Deploying and Managing Web Services

Distrust and caution are the parents of security.
—Benjamin Franklin

Deployment- and management-related concerns, in practice, will circumscribe all the real-world operational issues currently facing those trying to use or offer production-grade Web services for enterprise-level applications. This, as the saying goes, is when the rubber really hits the asphalt when it comes to Web services, or when one quite literally has to start reconciling the hype with the hardware. The real problem here, to immediately cut to the chase, is that the dynamics surrounding Web services have been dramatically undermined by circumstances outside the scope of Web-services technology and industry.

It is fair to say that the world—socially, economically, and technologically—has changed quite a bit and unexpectedly between mid-2000, when Web services were first being postulated, and now. First, there was the crash and burn of the dot.coms that tarnished the credibility, even if it was subliminally, of e-business. Then there was 9/11!

While the repercussions of 9/11 were still reverberating, there was the anthrax scare. The threat of terrorism, in all forms, including that of cyber-terrorism, started to impact all forms of decision making, whether corporate or personal. A siege mentality set in. In big cities such as New York people wait for another shoe to drop. The relentless attacks by the hacker community continue to add insult to injury. Pantophobia, a fear of everything, is rife. There are people who have stopped opening regular mail and others who have stopped using e-mail.
It is against this pervading climate of uncertainty and fear, where trust is in short supply, that one now tries to promote Web services—a brand new, unproven, iconoclastic technology that advocates information sharing and collaborative processing over the Web. Suffice to say that the original, somewhat utopian model of Web services being dynamically located (using UDDI and WSDL) and then automatically exploited—in a free-wheeling, “take it for a test spin,” plug-and-play manner—is now passé, especially when it comes to enterprise-class applications. There is an evocative analogy here pertaining to the free love culture of the mid-1960s and what happened to all of that with the advent of the deadly STDs.

The concept of glibly sourcing software functionality over the Internet from previously unknown service providers is no longer viable at the enterprise level—despite all protestations from the cognoscenti that it is indeed possible to have safe and secure scenarios. While the standards-based technology offered by Web services still has tremendous potential and appeal, for the time being, the purveyors of Web services will have to be carefully vetted by using the traditional due-diligence methods involving reference checks, credit ratings (e.g., Dun & Bradstreet rating), installed base, financial reports, and testimonials. UDDI—which already includes the Publisher-Assertions structure, which can be used to enumerate certifications, memberships, relationships and so on—could still help in providing initial (and moreover programmatic) first-cut validation. There can even be WSDL-centric service contracts. But most enterprises will want some level of face-to-face interaction with potential Web-service providers—even if it is via videoconferencing.

In addition to wanting traditional validation of providers, there is also an understandable preference by most corporations, at present, for acquiring or licensing required Web services from their providers and then deploying them on in-house servers, behind the corporate firewall. This obviously eliminates the uncertainty and risk of relying on a Web service being run by a third party at a remote location. It all boils down to control. In today’s climate of uncertainty and distrust, enterprises want to have as much control as possible of their destinies—and that means controlling as much as possible of their IT systems, resources, and dependencies. This does not preclude outsourcing, but outsourcing done with control, contracts, and commitments. Web-service use, rather than being Internet oriented, has, for the time being at least, become intranet/extranet focused.

These concerns about trust, security, and thus control have in essence changed the fundamental Web-services paradigm in the eyes of corporate IT professionals. There is now an added level of complexity and expense.
Bernard Borges, an IBM Web-services architect, was quoted in the September 8, 2003, issue of *InfoWorld* as saying, “People in general thought Web services were simple and cheap, and it turns out they are complex and not so cheap. I think the whole notion of a Web service was using them as a simple integration tool. [But] it painted a rather rosy and low-tech picture compared with proprietary EAI products.” And, as *InfoWorld* notes, somewhat tongue in cheek, IBM, as a Web-services pioneer, was one of the parties instrumental in propagating the “Web services are simple” picture. But life in general was simpler prior to 9/11.

The bottom line here is that the Web services-in-practice paradigm has changed between what was first thought when the XML Web-services

<table>
<thead>
<tr>
<th>Initial Expectations</th>
<th>Reality, mid-2003</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic, on-the-fly location and invocation</td>
<td>Dynamic location (using UDDI) but careful evaluation of providers’ credentials prior to trying out the Web service</td>
</tr>
<tr>
<td>Invoke and use across the Internet</td>
<td>Usage restricted to intranet/extranet behind firewalls</td>
</tr>
<tr>
<td>Mechanism to facilitate exploitation of external, third-party software functionality</td>
<td>Methodology for in-house software development and software functionality sharing between selected partners</td>
</tr>
<tr>
<td>Focus on functionality and results rather than the platform</td>
<td>Need to know about the platform, because the platform could be an issue</td>
</tr>
<tr>
<td>Distributed, trust-based control</td>
<td>Tight, in-house control</td>
</tr>
<tr>
<td>Usage-based payment model</td>
<td>Acquire or license Web service so that it can be deployed in-house</td>
</tr>
<tr>
<td>Distributed management across domains</td>
<td>Centralized, in-house management</td>
</tr>
<tr>
<td>Simplicity is the byword</td>
<td>Not as simple as hoped—security concerns alone make it much more complex and convoluted</td>
</tr>
<tr>
<td>Inexpensive means of obtaining software functionality</td>
<td>Not as inexpensive as initially hoped but still attractive</td>
</tr>
<tr>
<td>Service provider can update Web-service software provided I/O model and promised results stay the same</td>
<td>Any and all changes to the Web-services software or infrastructure need to be carefully vetted, regression checked, and monitored</td>
</tr>
<tr>
<td>Standard Web-centric security measures based on authentication, digital certificates, digital signatures, SSL, firewalls, etc., will suffice</td>
<td>Security concerns will continue to dampen all out exploitation of what Web services can offer</td>
</tr>
<tr>
<td>New distributed, decentralized software model</td>
<td>XML-based extension to intra-enterprise object-oriented programming</td>
</tr>
<tr>
<td>Standards based and simple—no need for expensive consultants</td>
<td>Better hire some consultants to determine optimum strategy, options, and action plan</td>
</tr>
<tr>
<td>Simplify and expedite application integration</td>
<td>Jury is still out</td>
</tr>
</tbody>
</table>

Table 7.1 *Web Services Usage Model from an Enterprise Usage Perspective*
While the original Web-services model envisaged Web services being freely invoked across the Internet, the current corporate preference is the intranet/extranet model.
vision was postulated in mid-2000 to what is thought now as enterprises wrestle with how best to exploit Web services in a safe and secure manner. The intelligentsia faction of the Web-services community continues to try to address some of these deployment-related issues with new standards. To this end there is the nascent Web Services Manageability 1.0 specification, as well as a raft of security-related specifications such as Web Services Policy Attachments, Web Services Policy Assertions, Web Services Trust Language, Web Services Secure Conversation Language, Web Services Security Policy Language, and Web Services Policy Framework, as shown in Table 1.1. These emerging standards, however, have a long way to go and need to be bolstered with other identity and trust management standards before enterprises will be willing to again explore the dynamic, free-wheeling model first espoused for Web services.

Table 7.1 summarizes how the Web-services usage model has changed since its inception, while Figure 7.1 depicts this change graphically.

### 7.1 Web services: The risk assessment

This candid assessment of how Web services are currently being viewed and used by enterprises should not be construed in any way as portending the demise of Web services or as an indictment of the enabling technologies associated with Web services. Much of what has transpired has to do with a climate of uncertainty and fear that came to be independent of Web services. At the time of writing, I am unaware of any security breaches that have been attributed to Web services. This could, however, be attributed to the prudence shown to date when it comes to the use of Web services.

At this juncture it is worth reiterating that when everything is said and done, the basic Web-services model, which is based entirely on the exchange of XML documents, does protect users from viruses and worm-like security threats. Even the potential danger from the optional SOAP attachment capability can be negated by ensuring that the application invoking Web services does not automatically open any attachments without explicit authorization from a designated systems operator. This type of control, which can be enforced via the calling application, should not be underestimated or ignored. The calling application, which would typically be written in-house or developed specifically for an enterprise, does have total control over all of the interactions that take place with invoked Web services.
When a Web service sends information back to the application, in the form of one or more documents conveyed using SOAP, it is still up to the application as to how it processes those documents. Enterprises could insist that developers design applications like the Java applet “sandbox” model—that is, information sent back from a Web service cannot in any way cause the application to perform any malicious or unexpected actions. In other words, the application developers have to do their utmost to prevent scenarios similar to the unpredictable but inevitably dangerous results in the event of buffer overflow security exposures that are currently plaguing Microsoft’s Windows software. One of the key things to remember in this context is that the calling application can play a pivotal role in ensuring that Web services can’t compromise enterprise resources.

