Advantages of Web Services

There are many advantages to using the Web services architecture over any other. In fact, some Web services applications would be expensive or impossible to duplicate using any other technology. One of the challenges for technical leaders is to discover opportunities in which Web services can be used to solve today's problems. In addition, we need to be looking for Web services solutions to problems that are ever present, but never solved. These areas include the cost of doing business, in effect, the cost of software development, and the time that it takes to react to new market opportunities. In this hour, we will look at a fairly extensive list of the advantages that Web services provide over more traditional approaches. In addition, we will cover some concrete examples of what is happening and what can happen in the future using Web services.

In this hour, you will learn about the advantages of using Web services for

- Integrating legacy systems
- Lowering operational costs

- Lowering software development costs
- Getting systems done faster
- Interfacing with customers
- Integrating with external business partners
- Generating new revenue
- Supporting new business models

Legacy Systems

New Term Many of the early adopters of Web services technology have concentrated their efforts at interconnecting *legacy* (established, reliable, operational) systems to each other. Their reasons for doing this are

- Working on internal systems entails less technical risk than working with outside entities. For this reason, companies have decided to keep the trial-and-error phase of new technology adoption out of the spotlight.
- Many firms allow their internal developers to gain experience on legacy systems before allowing them to interface with their customers' systems.
- To make a Web service useful, a client must exist. Internal projects give the developers control of both the client and the service.
- Older solutions are problematic. Many of these projects replace older programs that were written using other, often inferior approaches. Many of these approaches, like flat file transfers, are fragile. Other development projects, written in CORBA or DCOM, tend to be expensive to maintain and enhance. For this reason, they are obvious targets for replacement.

Web services wrap nicely around legacy systems, regardless of the language that they were written in. Many developers roll their eyes when they are asked to work on 25 year old COBOL or PL/1 systems. They mistake old for bad. From the standpoint of management, an older software system that has been running the company's shipping department for more than 20 years is worth its weight in gold, regardless of how uncool the technology is. Customers pay for products that get delivered properly, regardless of how stylish the program that managed the delivery process is.

The thought of writing, testing, and implementing a new system to replace one that already works well is a tough sell to a company's management. Why would they pay money to replace a solid performer with one that is certain to be buggy for the first few months, if not years?

Web services can be written in such a way that they require little or no change to the legacy code base. They can be written to interact with the older code similar to the way that a *graphical user interface (GUI)* would. The end result is a state-of-the-art distributed system that retains all the equity that a company has built up in its legacy code base.

Lower Operational Costs

Any company that is going to succeed in the long term is constantly trying to lower its costs of doing business. These cost improvement programs are focused on both internal and customer-oriented systems. Web services provides opportunities to save costs in these areas:

- **Cheaper than a LAN**—It is hard to imagine any way of interconnecting computers that is cheaper than using the Internet. Essentially, every computer that can access the Internet could serve as either a Web service, a Web services client, or both. A LAN requires that you interconnect all computers with some sort of cable. The Internet allows you to connect via each computer's existing service provider.
- **Low-cost Electronic Data Interchange (EDI)**—You might find some cost savings if you set up electronic data exchanges with your best customers and vendors. EDI has pioneered this effort, and Web services are poised to expand it in every direction.
- **Remote Status Reports**—The fact that any two computers can participate in a Web services transaction means that you can get a status report from any organization in the world at any time of the day or night. The potential savings from the elimination of paper-based reporting systems is huge.
- **Remote System Management**—There is virtually no limit to the amount of remote management that can be done using this technology. Most of the savings occurs when a technician can troubleshoot a problem from his own desk without incurring any travel overhead or expense.
- **Dynamic Routing of Service People**—Web services are being expanded onto *personal digital assistants (PDAs)* such as the Palm and Handspring models. This allows people who still have to travel from site to site to use just-in-time dispatching. Instead of planning the whole repair day in the morning, a truck can receive the next assignment when that worker is actually ready to work on the job. This can result in large increases in efficiency and less customer downtime.

Lower Software Development Cost

For the past 20 years, the Holy Grail of software development has been the concept of code reuse. The idea of code reuse is as simple as it is intuitive. Instead of reinventing the proverbial wheel a thousand times, you invent it once and use it a thousand times.

The complete promise of code reuse has remained an elusive target. One problem has always been the packaging of the code. For code to be reused, it has to be created in a way that makes it easy for a new program to find and use it. If the code existed, but you didn't know about it, you could not receive any benefit from it. If you knew about it, but it was written in Java, a Visual Basic developer probably couldn't use it. The fact that Web services can interoperate regardless of the language that each of them is written in is a tremendous development in the code reuse world.

