, because the W3C has several draft proposals that hopefully will resolve this problem.

- **Authentication**—Authentication answers the question, “Who is contacting me?” At present, the standards ignore this issue and delegate it to the Web services Container. This causes proprietary solutions to be created and breaks down the promise of portability. Until a universal single sign-on solution is agreed upon, this problem will remain.

- **Nonrepudiation**—Nonrepudiation means that you have rock-solid proof that a communication took place. This is also delegated to the Web services Container.

- **Transaction**—Many activities that would be well suited to Web services are very complex. They involve dozens of smaller actions that must either all complete or all be rolled back. Closing on a new house is a good example. The last thing you need is for your financing to fall through but half the transaction (such as the escrow setup and courthouse registration of the lien) to complete anyway.

- **Billing and Contracts**—In most cases, you’ll want to charge for the use of your service. As a result, a way for pricing contracts to be negotiated and maintained needs to be created. The current specifications provide for service discovery, but do not contain a mechanism for handling pricing for the service or performing automatic billing for the use of the service. This issue becomes even harder if different customers want different billing rates. This also ties into the provisioning problem because without some sort of a contract in place, secure provisioning becomes unrealistic. Contracts also need to determine service level agreements. (Who would pay for a service unless a contract stated that it would be available 99% of the time?)

Because the specifications don’t have an agreed upon mechanism for handling these issues, many vendors providing Web services tools have built their own solutions. This can lead to problems when moving from one vendor’s tools to another or getting two different vendor’s tools to talk. As a result of this billing problem, most Web services posted to public UDDI registries are still free to use. On the other hand, companies providing Web services for their business partners typically do not post their service on public UDDI registries and can therefore control access and billing for use of the service through traditional manual billing methods.
• **Provisioning**—This is the adding of valid user accounts to a system to allow users to access the service. Currently there is no agreed upon standard way to do this. The service provider and consumer need a mechanism to exchange provisioning information, and the service provider must know who it trusts as a source of that user information.

• **Scalability**—Because it is possible to expose existing component systems such as Enterprise Java Beans as Web services, it should be possible to leverage the load-balancing and other scalability mechanisms that already exist. But are there unforeseen stumbling blocks along this path? Does there need to be a new kind of “Web service” application server? Some vendors have come up with mechanisms to provide for load balancing, but no standard for it currently exists.

• **Testing**—Because of the new structure of Web services and the decoupling of the clients from the server, testing of Web service–based solutions can be challenging. Short of adopting a “try it and see if it worked” mentality, there really are few ways to test a Web service system. This is especially true for situations in which the client applications are written by one party and the server is written by another who charges for its use. How would one test a potentially half-built client against a service that you don’t control? Even more important is what happens if that client does something to break the service?

Most, if not all, of these problems will be addressed in subsequent versions of the standards. Until then, we have no choice but to implement proprietary solutions with plans to revisit them when the technology matures.

**Newness of Web Service Technology**

Most technologies are over hyped in their first two years because it typically takes that long for the platforms to mature to the point of fixing initial problems and the toolsets to become powerful enough to provide good solutions. During this time, you usually hear a lot of people promoting all the virtues of the new technology, but few of the issues have actually been discovered. Those issues can only be found with time and work. At the time of this publication, Web services are still in this phase of maturity.

Conversely, the same technology will usually be under recognized after the four- to five-year mark. This is unfortunate because at this point the industry would’ve learned how to work with the technology effectively, staff would’ve come up to speed, and the tools would be mature. But, because of the initial over-hype situation and the failure to live up to initial expectations, the technology in question can quite often be orphaned by the very same people who were its biggest initial champions or, worse yet, be replaced with
the new solution of the week. That’s not to say that the technology is totally abandoned. Rather, it is often the case that the technology has just moved beyond the “fad” phase and into the realm of being useful (sometimes extremely so), but no longer “sexy.”

Figure 3.1 illustrates the typical technology adoption and maturity lifecycle.