In practice, the real dangers with Web services relate to bait-and-switch or Trojan horse scenarios that set out to deceive the calling application. This is where a Web service is not what it claims to be—particularly when it is dealing with sensitive information (e.g., credit card details, medical records, financial data). For example, the Web service, in addition to performing what it is supposed to do and thus satisfying the requirements of the calling application, may also be performing nefarious actions with the information it receives—unbeknownst to the users of that application. A Web service being provided at a remote site, by an unknown third party, unfortunately, has plenty of scope and opportunity to engage in such unauthorized and unethical behavior. For example, an unscrupulous Web-service provider is likely to be able to intercept incoming or outgoing data at multiple points within the processing stack and make unauthorized copies of the data.

There are multiple scenarios that pertain to such rogue Web services. The key scenarios that one needs to be on the guard against include:

1. A Web service that is corrupt from inception despite masquerading as being above board and respectable—that is, one that has been developed from scratch to scam unsuspecting users. This can be thought of as a Trojan horse setup.

2. A legitimate, bona fide Web service that gets swapped out, with or without the knowledge and concurrence of its provider, by a rogue Web service that emulates all the functions of the original but in addition performs illegitimate operations behind the scenes. This is a bait-and-switch scenario. A variation of this would be for the calls to the genuine Web service to be intercepted and diverted to a rogue Web service.
3. The platform on which a bona fide Web service is being run on is compromised, unbeknownst to the service provider, and thus enables a hacker to intercept or alter the data that is being processed by the Web service. This is the concern people currently have with Windows platforms, given the various documented security vulnerabilities that could enable a hacker to surreptitiously take control of a Windows machine. This is a platform vulnerability–related issue, which, at present, is focused mainly on Windows platforms—though other platforms could also, in theory, be compromised by a relentless hacker.

4. A bona fide Web service offered by a respected provider that has, nonetheless, been compromised, covertly, by one or more members of the development team. This is another variation of a Trojan horse scenario.

5. A rogue or compromised Web service could flood the calling application with spurious or continually duplicated data, thus creating a denial-of-service (or replay attack in the case when duplicated data is being used) scenario at the application end.

Note that having service contracts or service-level contracts per se will not safeguard an enterprise from such rogue or compromised Web services that intend to deceive the calling application. There are tools being developed that will enable a Web-service evaluator to determine the various actions that will be performed by a Web service, depending on the types of XML input parameters it receives. These types of tools will supplement and simplify the source-level review of Web services to determine their integrity.

WebInspect 3.0 from SPI Dynamics (www.spidynamics.org) is one such tool. It sets out to assist in assessing the fidelity of a Web service by exploring all of the XML input parameters it claims to accept and then performing various parameter manipulation on each XML field, looking for vulnerabilities within the service itself. There are multiple advantages to using these types of automated execution path mapping tools. If well conceived, as WebInspect 3.0 indeed appears to be, such tools can be thorough and systematic—thus ensuring that all potential paths have been adequately explored. They could also bypass the need for source-code-level review—especially if, for intellectual property reasons (if nothing else), the Web-service owner is unwilling to share source code with any potential evaluator, particularly if there is no financial commitment already in place.

But here, however, is the rub. Even a line-by-line source-code review of a Web service is not a safeguard against rogue or compromised Web services.
That is the problem. There could be a bait-and-switch or data interception outside the Web service. Moreover, the platform could be infiltrated at a later date. The bottom line is that safeguarding against rogue Web services is a convoluted challenge that has to be carefully pursued, in the case of remotely-invoked Web services, on a case-by-case basis.

In addition to these dangerous rogue Web-services scenarios, there is also the ever-present concern that information exchanged with a remote Web service, over the Web, can be intercepted or diverted during transmission. Encryption such as 128-bit SSL, obviously is the first-cut protection against such unauthorized access, though one can always argue, quite cogently, that given enough time and processing power this type of encrypted data can be eventually deciphered. However, the real key here is obviating any meaningful ROI of trying to intercept and deencrypt the data in question. The bottom line here has to do with making realistic determinations as to the types of data that are to be transmitted. If the data are extremely sensitive or valuable, then maybe they should not be sent across the Web. Figure 7.2 takes the standard remote Web-services model introduced in Figure 1.2 and annotates it to show how and where this model may be compromised.

**Figure 7.2** Areas where security in a Web-service configuration may be compromised.
7.1 Web services: The risk assessment

7.1.1 Growth scenarios for external Web services

All of this said, one can come up with the following examples of how and when enterprises may still be able to safely and gainfully use the originally postulated remote Web-services model in the context of new applications:

- If the remote Web service is being used to process relatively innocuous data—for example, sending a zip code to receive mapping, weather, or traffic data; sending a flight number to ascertain the estimated time of arrival for that flight; sending unqualified numeric strings for currency conversion or sales tax computation.

- Web service has been thoroughly vetted, at the source-code level, and is being hosted by a trusted provider that offers audited, collaborative change control (i.e., Web service cannot be updated without documented permission) as well as continuous remote monitoring to ensure that no changes have been made to the Web-service software or infrastructure.

- Web service dealing with scientific data (e.g., material strength coefficients, chemical properties, etc.) offered by academic or nonprofit organizations.

- Web services that deal with potentially sensitive but nonetheless overtly public-domain information (e.g., property tax records from local authorities, voter registration records, court rulings, etc.).

- Required Web service, possibly dealing with sensitive information, is only available on a remote invocation basis from a specific organization, albeit with elaborate access control, authentication, and encryption mechanisms in place (e.g., a tax-related service from a federal or state tax authority, employment eligibility–related data from an immigration authority, homeland security–related updates from the pertinent authority).

There can be exceptions to this list. For example, management of a company in a highly cut-throat, competitive market sector (e.g., car rental, hotel reservation, stock brokerage) may argue persuasively that even apparently innocuous data related to its business operation could be used by the competition to gain a market edge. In such situations even zip codes, flight numbers, or unadorned numerical strings can be construed as constituting vital corporate data that has to be carefully safeguarded.

To be fair, in today’s highly competitive global market, with the data mining tools that are readily available, information such as zip codes, flight numbers, currency conversions, and sales tax calculations could be
dissected, analyzed, and reanalyzed to determine what or how a company is doing. Even the number of calls to a specific Web service (e.g., shipping rate calculation, sales tax calculation) over a given period of time could be used to determine the health and vitality of a company. Therefore, depending on the specifics, there can always be valid reasons as to why an enterprise may determine that using an external Web service is just not the prudent thing to do—despite all assurances of Web-service integrity, end-to-end encryption, and the innocuousness of the data.

At this juncture it should also be noted that enterprise-level applications, though a key and potentially highly lucrative market, are by no means the only market for Web services. There is still a huge market for Web services in the nonenterprise application arena. One could even argue that from a Web-services perspective, this personal-use market, with some overlap with the small office, home office (SOHO) market, could be as large and important as the enterprise market.

Remind yourself of all the ingenious, incisive, and magnanimous freeware and shareware software that is readily available on the Web and outside of it. The creators of all such software, as well as their successors, are prime candidates to create a whole new genre of freeware and shareware software based on the original, mix-and-match, plug-and-play Web-services paradigm. This market, which is not as susceptible to or concerned about information pilferage, is ideally suited to truly exploit the platform-independent, dynamic invocation promise of XML Web services. The bottom line here is that the scope and applicability of Web services are not restricted purely to the enterprise market. Even if enterprises persevere in using Web services predominantly on an intranet/extranet basis, remotely invoked Web services, running on third-party platforms, can still flourish within the nonenterprise sector.

Noncorporate portals (e.g., public portals or specialized content portals), are another growth area for externally invoked Web services. As mentioned in Section 1.2, Web services complement portals. This was why IBM felt compelled to institute the concept of Web Services for Remote Portals (WSRP)—where the term remote in this instance alludes to remotely invoked Web services, in this instance, Web services with integrated GUIs. Remotely invoked Web services from diverse third parties are the ideal, cost-effective way for a portal provider to offer customized, value-added functionality.

To begin with, most portals will want to use Web services for much of the de rigueur but nonproprietary utility functions users expect from a public
portal: weather, sunrise/sunset times, phases of the moon, top news stories, horoscopes, and so on. Web services also provide portal providers with an enormously powerful, flexible, and standards-based mechanism for integrating external functionality. Consequently, it is safe to say that remotely invoked Web services will certainly flourish within the continually expanding portal market.