For some time, it has been possible to create a Web service–style system by interconnecting programs written to the DCOM and CORBA standards. Practically speaking, the torment associated with doing that was so great that only the most important problems were solved in this way. To make matters worse, most company firewalls block these types of transactions, making it impossible to implement them.

With Web services, there will no longer be a need to translate DCOM objects into CORBA and vice versa. For the first time in history, all major hardware and software vendors are in agreement that Web services should exist and interoperate seamlessly. The reason for this ecumenical feeling is that Web services technology can be implemented on any computer platform using any development tools. This keeps every vendor in the game and reduces the need for professional gripers to tell us why the other company's approach is weak. Now they tell us that Web services is the future, but the competition's toolset is weak.

This ecumenical bliss is underscored by the fact that a very fair-minded organization, the *World Wide Web Consortium (W3C)*, is the official keeper of the Web services quasi-standards that they call recommendations. Most vendors have people who sit on W3C committees and steer the new proposals in the direction that they want to see them go. The end result is that everyone remains in synch with the Web services community as a whole.

The high level of vendor commitment to this technology lowers the cost of using it. Not only does competition drive down tool prices, but it also encourages innovation by pressuring the tool vendors to keep thinking outside the box.

Faster System Development

Better tools mean faster development, which in turn leads to lower development costs. Another way development costs are lowered is by allowing a wide variety of programming skill sets to play in the game. If you have an army of COBOL programmers, you can hire one Web services guru to set up the infrastructure, and then have your staff write everything else in COBOL. If you use PERL, keep your development in PERL. Web services tools are on the market that interact with almost any language. Even if the tool support is weak for your favorite language, you can create the WSDL and SOAP messages manually and communicate that way.

Many vendors provide rapid development tools that go beyond what is available for creating user interfaces. Web services are based on the creation of both a service and a formal description of the service called the *Web Services Description Language (WSDL)* document. This document contains enough information about the service to allow a client to be generated. Instead of starting with a blank sheet of paper, the programmer of a Web service client starts with a generated client that can successfully access a Web service. He can then enhance that client to interact with the other systems on the client side. The time and learning curve savings from using this approach is significant.

Many modern development tools can also generate Web service code from functional descriptions. Although these services are more limited than hand development solutions could be, they offer the advantage of speed. Even for complex requirements, the generated part of the Web service code can get the project started. From there, the programmer can hand code the unique parts that are beyond the capabilities of the tool.

Better Interfaces with Customers

Every business guru from Tony Robbins to Steven Covey will tell you that the way to succeed in business is to get close to your customers. Vendors who succeed in integrating their systems such as orders, shipping, and billing are nearly impossible for their competitors to dislodge because they are woven into the business processes of those customers.

One example of this would be an automated reordering system. If you provided an automated way of interfacing to the point-of-sale system at a major food retailer, you could know, by sales data, when to send more disposable diapers to a certain store. The competition would have trouble breaking into that account because they can provide product but not convenience. Convenience can translate into big savings for a customer because it lowers his cost of doing business. Figure 2.1 shows this business interaction.

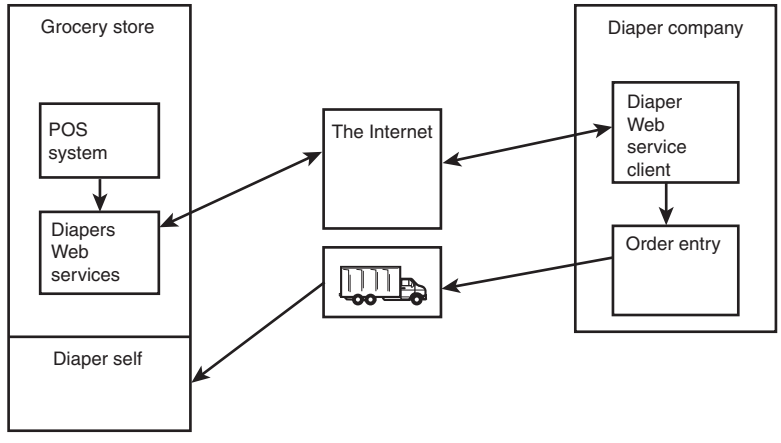

FIGURE 2.1
Web services allow tighter integration between vendors and their customers.