**Disadvantages and Pitfalls of Web Services**

Because Web services are a relatively young technology, the standards and specifications are evolving rapidly. Applications built on one version of the specification can be quickly outdated. The specifications are also still evolving in some areas of Web services. As a result, some vendors have chosen to provide their own proprietary solutions rather than wait for a standard to emerge.

Also, because the Web service area is relatively new, a large number of vendors are fighting for market share. Although this stimulates innovation, it also means that developers who buy into the wrong vendor might be orphaned a year or two down the road and be forced to switch to another vendor’s solution. This can be a big headache and cost a good deal of time and money. Even though Web services are built on standards, each vendor has its own way of implementing those standards, which could require major retooling of your system if you have to change toolsets. All markets go through a shakeout period in which the most popular (and usually powerful) products remain and the majority of the others fade away. This has not happened yet with the Web services tools market.
Staffing Issues

Staffing can be one of the most frustrating parts of implementing any new technology. Because Web services are a fairly new solution, it can be somewhat difficult to find qualified and experienced staff to implement a workable solution.

The staffing issue can be twofold. Managers and architects are not up to speed on the technology and, as such, might not trust “that new-fangled gizmo with all the hype.” On the other side of the equation is the development staff who aren’t familiar enough with all the ins and outs of the technology to know how to properly build solutions based on it. Hopefully, by reading this book, a lot of the problems can be resolved for both of these groups.

As with any new technology, to find qualified, experienced personnel, it might be necessary to pay a premium price. Although the recent downturn in the IT marketplace has made the available talent pool deeper, truly good developers can still be hard to find. It might be necessary to bring in an experienced contractor to help jumpstart your project. The other alternative is to train from within your organization. Both solutions have their good and bad points and must be weighed with your organization’s needs in mind.

Summary

In this hour, we’ve discussed what issues currently affect Web services. We’ve looked at how some of the underlying technical foundations could be an issue for Web services.

We then discussed some of the performance problems associated with this technology. We saw how the fact that Web services rely on nonpersistent connections found in HTTP can be a performance concern and how the conversion of data into and out of XML can slow down processing.

Next, we examined some issues that exist because of present limitations of the specifications and standards. We saw how various vendors have come up with work-around solutions that are nonportable and thus should be avoided.

Finally, we discussed some of the issues regarding the young age of Web services technologies and how this relatively young solution can cause staffing problems.

By keeping these points in mind, a proper choice can be made between a Web services–based solution and other ones.
Q&A

Q If Web services have all these problems, should I use them for production-level systems?
A Although Web services do have some issues to be resolved, many of the alternatives do as well. Each technology has strengths and weaknesses. Now that you know what they are for Web services, you can make an informed decision on whether to use them for your project or not. If you’re building a system to be used by many other people, Web services are probably your best bet.

Q If I go with Web services, how can I minimize the impact of the changes to the standards?
A The easiest way to reduce impact is to limit yourself to using only standards-based options from the toolkit that you choose. That way, when the standards bodies agree on a solution for, let’s say, security, you can easily add it to your code once your tool vendor updates its tools. Otherwise, you might be locked into a proprietary solution that might not be supported at a later date.

Workshop

This section is designed to help you anticipate possible questions, review what you’ve learned, and begin learning how to put your knowledge into practice.

Quiz

1. What issues are inherent to Web services because of their underlying technologies?
2. What affects the performance of Web services?
3. What sort of staffing issues can one expect?
4. What are some of the problems found because of gaps in the specifications?

Quiz Answers

1. Immutable interfaces, unguaranteed availability and execution, and a one-size-fits-all requirement.
2. The underlying HTTP not maintaining connections long term, the conversion to XML, and the need to choose efficient server-side programming languages to build your service with.
3. As with any new technology, good people will be expensive and hard to find.
Activity

1. You should take a look at the w3c.org Web site to see what emerging standards are coming down the road that are meant to address the problems we discussed in this hour. Industry and the development community have been working hard to fix some of these problems, and many of them might be resolved by the time you read this.