Another nascent but potentially burgeoning market for Web services is the voice and smart phone arena. There is already considerable momentum around VoiceXML (i.e., Voice eXtensible Markup Language) fostered through the VoiceXML Forum (voicexml.org) an industry organization founded by industry titans AT&T, IBM, Lucent, and Motorola. VoiceXML is a standard essential to making Internet content and information accessible via voice and phone. Given that it is a bona fide XML derivative, it can be easily adapted for use with existing Web-services technology. In addition to voice-based applications, there is an insatiable demand for value-added functionality for the increasingly powerful and versatile cell phones—some of which already support Java virtual machines. Web services are well poised to help developers exploit this new and exciting market for sophisticated software, interactive voice applications, and smart phones.

The final point that should be noted is that the software vendor community, with IBM in the lead, recognizes that there is a huge pent-up market for security and management tools that would make remotely invoked Web services more palatable for enterprise use. IBM’s Web-service gateway, which was introduced in the spring of 2002, along with SPI Dynamics WebInspect 3.0 and other software vulnerability assessing software such as Sanctum’s AppScan (www.sanctuminc.com), are precursors of things to come.

The express goal of IBM’s proxy capability for Web services is to enable the externalization of Web services beyond the boundaries of an enterprise network. This gateway, which serves as a wonderful prototype for subsequent offerings, can act as a bidirectional proxy for services outside an enterprise firewall as well as for those within. It is, in effect, an XML application firewall for Web services; it ensures, per the now-accepted firewall model, that there are no unguarded end-to-end connections into an enterprise network that can be exploited by hackers. This gateway does not safeguard users from rogue or compromised Web services. However, if you do plan to use an adequately vetted and thus trusted external Web service, doing so through this gateway will ensure that all the Web services–related interactions are controlled, deterministic, and protected (as far as possible) against abuse by hackers.
When an enterprise application wishes to use external Web services, this gateway, using the WSDL definitions for the required service, will import its functionality into the gateway—albeit in the form of an outbound remote call to the original Web service. However, the calls from the enterprise application to the Web service now get terminated at the gateway. There is no end-to-end connection between the calling application and the original Web service. Instead, calls to that external Web service are intercepted and serviced by the gateway, as shown in Figure 7.3.

The gateway in essence acts as an application-level Web-service firewall. It performs a corollary function when applications outside the enterprise firewall wish to invoke a Web service that is deployed inside the firewall.

### 7.2 Security considerations for Web services

There are three very distinct sides to Web-services security if one looks at this pivotal issue from a holistic, all-inclusive standpoint. Web services by design only work per a strict client/server model (albeit within a larger n-tier architecture)—where, depending on one's perspective, the calling application can be considered to be the server while the Web service being invoked becomes the client, or vice versa, where the calling application is deemed to be the client of a Web service. In reality it does not matter which is the server and which is the client, as long as one recognizes that there will always be two independent software components at play—with
7.2 Security considerations for Web services

Thus, from a security standpoint, to guarantee total coverage one has to look at both software components and the connection between them. It means that it can be counterproductive to focus on just one side of the model only to discover down the road that the other side is relatively exposed. Figure 7.4 highlights all of the perimeters that have to be safeguarded when it comes to Web services.

Enterprises already have policies and technology in place to protect their networks (e.g., firewalls, SSLs, virus scans, intrusion detection alarms) and applications (e.g., authentication). Web services–related security measures do not in any way displace any of the existing security policies, procedures, and technologies in use by enterprises. That is a given. Instead, they require that existing technology (e.g., authentication, digital signatures) be extended, as well as the addition of a whole new Web services–specific layer of safeguards (e.g., XML application gateways and software validators such as WebInspect), if one wants to deal with remotely invoked services.

Given that enterprises have now relied on computerized, mission-critical applications for over four decades, the access control and user validation mechanisms needed to deter unauthorized access are well known. Much of these mechanisms now fall into what is referred to as identity management or the “technology of trust.” Stringent authentication, whether it be with user ID/password schemes, digital certificates, or two-factor token-based
authentication such as RSA Security’s SecurID scheme, is the basis for identity management. New applications that intend to invoke Web services should continue to rely on proven identity management and access control mechanisms to ensure that they can be accessed only by authorized users. This is standard application-level security and is independent of whether an application has any Web-services affiliations or not.

The introduction of Web services, however, mandates that application-specific security has to be extended to encompass the new application Web-services interface—if one intends to pursue the remote Web-service model over the Web. At a minimum, there are at least seven different types of security measures that may need to be enforced at each individual application Web-services interface—though the need for some of them (e.g., nonrepudiation) will depend on the nature of the data and transactions involved. These security measures are as follows:

1. Stringent service provider/service requester authentication between the application and each Web service it invokes
2. Access control, possibly at both ends, to determine the functions that may be requested—per invocation, based on the authentication instance
3. Digital signatures to ensure the validity of contents
4. Nonrepudiation to preclude either side from disowning a transaction once it has been executed
5. XML application firewall, such as IBM’s Web Services Gateway, to decouple the end-to-end communications connection at the enterprise network boundary
6. Proven data encryption end to end—most likely with the industry standard SSL or its successor TLS
7. Denial-of-service/replay attack detection and diversion mechanisms—which typically come with powerful traffic pattern sampling, analyzing, profiling, and reporting tools that will continually monitor the network interface to spot any unusual trends

None of these security measures can or is meant to identify rogue or compromised Web services—though bidirectional authentication could make it harder to surreptitiously hijack transactions being sent to one Web service and divert them to a rogue service. Consequently, it is still of paramount importance to unequivocally establish the credentials, trustworthiness, and technical competence of a Web-service provider by using
traditional, time-tested methods before one ever gets around to worrying about implementing any of these measures. One course of action is to consider only Web services from well-known, well-funded entities that can reasonably be expected to run a tight ship with all necessary safeguards to prevent Web services from being hijacked, hacked into, or otherwise compromised.

Once a service provider and the necessary Web services from that provider have been adequately validated and chosen, authentication, particularly in the form of digital certificates, can be used to make sure that calls to a specific Web service are in fact being responded to by the expected service. This authentication will typically be done in both directions: the application authenticating the Web service and the Web service authenticating the application. Depending on the sensitivity of the transactions involved, one could consider multilevel authentication—for example, an initial digital certificate–based authentication followed by a two-factor authentication using RSA’s widely used SecurID. (For the record, it is worth noting that at present this RSA SecurID technology is being used by over 10 million Web users around the world for secure application access, VPN utilization, Web server protection, and corporate portal partitioning.)

Proven authentication technology, including digital certificates, existed well before the advent of Web services. The success of corporate portals, e-commerce, and b2b applications was contingent on the availability of relatively impregnable authentication—always keeping in mind that, as with any type of security, total 100 percent infallibility is next to impossible to guarantee. But it is safe to say that this technology is mature, widely used, and well understood by those whose job it is to safeguard corporate assets.

Over the last 2 years there have been considerable effort, led by IBM, Microsoft, BEA, and security stalwarts VeriSign and RSA, to come up with new XML-based standards to more tightly align this prerequisite technology with Web services–related methodology—in particular, SOAP.

The key specifications on this front, spearheaded by the overarching WS-Security initiative, are listed in Table 1.1. This, however, as one should be able to appreciate, is a very dynamic and fluid arena at present, with new specifications being added and many updates to the existing ones. For the latest status on these specifications visit www.xmlweb.org, a new portal committed to promulgating the latest status on Web services–related technology.

The good news is that existing authentication, digital signature, SSL/TLS, and nonrepudiation solutions can be easily used very effectively, at the application Web-service interface ahead and independent of the XML-
7.2 Security considerations for Web services

centric security standards. The goal of the new standards is to facilitate tighter integration. But waiting for these standards and products that comply to them should not be used as an excuse for not exploiting Web services. The exact security measures, from the previous list, that should be implemented on a particular application–Web-service interface will depend on a number of factors, including:

- The nature of the data and the types of transactions involved
- The type of network connection being used (e.g., a VPN connection will already have significant authentication and encryption capabilities built in)
- Whether the Web service is public (i.e., available to a variety of subscribers) or whether it is controlled (i.e., only available to a select set of tightly controlled and licensed users)
- The functionality offered by the Web service, given that access control would be superfluous with a single-function Web service
- Reliable safeguards available at the ISP level—particularly in regard to virus scanning and denial-of-service attack interception
- Safeguards, contractually agreed upon, offered by the service providers (e.g., guarantees against repudiation of transactions)

Suffice to say that determining the appropriate cocktail of security measures for any given application Web-service interface will be a complex and multifaceted exercise that will have to be undertaken on a per-service basis. In addition, there will invariably be pressures to consider optional refinements, such as adopting single sign-on schemes to minimize the amount of individual authentication required each time a service is invoked or when invoking different services offered by the same provider. Single sign-on is always a double-edged sword. Though it simplifies and expedites repeated access to the same provider or the same service, at the same time it lowers your authentication barrier, albeit by a smidge, increasing the chances of it being breached by a determined perpetrator.