Another way to endear yourself to your customers is to allow them to peek inside your systems to check on order status. For example, when I ordered a camera online and had it shipped by UPS, I would get online every day just to see where the package was. I watched with interest as it moved from warehouse to warehouse and finally onto the truck for delivery to my home.

Many companies have systems that could be opened up without much risk. Contractors could provide status reports in this way to put their customers at ease. Car companies could track the progress of custom vehicle orders so that the customer could enjoy the process of buying that special car. Baseball teams could publish their internal statistics for the rabid fans to follow.

Some Web services can provide savings to your company, while allowing your customers better access. For example, allowing customers to manage their own account data frees up your internal resources and increases the accuracy of the data. When a customer's shipping address changes, he could access and change that data via a Web service.

Better Integration with External Business Partners

Modern corporations are anything but monolithic entities. Most companies are built on the foundation of business partners—other companies that have traditionally been classified as suppliers. Forward-thinking firms have concluded that the best way to lower costs

is not to beat your suppliers down to get them to cut their margins, but to help those suppliers lower their costs. It is better to find suppliers who will allow a level of integration that will benefit both companies.

One example of this is outsourced human resources. Writing paychecks to engineers is essentially the same thing as writing paychecks to the dancers in a Broadway play, even though their daily work activity is very different. For this outsourcing to be truly less expensive, however, the HR vendors' systems must be able to interact with their clients' systems well.

Web services are ideal for this type of interaction. You can interconnect two different systems without making significant alterations to either one. This can enable an outsourced company to act as a Web services client in order to collect payroll and personnel data. Figure 2.2 shows these businesses interacting.

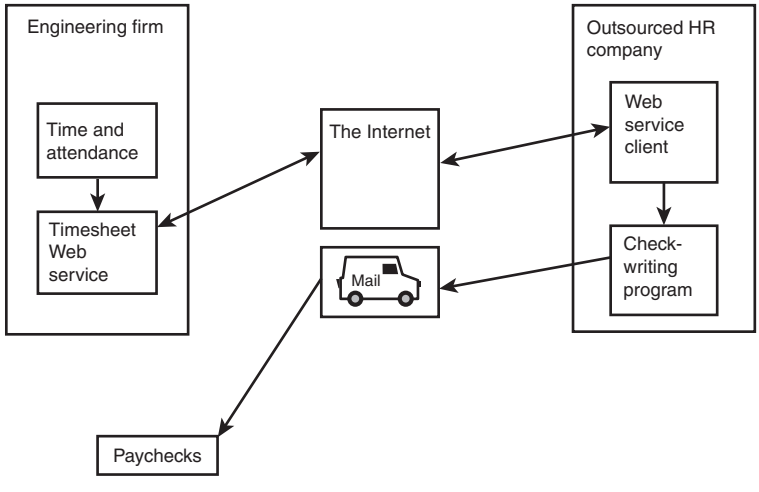

FIGURE 2.2
Web services allow tighter integration between business partners.

New Revenue Opportunities

NEW TERM This level of cooperation between computer systems will open up new opportunities for virtual enterprises. A *virtual enterprise* is one that is made up of several smaller enterprises. For example, if a hotel in Orlando booked your room, scheduled your flight, reserved your rental car, and sold you tickets to the theme parks, you would be doing business with a virtual company. No single company owns airplanes, cars, hotels, and theme parks. Each company owns one of these assets and offers the others through partnership agreements. In fact, you could start a company that owns none of these assets. Your sole reason for existing is to provide one-stop shopping for these types

of packages. If you have a strong advertising strategy and the proper business agreements, you could sell merchandise that you have never actually owned. Figure 2.3 shows these businesses interacting:

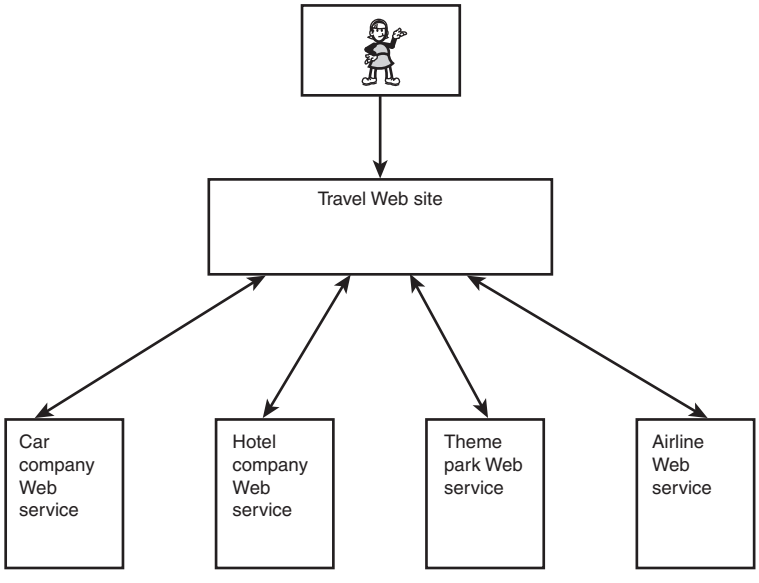

FIGURE 2.3
Web services allow new businesses to be assembled from smaller businesses.