The bottom line here is that there are plenty of well-proven security measures to adequately safeguard most enterprise-level Web-services scenarios—but you have to be diligent in working out exactly what you need to implement per interface, and you need to be continually vigilant to make sure that all the security measures are functioning as they are supposed to. In many cases it might be best to work with established enterprise security experts, such as RSA, VeriSign, or IBM/Tivoli, to determine an appropriate security architecture for Web services. Plenty of consulting firms, such as
Accenture, EDS, BearingPoint, Deloitte, and individual consultants, will trip over each other trying to help you formulate Web services–related security strategies for a fee.

7.2.1 Digital certificates and PKI

Digital certificates (DCs) are now routinely and transparently used on the Web for e-commerce, online banking, electronic trading, and corporate portal access applications—so much so that the “properties” option of popular Web browsers now includes a button that will show the DC that is in force at any given time. Figure 7.5 shows the DC details available from

![Figure 7.5 Details of digital certificate usage shown by Microsoft’s Internet Explorer—in this case, for a secure, authenticated connection with the Charles Schwab portal.](image-url)
Microsoft's Internet Explorer—in this instance for a secure connection with electronic trading powerhouse Charles Schwab.

Digital certificates are an electronic credential (in the form of a small file) issued by a trustworthy organization such as a large company (e.g., IBM, Microsoft) or a security-specific entity such as VeriSign. Figure 7.6 shows a
page from VeriSign's portal providing an introduction to one its public-key
infrastructure-based digital certificate programs. A DC vouches for an indi-
vidual's, a business's, or an application's identity and authority to conduct
secure transactions over the Web. DCs are in essence the Internet equivalent
of a travel passport. They are a universally accepted means of establishing
one's identity and thus gaining entry to a protected resource. DCs are meant
to replace traditional user IDs and passwords, which are not as secure or
trustworthy—hence, their applicability in the Web-services arena.

The issuing organization of a DC is called a Certificate Authority (CA).
Consequently, VeriSign is an example of an existing and well-known CA.
Refer to www.pki-page.org for an extensive list of CAs around the world as
well as products that can be used to set up a CA. A CA can issue a DC to
individual users, a server, or an application—where application in this con-
text will also include a Web service, which is but a mini-application. If your
company currently does not have any DCs, you may want to contact a CA
and begin the process of acquiring a valid DC for use in Web services as
well as other scenarios (e.g., corporate portals).

The acceptance of a DC as a valid credential is totally contingent on the
recipient's trust in the CA. Therefore, just having a DC is not enough. You
need to have confidence in the CA that issued it, especially since there are
no uniform requirements, at least at present, as to what information a CA
checks before issuing a DC. Some public Internet CAs may only ask the
user for name and e-mail address in order to issue a DC. This is totally
inadequate.

Fortunately, DCs today are closely tied in with PKI. CAs that support
the PKI Exchange (PKIX) standards typically insist that the requester's
identity information is further validated by a Registration Authority (RA)
 prior to a DC being issued. RAs are an integral part of a public-key infra-
structure. There are proven PKI-specific products, such as RSA's Keon and
IBM's Tivoli Trust Authority, that can be used by companies so that they
can act as RAs. These systems allow companies to set up centralized or
distributed electronic enrollment centers that validate user requests for DCs
on behalf of the CA. Figure 7.7 shows a section from RSA's portal introduc-
ing the PKI capabilities offered by Keon.

The key thing to bear in mind here is that a DC, in the end, is only as
good as the CA that issued it. So all DCs are not necessarily equal. If you
are going to use DCs as a base-level security measure to establish the ongo-
ing bona fides of a previously selected service provider, then you will have to
check which CAs meet your security criteria. For example, you may not
want to accept DCs from unknown CAs. Invariably it would be best to for-
mally agree on which DCs you will accept from a provider up front when evaluating and validating the provider.

DCs leverage public-key cryptography. Public-key cryptography relies on the use of a pair of associated keys for encryption and decryption—known as public and private keys. There is a mathematical relationship between these two keys that guarantees that anything encrypted with one key can be cleanly decrypted by using the other key. Which key is made public and which is kept private is totally arbitrary. With public-key cryptography, one can freely distribute the public key without in any way compromising or revealing the nature of the private key. The public key can even be stored in a public directory or a Web page. The private key, true to its name, must, however, be kept guarded and secret.

The public and private keys only work in tandem. That, so to speak, is the key. In other words, a document encrypted with the public key can only be decrypted with the private key. The document cannot be decrypted with the openly available public key, since this would nullify the whole point of encryption. The same applies the other way around. A document encrypted using the private key can only be decrypted using the public key.
makes sense because the private key is kept secret and should not be available for use by other parties. Figure 7.8 provides an overview of how public-key cryptography works, and Table 7.2 summarizes which keys are used for which operations. A DC will include the certificate holder’s public key.

Digital certificates are defined by the X.509 standard. An X.509 certificate is typically a small file that contains:

- The DC holder’s distinguished name (DN) with the identifying information for the certificate holder, which, depending on the identification criteria employed by the issuing CA, can include the common name of the certificate holder, holder’s organization (i.e., company name), any organizational units, street address, fully qualified domain name (i.e., company URL), an IP address, and e-mail address
- The certificate issuing CA’s distinguished name, which will include the same set of information as that of the holder’s DN
- The certificate holder’s public key
- The CA’s digital signature, which is a hashed (i.e., encoded) code that in turn is encrypted with the private key of the CA and is then used as the seal of authenticity for the DC. A digital signature, which
should not be confused with a digital certificate, is the equivalent of a tamper-detection seal on a pharmaceutical product.

- The validity period (i.e., shelf-life) for the DC
- A unique serial number for the DC

Today, there are several different types of DCs, each for a specific security scenario. Some of the main types of DCs include:

1. CA certificates, which digitally validate the identity of the CA; they contain identifying information as well as its public key. CA certificates can be signed (i.e., further validated) by another CA (e.g., VeriSign) or be self-signed if the CA does not require additional validation. A self-signed certificate is referred to as a trusted root for obvious reasons. Self-signed certificates, unless from large corporations, may not be appropriate for Web-service applications.

2. Server certificates to validate a server’s identity and provide information as to who owns the server. They will contain the public key of the server’s owner. These could be used by both sides to make sure that the servers on which the application and the Web service are running have not been changed without adequate notification.

3. Client or user certificates, which validate the client’s identity and contain the client’s public key. Authentication of server and client certificates can be used as the basis for encrypted, client/server communications per the SSL protocol.

4. Object signing certificates, which are used as the basis for digital signatures that vouch for the integrity of the accompanying object as well as the originator of that object.

<table>
<thead>
<tr>
<th>Table 7.2</th>
<th>Summary of Which Keys Are Used for Which Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations</td>
<td>Key Used</td>
</tr>
<tr>
<td>Send an encrypted document</td>
<td>Receiver’s public key</td>
</tr>
<tr>
<td>Send an encrypted digital signature</td>
<td>Sender’s private key</td>
</tr>
<tr>
<td>Decrypt a received encrypted document</td>
<td>Receiver’s private key</td>
</tr>
<tr>
<td>Authenticate received digital signature</td>
<td>Sender’s public key</td>
</tr>
</tbody>
</table>
The widespread use of DCs as well as digital signatures to validate the authenticity of a document (e.g., XML document) is made considerably easier by PKI. PKI, as indicated by the inclusion of the term *infrastructure* in its name, provides an incisive framework for facilitating secure but at the same time accessible and easy-to-use public-key encryption and digital signatures. The term *public* in the title, however, can be deceptive. PKI is not a centralized, public service for distributing and managing keys. The term *public* in this context refers to the public key in the public-private key combination. Today, PKIs are typically implemented on a corporate basis with PKI facilitating products from a variety of vendors, including IBM, VeriSign, and Entrust. IBM now even includes a comprehensive PKI on its flagship z/OS operating system for mainframes.

PKI sets out to hide the intricacies involved in providing a robust public-key encryption system so as not to intimidate users. In the true spirit of an infrastructure sustaining a utility such as that of the electric power grid or the cable TV network, PKI is a nonintrusive, transparent service. You can use PKI without actually knowing or caring that it is there, what it does, or how it does it.

There is a three-way, symbiotic relationship between PKI, digital certificates, and digital signatures. The current use of digital certificates for Internet applications is contingent on PKI. Without PKI, DCs would not be as convenient to use or administer. PKI consists of four primary components:

1. CAs that issue and validate digital certificates
2. One or more registration authorities (RAs), which work with the CA to verify a requester’s identity prior to the issuance of a digital certificate
3. One or more public directories (typically based on LDAP), where digital certificates, which will contain the public keys of their owners, can be stored
4. A comprehensive and foolproof certificate management system

As an infrastructure scheme, PKI is expected to cater to key backup and recovery. Users typically have to enter a password to access their public-key encryption keys. But users forget their passwords or may leave the company. If there is no way to recover encryption/decryption keys in such instances, valuable company information, which has been previously encrypted, may be forever lost. Thus, it is essential to have a secure and reliable way whereby a designated and fully validated security administrator can recover
encryption key pairs through the PKI system. Since such recovery will be possible only if the key pairs are properly backed up, it is assumed that the CA will automatically keep a backup of keys that have been issued.

In this context of backup and recovery, it is, however, important to note that recovery is only required for the key used for encryption/decryption. Keys used for digital signatures are not expected to be backed up or recoverable. The reason for this is that having duplicate (i.e., backed up) and thus recoverable signing keys undermines the overall integrity of a PKI. This has to do with something called nonrepudiation.

Repudiate means to disown or reject an action. Thus, denying involvement in a transaction is known as repudiation. Disowning a hotel charge made to a credit card via the signature-on-file scheme favored by North American hotels is an example of a repudiated transaction. Nonrepudiation precludes a person from being able to disown having participated in a transaction. In the digital world this means ensuring the sanctity of digital signatures. A prerequisite for this is the inability of a user to deny using his or her digital signature—hence, the undesirability of having backups of signing keys.

If there are no backups and the user has sole and secure custody of the signing keys, then the latitudes for repudiating a digital signature are considerably reduced. However, just as in the nondigital world, with pen and ink–based signatures, there will still occur situations where users will try to repudiate digital signature–backed transactions. At this point, other electronic logs and traces will have to be used to enforce nonrepudiation.

The loss of a signing key, unlike that of an encryption key, is not catastrophic to business. The user will just have to obtain new keys to use with future signatures. The desirability of keeping backups of encryption keys but not those of signing keys means that users should not be allowed to use the same keys for both purposes because this will again mean that there will be more than one copy of the signing keys. A PKI must support two key pairs per user: one pair for encryption and the other for digital signatures.

Then there is the issue of the longevity of key pairs. To maximize security, key pairs should be periodically updated. A PKI can enforce this based on key expiration thresholds. However, since users may need to go back and decrypt documents encrypted with older keys, the PKI also needs to maintain a history whereby prior keys can be securely recovered. Again, this would not apply to signing keys. The PKI should destroy the old signing key pair each time a new pair is assigned.
7.2 Security considerations for Web services

Another issue pertaining to PKI and DCs relates to certificate revocation. In the event of a security compromise, DCs may have to be immediately revoked in advance of their built-in expiration setting. This may happen, for example, if the private key corresponding to the public key published in a DC gets into the hands of an unauthorized individual. Another instance would be when a user leaves a company. To cater to certificate revocation, a PKI must maintain a scalable mechanism to publish the status of all certificates, whether they are active or revoked. Application software validating a DC will first make sure that it has not been revoked.

The final topic related to PKI has to do with cross-certification, given the current absence of a global PKI scheme. Cross-certification is also referred to as PKI networking. Cross-certification permits multiple, autonomous CAs to work cooperatively so that they have a mutual trust relationship. One scenario for this might be that of CAs for trading partners having the ability to validate DCs issued by the other CAs. Another scenario might be that of a large multinational enterprise that decides to implement CAs on a geographical basis for scalability and management purposes. Yet again, today's leading PKI solutions are typically able to address all such requirements.

7.2.2 Two-factor authentication

Two-factor authentication, as with DCs, is another powerful and proven authentication system that can be used by itself to authenticate Web services and applications to each other, or in conjunction with another scheme (e.g., DCs). Two-factor authentication gets its name from the fact that it requires users (or applications in this instance) to identify themselves by using two unique factors, one on top of the other. One of the factors would be something users know (e.g., password or PIN), while the other factor would be something they have. Automated teller machine (ATM) cards, though not based on RSA's SecurID, are an easy-to-understand example of two-factor authentication, where your personal PIN is the factor that you know, while the ATM card with its encoded magnetic stripe is the factor that you have.

In reality, RSA does offer an ATM card-type system based on a chip-embedded smart card and a smart card reader connected to the user's PC. Such a smart card-based system, though offering exceptional security, is obviously not applicable for application to Web-service interactions. Fortunately, RSA's SecurID can be and is widely used very effectively by millions of users on a daily basis, without the use of this smart card system.
RSA’s SecurID system, in general, works with a user-specific secret password and a token. The password is the factor that a user/application knows, while the token becomes the factor that the user has. The token is a time-synchronized code, which is periodically generated, typically every minute, starting with a unique seed code supplied by RSA. RSA can determine the validity of the token based on the time-sensitive code entered by the user. A valid token proves that the user has access to the factor he or she is supposed to hold. This token in effect becomes the equivalent of the magnetic stripe on an ATM card. To be successfully authenticated, a user/application has to enter this continually updated code (i.e., the token) and the user-specific secret password. RSA tokens can be generated by using RSA supplied software, which can be run on many different platforms. Obviously in a Web-service scenario it is important that this token-generation software and the user-specified passwords are safeguarded to ensure total security.

Since it is essentially an extension to normal password-based security schemes, applications can be modified to use two-factor authentication relatively easily and quickly. Typically a system administrator would have to enter a regularly changed password, as a seed, to activate the token-generation software. If this password is valid, the security software will automatically generate the token, append it to the entered password, and then send this combined key, suitably encrypted, to the security server for authentication. Despite its simplicity from an end-user perspective, two-factor authentication is obviously a significantly more powerful security scheme than a password-only (i.e., one-form) authentication scheme. Large and high-profile high-tech companies such as Cisco have been successfully using two-factor authentication, initially with hardware tokens but with a rapid transition to software tokens, since c. 1995 to partition and safeguard their employee-only intranet portals.

7.2.3 Secure Sockets Layer

SSL, or its successor TLS, should be considered a mandatory prerequisite technology for any Web service that deals with sensitive information. SSL, a client/server-based security mechanism developed by Web browser pioneer Netscape Communications in 1996, is the accepted and trusted basis for most of today’s secure transactions across the Web. SSL is so widely used that it appears to be ubiquitous. It is supported by all popular commercial Web browsers and Web servers (e.g., Microsoft’s Internet Information Server [IIS], the open software Apache Software Foundation’s Apache HTTP Server, IBM’s HTTP Server, etc.). In the context of Web servers it is
used between a browser and the Web server. It can also easily be used in other scenarios—in particular, application-to-application setups.

The little locked padlock icon that gets displayed at the bottom of a Web browser window whenever a secure transaction is being performed (e.g., credit card purchase or online stock trading) indicates that the security in force has been supplied via SSL technology. Figure 7.9 shows an example of the locked padlock icon indicating that SSL security is in force. Whenever the locked padlock icon is on display, the address field at the top of the browser showing the URL invoked is likely to say “https://www. etc. etc.,” rather than “http://www. etc. etc.” The “s” following the HTTP denotes SSL—in this case HTTP with SSL, or HTTP over SSL. HTTPS transactions are usually conducted across port number 443, while HTTP typically uses port 80. SSL, consequently, is not something new or rare. It is a scheme that most people have already encountered and have successfully used even without realizing that they were doing so. Ports 80 and 443 will typically also be the ports used by Web services if they are relying on SOAP over HTTP/HTTPS as their transport mechanism.
SSL is a Transport Layer (i.e., Layer 4) protocol. As such it provides authentication, integrity, and data privacy for applications running above the TCP Layer (i.e., Layer 3). SSL supports digital certificates. Given its cli-

Figure 7.10  High-level view of the process involved in setting up a secure SSL connection between a Web server and a Web browser user. This process is the same for secure SSL connections between other client/server pairs, including an application and a Web service.
ent/server orientation, SSL uses digital certificates to authenticate the server and the client—in this case the application and the Web service it intends to invoke. This authentication process, which is achieved via what is referred to as an SSL handshake, typically requires a user ID/password exchange—with the user ID and password being conveyed in encrypted mode with a public key. Following this authentication process, the SSL protocol sets about negotiating a common encryption scheme acceptable both to the client and the server. SSL does not do end-to-end data encryption.

Providing end-to-end client-to-server encryption was never a goal of the SSL protocol. There are well-established industry standards (e.g., 56-bit DES and 168-bit triple DES) and commercial ciphers (e.g., RSA) for enforcing end-to-end security. What SSL does is negotiate an encryption scheme acceptable both to the server and the client (e.g., triple DES)—and then invoke this mutually accepted encryption scheme for encrypting the data flowing between the client and the server. Thus, the security services provided by SSL can be summarized as server authentication via digital certificates, optional client authentication with digital certificates, acceptable encryption scheme negotiation between the server and the client, and invoking the accepted encryption scheme to ensure that the data flowing between the client and the server is indeed encrypted and tamperproof on an end-to-end basis.