The key to making this type of virtual business work is the low-cost interconnection between computer systems. If the cost of interconnection is too high, your enterprise will have to pass on too much cost to the customer. If your customer feels that you are too expensive, he will bypass your site and individually purchase services from each vendor.

Fortunately, Web services make it possible to create interconnections for far less cost than traditional approaches. In addition, the time needed to program a new service connection is far shorter using the Web services approach.

Another new potential source of revenue can be found in charging directly for the use of your business processes. This area is so new that we have to go into creative mode to even think of examples. One example that comes to mind would be a cost-estimating system. It is critically important for construction companies to accurately estimate the cost of building a certain building. Normally, these companies write internal systems that they plug the details in to. The result is a report showing a cost breakdown for the project.

You could imagine a group of programmers breaking off from one construction company and starting their own software company. This company could hire the best estimators in the industry and produce a superior cost-estimating system. Every construction company

would run its own numbers internally, but it could pay our new company to run them too. The combination of the two estimates would serve to either increase confidence in the original number or to flag a potential problem if the two numbers don't agree. Figure 2.4 shows these businesses interacting.

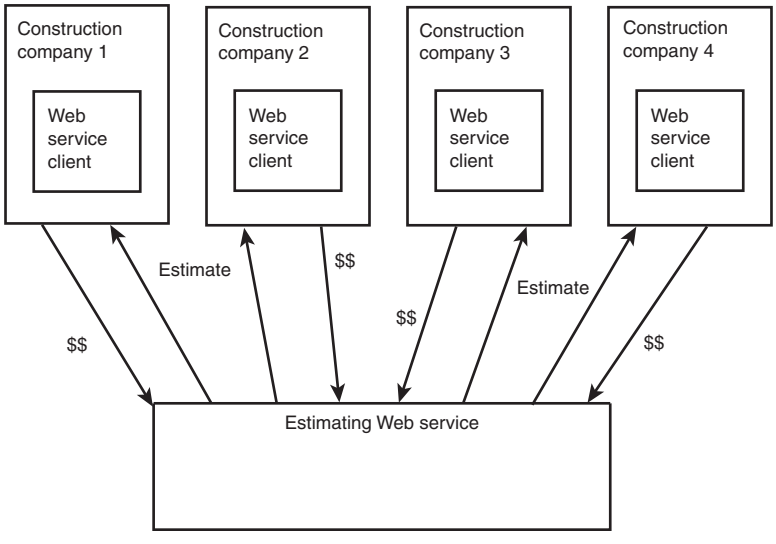

FIGURE 2.4
Web services allow new types of businesses to exist.

Another place where a Web service could be a revenue generator is in the field of product design. Engineering analysis software is used in the design of nearly all the physical goods we use. The automobile industry is an especially heavy user of this kind of software. Analysis is performed on the plastic in a dashboard, the beams in a chassis, the vibration in a steering column, and so on. Normally, a virtual solid model is created in the computer and submitted to one or more of these analysis programs for evaluation. The result of the evaluation might cause a design to be changed to better meet its requirements.

The software packages that perform this work are very expensive and normally only used by companies that have a lot of this work to perform. Smaller companies must either do this analysis by hand or simply do without. If the vendors of these packages could turn them into Web services, they could then charge based on the usage and not by the license. This could allow that vendor to sign up new customers easily because the initial expense would be very low. In addition, the fees paid to the vendor would come out of the customer's normal expense budget and wouldn't require a capital expenditure.

Completely New Business Models

Another opportunity for a new revenue stream would be in selling empty seats, mechanics bays, and so on. All service businesses have perishable assets. An empty bay in an automobile repair shop or an empty table in a restaurant is perishable. If it is not used this hour, it is wasted and cannot be retrieved and sold in the future. Finding a way to sell these empty seats, even at a steep discount, would improve the profitability of these businesses. A large tire and brake chain could create a Web service that prices the same procedures differently, depending on the demand. For example, a brake job performed at 10:00 a.m. on a Saturday would be charged full price, but at 8:00 a.m. on Tuesday, it might be half price.