The latest version of SSL is called Transport Layer Security (TLS). The two terms will get used interchangeably in the future, with SSL most likely being used more often, even when TLS is being referred to, given its current popularity. Figure 7.10 provides a high-level schematic of how a secure SSL connection is established between a Web server and a browser user.

### 7.3 A predeployment checklist for Web services

Despite the allure of being considered the latest and greatest in software development methodology and the fleeting fame of once having been hailed as the next big thing on the Web, one should never lose sight of the fact that Web services, in the end, are still nothing more than another incarnation of client/server technology. Consequently, though one may want to believe that things must be different and more difficult, in reality all of the deployment- and management-related issues that pertain to application scenarios involving Web services are no different from those that apply to any other client/server configuration. Distributed computing is not new, and the vanguard of today’s systems can be traced back to the mid-1960s. The
advent of PCs in 1981 accelerated the interest in client/server in relation to worldwide commerce, and to this end it is worth noting that IBM’s SNA-based LU 6.2 advanced program-to-program communications (APPC) was introduced in 1982 for mainframe-to-PC applications around the same time that UNIX distributed computing was making a name for itself.

Control, or to be precise who controls what, is the one area where the Web-services picture differs, in general, from most previous client/server permutations. In the past the client/server model was typically used in what would now be referred to as intranet/extranet configurations. All the software, both on the clients and the servers, would be under the control of one entity. Though the computing was distributed, the control of the software was centralized. Web services offer an alternate model, though corporations at present are opting to ignore this Internet model and are yet again falling back on intranet/extranet deployments. What is important here is that the deployment- and management-related issue in relation to Web services is still the same as for other client/server implementations.

Prior to the emergence of the Web, client/server implementations enjoyed a star-crossed, checkered history. Hindsight has shown that the total reliance on standards contributed much to the success of the Web model. Web services set out to capitalize on that. So at least in the case of Web services, unlike with enterprise-level client/server solutions as recently as the early 1990s, one does not have to wrestle with network protocol (i.e., IPX, NetBIOS, SNA, TCP/IP, OSI) or network topology decisions. Web services, by definition and design, will be standards compliant and as such conform to a consistent networking profile. This means that network compatibility issues, for a change, will not require much debate. One should thus be able to quickly tick off the SOAP, WSDL, etc. related concerns and devote more attention to the other issues.

Given the client/server nature, there will always be two separate sides to be considered for all of the deployment- and management-related issues—with overlap from both sides to cover the network interface as well. As discussed in the preceding sections, most enterprises will already have a firm handle on the issues pertaining to the calling application side, given that this is the driver’s side of the equation. Enterprises will already have experience with selecting, deploying, maintaining, and managing applications. It is the external, outside-the-firewall Web-services aspect that will be alien—and as such will demand attention. The issues, as with other client/server software, that one should focus on at this juncture include:

- Pricing and licensing
Service-level agreement, which covers availability, performance, and scalability (e.g., one-second turnaround time, 20 transactions per minute, etc.)

Maintenance contract, which deals with how any problems with the software will be handled, including how problems will be reported and escalated, whether there will be a charge for problem resolution, testing of the fix, end-to-end change management, and so on

Liability protection, particularly against bad data, data interception by employees, unauthorized use of information, and use of unauthorized software (e.g., copyright violations)

Upgrade policy covering notifications, regression testing mechanisms, and cut-over procedures

Long-term indemnification, if applicable (e.g., source code deposited in escrow)

Platform selection

Assignability—that is, options in the event of a merger or acquisition, and whether the software (i.e., the Web service) or the operation of that software can be sold to another party during the period of the contract

Privacy policy and guarantee

Security safeguards, including intrusion detection (especially in the context of a platform being compromised)

Load balancing and failover configurations

Disaster recovery scenarios

Intellectual property issues and waivers

Marketing and publicity rights—e.g., service providers wishing to publicize that so-and-so uses their \textit{xyz} Web service

Management of the software covering change management, problem management, performance management, network management, and gateway (e.g., XML application firewall) management

Usage monitoring and reporting

Of these, the pricing and licensing of Web services are likely to prove to be the most intriguing. Way back when Web services were first being bandied around, there was an implication that many services would be available on a no-charge, freeware-type basis. This could still prove to be the case,
though IT professionals have known from the start that most Web services they would wish to use or, conversely, offer to others are likely to have some cost and licensing associated with them. To this end, Microsoft’s statement (reproduced here) as to how one can go about purchasing its trend-setting MapPoint Web service offers a glimpse of how the pricing model will evolve:

Customers purchase the MapPoint Web Service as an annual subscription direct from Microsoft. There are two primary licensing models:

1. **Per user** is for “known” user applications, such as within a call center or fleet tracking applications
2. **Per transactions** is for “anonymous” user applications, such as a Web site locator or travel portal

Pricing is dependent on the numbers of users and/or transactions you purchase. More information can be found on [http://www.microsoft.com/mappoint/webservice/howtobuy.mspx](http://www.microsoft.com/mappoint/webservice/howtobuy.mspx).

Suffice to say that the pricing model for Web services is at present in an embryonic stage, with most people, quite rightly, still focusing their attention on the technical side of things. IBM, a major shaker and mover when it comes to Web services and a company that knows a thing or two about complex, usage-based pricing models, has to its credit already started publishing position papers about possible accounting and metering schemes for Web services. Microsoft is already espousing usage-based pricing. What is abundantly clear is that there will be a very wide spectrum, ranging from freeware to expensive, premium offerings coupled with many permutations, in the case of the nonfreeware offerings, as to how the pricing will be structured.

The pricing options available will definitely include one-time charge schemes, periodic licensing (e.g., monthly or yearly), and umpteen usage-based options. In addition, it is likely that there could be third-party Web services distributors—though the inherent dynamic discovery capability of a Web service dilutes some of the potential value that can be offered by a distributor. In the case of Web services the main value that a distributor is likely to provide is that of handling the billing and collection. Indubitably, there will also be syndicated Web services. Figure 7.11 sets out to highlight some of the possible pricing models that will be available for Web services, depending on who is offering the service.

The implications of other items in the preceding list should be self-explanatory, but the management-related issues are elaborated below. As with security, how one would go about addressing each of the items in this
list will depend and vary on a case-by-case basis. There are no hard-and-fast rules that will cover all instances. Remember that just because we are dealing with a new software methodology does not mean that the famous French adage “plus ça change, plus c’est la même chose” (the more things change, the more they remain the same) ceases to apply. The software-related issues, such as failures, upgrades, change management, and intermittent slowdowns, will still be the same.

7.4 The platform issue—yet again

It goes without saying, but it has to be said for completeness, that there are two platforms that need to be considered for each Web-service invocation:

1. The platform on which the calling application is running
2. The platform hosting the Web service being invoked

Within the context of this duality, one can easily identify the permutations shown in Table 7.3 as being likely candidates,

The platform issues, in the Web services context, as discussed in the context of both .NET and Java, pertain to:

1. Capacity (i.e., how many concurrent sessions it can handle)
2. Performance (i.e., delivery of expected response times under normal workloads)
3. Scalability (i.e., what the upper limits are when it comes to handling increasing workloads)

4. Overall stability in terms of reliability, availability, and serviceability (i.e., the platform can meet the expected uptime criteria, given that mission-critical applications for Fortune 500 corporations require 99.999% uptime over a year)

5. Vulnerability (i.e., the platform is noted for being safe from hackers)

6. Manageability (i.e., provides the necessary tools for remote monitoring and management)

Though these concerns have been discussed as they relate to .NET and Java, it is now important to yet again reiterate that both sides of the Web services model are platform independent in that the calling application for a Web service nor a Web service being invoked need to be restricted to a specific platform. This, as highlighted in Figure 6.1, does not mean cross-platform portability. It does, however, mean that the Web-services model is not confined to any specific platform permutations. Java and .NET are also not the only viable platforms for Web services or the applications that intend to invoke Web services. Any of the traditional platforms (e.g., IBM’s CICS, BEA’s Tuxedo) can also be used for either side. This is particularly germane in legacy modernization scenarios, as discussed in the beginning of this book, whereby proven business logic from existing mission-critical applications can be isolated and repackaged as Web services using host integration tools such as IBM’s WebSphere Host Publisher. The bottom line here is

<table>
<thead>
<tr>
<th>Calling Application</th>
<th>Web Service Being Invoked</th>
</tr>
</thead>
<tbody>
<tr>
<td>On the same system (i.e., collocated)</td>
<td></td>
</tr>
<tr>
<td>On the same system, on different (but homogeneous) partitions</td>
<td></td>
</tr>
<tr>
<td>On the same system, but on heterogeneous partitions (e.g., z/OS and Linux on a mainframe)</td>
<td></td>
</tr>
<tr>
<td>System_A</td>
<td>System_B</td>
</tr>
<tr>
<td>But System_A and System_B are of the same type (e.g., UNIX servers)</td>
<td></td>
</tr>
<tr>
<td>System_A</td>
<td>System_B</td>
</tr>
<tr>
<td>But System_A and System_B are significantly different</td>
<td></td>
</tr>
</tbody>
</table>
that any platform, provided it offers an appropriate transport scheme, can be used for Web services–based application scenarios.

The goal when it comes to platform selection, however, is to ensure that the platforms selected adequately meet the performance, uptime, security, and manageability expectations for the overall application in question. The big danger here is ending up with totally mismatched combinations, such as a high-volume, mainframe-resident, mission-critical transaction processing application (e.g., travel reservation system) serving 50,000 concurrent users relying upon a crucial, oft-invoked Web service that happens to be deployed on an old Pentium III server still running Windows NT 4.0. But you cannot make generalizations even with this very uneven combination.

It is indeed possible, depending on the processing involved, that the out-of-date Windows platform may still provide the necessary functionality, with adequate resiliency to meet expectations, especially if the server is located in a data center that is manned 24/7. One assumes that one would not be as concerned if the platforms were reversed, so to speak (i.e., a VB application running on an old WinTel server relying on a Web service deployed on a newly upgraded, multimillion dollar mainframe). However, it is fair to say that mismatched platform combinations invariably warrant particular scrutiny to avoid costly breakdowns in expectations down the road—when the application is in full swing, handling mission-critical production work.

The desirability of cross-platform portability, such as Java, is a contingency against mismatched platform pairing. If a Web service is portable, then one does have the option, albeit at a cost, of having it hosted on a different platform if one has reservations about the platform it is currently being offered on. Given that this platform specificity is primarily going to be an issue if a Web service is limited to Windows, one still could have other options to mitigate the situation. Key among these would be to acquire or license the Web service so that it can be run in-house, behind the corporate firewall, on a carefully tended, dedicated WinTel server—or, per the current trend, on a WinTel blade on a server rack.

The bottom line here is that with all of the debate that has ensued around .NET and Java, enterprises now have enough background and parameters to evaluate the platform criteria for specific Web-service scenarios—indeed, independent of whether Java or .NET even comes into the picture. With all of the various platforms and platform options available today, IT professionals, given a set of service-level criteria, should be able to craft appropriate pairings to meet expectations.
Managing it all

The disciplines and technologies required to incisively manage Web services–based application scenarios fall under the umbrella of what system vendors and system integrators have referred to since the start of the 1990s as total enterprise management, or system management. Total enterprise management goes way beyond just network management to encompass hardware platforms, operating systems, applications, operational procedures, and even business issues (e.g., making sure maintenance contacts were paid in time and kept current). IBM, an early proponent of this strategic field, neatly captured what it was all about by branding it SystemView—to contrast it from NetView, at one point the gold standard for network management in Fortune 1000 companies.

Though many corporations have yet to get around to exploiting the true potential of total enterprise management, the requisite enabling technology...
is well established and readily available from a wide number of bluechip names. Thus, proven total enterprise management solutions, which also encompass the needs of managing Web services, are available from the likes of IBM/Tivoli, Computer Associates (CA), H-P (under the well-known OpenView marquee), and Sun (i.e., Sun Management Center). Consequently, one does not have to be concerned that managing Web services–related software scenarios is going to be hampered by a paucity of appropriate management tools.

The key features of Sun's Management Center, according to Sun, are shown in Figure 7.12, and one should note the management disciplines of particular interest to Web services, such as change management and service availability management. At this juncture it is also useful to see what CA, an acknowledged leader in enterprise management with its Unicenter family of products, has to say about system management of distributed, multivendor environments of the type that will be used in Web-services scenarios. CA's perspective, shown in Figure 7.13, clearly captures what is indeed possible with today's system management tools.

In addition to the holistic total enterprise management solutions that treat Web services as just another software deployment configuration, there

---

**Enterprise Management**

**Unicenter Network and Systems Management**

*NETWORK & SYSTEMS MANAGEMENT AND MORE!!*

Systems Management software components perform specific tasks to monitor and manage the health and availability across complex, multi-vendor, multi-platform computing environments.

Systems Management solutions reduce the complexity of deployment and use of complex technologies by applying self-managing capabilities to deliver high levels of automation that can relieve the staff.

The Unicenter Network and Systems Management product manages the health and availability of operating systems and provides basic status management on all infrastructure elements such as network devices, business applications and database systems. Powerful auto-discovery builds a database with information on system elements and populates 2D and 3D system dynamic visualizations. Historian keeps you informed with past events and object status whereas predictive management capabilities inform you about possible bottlenecks in your systems and applications in future to take automated actions to avoid them. Portal technology provides personalized intuitive information for both technical and business focused administrators.

**Figure 7.13** A composite of material from Computer Associates Web site showing its perspective on system management. Keep in mind that CA has a long and respected track record in the management arena.
are also specific Web-services management offerings from the likes of AmberePoint (www.amberepoint.com). Figure 7.14 shows an overview of AmberePoint’s solution, which demonstrates that it recognizes the need to bring management, monitoring, and security capabilities to Web-services systems so that they can be transformed into trustworthy, enterprise-class business assets. Also note that per the repeated themes in this chapter, the AmberePoint offering also targets change management (i.e., enabling online upgrades and versioning), service-level monitoring, and multiplatform management. Yet again the message here is that, as with security measures (e.g., digital certificates, SSL), the necessary knowledge and technology for managing Web services are already in place.

The Web services–related entities that need to be managed with the new application model (Figure 7.15) include:
1. The calling application
2. The platform hosting the calling application
3. The (remote) Web services
4. The platforms on which the various Web services are running
5. Web services gateways (e.g., XML application firewall)
6. Network connection

The need to monitor and manage distant resources owned by third parties, as will be the case with remotely invoked Web services, will be seen by some as adding a whole new dimension of complexity to the overall management challenge. This is not necessarily so. Today’s IP-based, Internet-oriented management methodology, by and large, is location and distance independent. Entities are identified and managed per their IP addresses. Thus, the exact location of an entity is immaterial to the management software. As long as the underlying network can locate the required entity (e.g., be able to ping it), the management software will be able to access it irrespective of whether it is located on the same system or halfway around the world. In this context it is also worth remembering that enterprises now have considerable experience managing remotely hosted Web servers. Remotely hosted Web services are but an extension of this model.

Being able to monitor and manage third-party-owned Web services on remotely located platforms should, however, only be possible with the

Figure 7.15 The magnifying glass icons in this diagram signify the Web services–related entities that need to be monitored and managed with the new application model.
explicit blessing of the Web-service owner—as well as the operator of the remote platform, if the service is being hosted by another. One obviously would want to steer well clear of Web services that supposedly offer an open, unrestricted management interface, given the potential security vulnerabilities of such an unregulated service. Thus, being able to monitor and manage a remote Web service should be contingent on preestablished and validated authentication, access control, and data privacy (e.g., encryption) mechanisms. The management interface should not be a possible point of vulnerability (i.e., the weakest link) in the overall security architecture, since a determined hacker will most likely begin with probing this interface when looking for a way to break into the system.

What is possible and permitted when it comes to the remote monitoring and management of a Web service should be something that is explicitly discussed during the evaluation phase and agreed upon, contractually, prior to production use. It is not something that you would want to tackle on an ad hoc basis after you have already started to use a service. It is also important to remember that this remote management capability, though technically possible and philosophically pleasing, may not always be necessary or practical.

In some instances, depending on the nature of the service or the stature of the provider, one may not want to manage the remote service. Managing software, despite the assistance of the management tools, is a specialized function and should be undertaken only by knowledgeable, trained, and responsible system administrators. Thus, there is a tangible cost component associated with instituting and operating a remote software monitoring capability, especially if it is to be on a 24/7 basis. Consequently, in some instances it may be more practical to leave all the monitoring and management responsibilities with the service provider. A real-time collaborative scheme, based on e-mail, instant messaging, or cell phones, could then be used if any critical issues need to be communicated between the service provider and the service user. There will also be cases where the service provider is unwilling to provide direct monitoring and management capabilities and instead will be willing to offer only a notification mechanism. Proven and incisive technology to facilitate remote management of Web services is readily available, and management is not an impediment when it comes to Web-services exploitation.