A business with a fleet of vehicles could schedule them for maintenance based on the demand. They could write a Web service client that interacted with the brake stores whenever a maintenance procedure was required. The Web service could provide a dynamic price based on the demand at stores across town. The client would compare these prices to the cost of driving to each location and would choose the one that provided the best value.

In addition, consumers could use a Web-based version of this software. An applet might serve as the client running in a browser. The customer would enter his parameters such as the procedure needed, urgency, location of vehicle, and so on. The Web service would then provide a list of shops and prices. The customer could then choose the one that he considers to be the best value and could drive across town, if necessary, to patronize that shop.

Another type of customer-oriented Web service would be a towing system. A Web service could be written that accepts requests for a tow. Normally, this request would be created by a phone call to an operator at 1-800-tow-junk. A stranded motorist would place a high priority request at full price. If he is really desperate, he might even add a tip in the offer.

All the tow services that monitor this Web service have clients installed on Palm-sized devices. Whenever a tow is needed, it will appear on the Palm, along with the price offered. The first available tow truck will commit to do the job and accept it by clicking on the Palm device. After contacting the stranded motorist, he would get underway.

Not all tows represent emergencies, though. Car dealers, junk yards, and repair shops use towing services a lot. They are not in as big a hurry as a stranded motorist, so they would offer less for a tow. The tow truck would move these cars during slow times or even on a night shift when prices are best. Whenever an emergency comes in, he would go to it, make more money that hour, and return to the car dealer when it gets slow again.

Almost any business with excess perishable capacity is a good candidate for this type of empty-seat selling. The dynamic nature of the pricing can lead to better utilization of resources and higher profits. In low-tech businesses such as towing, a third party would be the likely Web service provider. The provider might even provide the Palm-sized devices to the trucks in exchange for a 10% fee on each tow scheduled through them.

Summary

In this hour, you learned about the advantages that might be available to you with Web services. We covered how Web services can be used to integrate legacy systems, lower operational costs, lower software development costs, and get systems done faster.

In addition, you learned how Web services can help you to interface with your customers better as well as with your business partners. Finally, you saw how Web services can help you generate new revenue and support new business models.

Q&A

Q How can Web services make legacy systems integration easier?

A Web services do this by providing a set of protocols for data exchange that can be used to move data from one legacy system to another.

Q How can Web services improve customer service?

A You can use Web services to provide your customers with better information about the status of orders, levels of inventory, and so on.

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. Why is development less expensive with Web services?
2. How can Web services be written in any language and still work?
3. What makes Web services easier to interface to a legacy system?
4. Name one new application that Web services make possible.

Quiz Answers

1. Because Web services are described precisely by a WDSL, some code on both the server and the client can be generated.
2. SOAP is written at a high enough level of abstraction that any language can implement it.
3. The fact that both the client and the server can be written as wrappers around legacy systems makes it easy to interface them.
4. Auctioning systems in which vendors of all types can sell their unused capacity at a discount holds great potential.

Activities

1. List the different categories of savings and revenue generation that are covered in this hour.
2. Choose the five types of savings or revenue that are most likely to be of interest to your organization.
3. Create at least one potential system in each of these categories. Share your ideas with others in the organization.

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 though 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

NEW TERM

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.

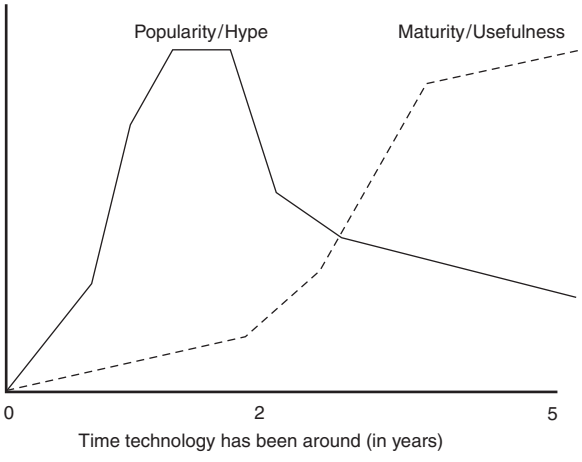

FIGURE 3.1
The typical technology hype/usefulness lifecycle.

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.
4. Billing, contract negotiation, authentication, transaction management, and security, among others.

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.