The key total enterprise management disciplines that are pertinent when it comes to Web services include:

- Problem management: This obviously applies to all entities, end to end, including the network, and one may want to augment this with
filtering and consolidating capabilities to prevent one underlying problem from being reported multiple times, with different flavors, from each of the managed entities.

- Performance management, including workload management and capacity management
- Application management, which embraces resource usage monitoring, user administration, and usage accounting
- Change management, particularly as it applies to the remote Web services
- Security management, which includes intrusion detection notification
- Usage monitoring
- Operations management, which with today's technology can include both automated operations (e.g., backups) and operator tracking (e.g., time taken to respond to an intervention request)

Management portals, where the information streams from multiple management disciplines, can be filtered, correlated, and structured better to aid comprehension, are also now becoming the focal point for total enterprise management. CA's Unicenter Management Portal, as shown in Figure 7.16, is a quintessential example of such consolidated management portals. Another emerging trend, which will in time play a role in Web-services

---

**Figure 7.16** The concept of a management portal, as exemplified by CA's Unicenter Management Portal.
management, is that of self-managing software, as epitomized by IBM’s ambitious autonomic computing program (née Project eLiza) and Microsoft’s new Dynamic Systems Initiative (DSI). The bottom line is that managing Web services–based application environments, though not a simple undertaking, is not going to be hampered by a lack of the necessary tools and expertise.

### 7.6 Q&A: A time to recap and reflect

**Q:** Have corporations had a change of heart when it comes to Web services?

**A:** Yes and no. The technological underpinnings of Web services and their potential when it comes to developing new enterprise applications are essentially beyond reproach and are not in question. However, the overall climate of distrust and security-related paranoia that has permeated North America and Western Europe since 9/11, exacerbated by the unending attacks on security soft spots on Windows platforms, has had a marked impact on how corporations are currently opting to use Web services. Rather than pursuing applicable Web services that may be on offer on the Web from unknown third parties, corporations, for the time being at least, are electing instead to exploit Web-services technology as a new, standards-based component model for software being developed in-house or by trusted partners. In essence the use of Web services is being confined to intranets and extranets.

**Q:** Is security an insurmountable issue when it comes to Web services?

**A:** No—most certainly not. While one cannot downplay the need for stringent security in any Web services–related scenario, it is important to keep things in perspective and to objectively evaluate the security implications of using a particular service on a case-by-case basis. Concerns about security need to be proportional to the actual sensitivity of the data involved. Thus, one should not generalize matters in this instance. There will be many Web services that deal in relatively innocuous data—for example, a weather service or even Microsoft’s MapPoint Web Service for mapping applications. Some will argue that knowledge of even the invocation of an external Web service, irrespective of the data that was exchanged, may provide a competitor or hacker with some level of insight. Again, it all depends on the application. One should use Web services with portals—particularly public portals. If such a portal routinely invokes Web services dealing with weather, mapping, horoscopes, stock indexes, and so on, one will be hard pressed to argue that knowledge that these Web services are being invoked would provide a competitor with significant intelligence—even though
some would counter that knowing the rate at which these services are being invoked could indicate the popularity of that portal. The point here, however, is that it is indeed possible to have scenarios involving remote Web services where security should not be the show-stopper—with the caveat that one always does have the option of still using available and appropriate security measures such as authentication and encryption.

Q: Is adequate security technology currently available to make remotely invoked Web services viable?

A: By and large the answer to this has to be a resounding “yes.” At this juncture one should not lose sight of the fact that e-commerce continues to grow and that trillions of dollars worth of financial transactions, including online banking and online trading, take place over the Web on a daily basis. All of the technology used for these types of transactions, such as digital certificates, digital signatures, SSL, and two-factor authentication, are also available to be used in Web services–based applications. A big concern with Web services is that of rogue Web services, where this could be due to an unscrupulous provider or a compromised platform. However, similar rogue trader or rogue Web site scenarios are possible with e-commerce. One minimizes the danger of this by dealing with reputable sites. The same should apply to selecting Web-service providers. Provided you are dealing with a reputable service provider and are using all of the available and pertinent security technology (e.g., digital certificates, SSL) it is indeed feasible to use remotely invoked Web services—particularly if one is dealing with data that is not ultrasensitive.

Q: Is it possible to locate, evaluate, license, and start using a Web service, all on the fly, over the Web, similar to buying and downloading a virus protection subscription online, using a credit card?

A: Yes, this will indeed be possible. This on-the-fly, everything done across the Web, e-commerce-based model, facilitated by UDDI, was the initial and compelling vision for Web services. It was even going to be possible to do all of this programmatically with the UDDI APIs. This original software functionality on-tap, over the Web, paradigm will continue to prosper, and one might check to see that the oft-mentioned Microsoft MapPoint service does indeed pander to this. This model will be the only one of interest to a large portion of the software development community (i.e., the cottage-industry sector made up of talented individual programmers and those who are working for small firms). Large enterprises selecting best-of-breed Web services for mission-critical applications may only espouse this totally dynamic, all-electronic paradigm for Web-service evaluation when the software is being invoked from stringently partitioned test machines. Once
they have located a promising service, the norm today is to revert to a traditional, big-ticket purchasing routine involving reference checks, credit checks, background checks, contracts, and so on.

Q: What are digital certificates?

A: Digital certificates are electronic credentials issued by a trustworthy organization such as a large company or a security-specific entity such as VeriSign. A DC vouches for an individual’s, a business’s, or an application’s identity and authority to conduct secure transactions over the Web. DCs are in essence the Internet equivalent of a travel passport. They are a universally accepted means of establishing one’s identity and thus gaining entry to a protected resource. DCs are meant to replace traditional user IDs and passwords, which are not as secure or trustworthy. They are thus the preferred means for authenticating users and applications in today’s Internet world.

Q: What is the role of SSL in relation to Web services?

A: SSL, a client/server-based security mechanism developed by Web browser pioneer Netscape Communications in 1996, is the accepted and trusted basis for most of today’s secure transactions across the Web. SSL is a Transport Layer (i.e., Layer 4) security protocol that works on a client/server basis. It provides authentication, integrity, and data privacy for applications running above the TCP Layer (i.e., Layer 3). SSL uses digital certificates to authenticate the server and the client of a particular transaction. In the case of Web services, this authentication would be for an application and the Web service it intends to invoke. The authentication can be bidirectional (i.e., both the application and the Web service make sure that they are indeed talking to whom they think they are). Following a successful authentication, the SSL sets about negotiating a common encryption scheme acceptable both to the client and the server. SSL does not do the end-to-end data encryption. SSL relies on well-established industry standards (e.g., 168-bit triple DES) and commercial ciphers (e.g., RSA). What it does is negotiate an encryption scheme acceptable both to the server and the client—and then invokes this mutually accepted encryption scheme for encrypting the data flowing between the client and the server. Thus, the security services provided by SSL in Web-services scenarios would involve bidirectional authentication via digital certificates, acceptable encryption scheme negotiation between the application and the Web service, and invoking the accepted encryption scheme to ensure that the data flowing between the application and the Web service is indeed encrypted and tamperproof on an end-to-end basis.
Q: What are the new security standards being developed for the Web-services arena?

A: There are a raft of new Web services–related security specifications being worked on. In some cases the goal of these specifications is to add XML support to existing technologies (e.g., XML Signature Syntax and Processing as well as XML Key Management Specification [XKMS]). In other cases the goal is to add security measures to SOAP (e.g., WS-Security, WS-Security Addendum, WS-Trust, etc.). A list of these security-related specifications can be found in Table 1.1. For the latest status of these specifications, as well as for news on any new security-related specifications, one should visit www.xmlweb.org.

Q: Is it possible to have Web services that are not deployed on Java or .NET platforms?

A: Yes, most certainly. Java and .NET are today’s strategic and popular software development platforms, particularly in the corporate arena. Web services, though truly a technology of the twenty-first century, are not, however, confined to these two new software platforms. Web services can be deployed on any platform that offers an appropriate Web-oriented transport that a calling application is willing and able to support. Web services are programming language and platform independent—as is the case for the applications that wish to invoke Web services. Thus, it is indeed possible to have Web-services scenarios that do not involve Java or .NET. C and C++ on UNIX and Linux platforms are obvious candidates for Web-services scenarios, as are traditional transaction processing systems such as IBM’s CICS and BEA’s Tuxedo.

Q: What are the potential pricing models for Web services?

A: There will be a wide spectrum of pricing options for Web services, ranging from freeware to expensive, premium offerings, coupled with many permutations, in the case of the nonfreeware offerings, as to how the pricing will be structured. The pricing options available will definitely include one-time charge schemes, periodic licensing (e.g., monthly or yearly), and umpteen usage-based options.

Q: Is management going to be an impediment to the deployment and use of Web services?

A: No.