Caching **XML** Web Services

23/10/2007

DOUGLAS B. TERRY AND
VENUGOPALAN RAMASUBRAMANIAN,
MICROSOFT RESEARCH

for Mobility

In the face of unreliable connections and low bandwidth, caching may offer reliable wireless access to Web services.

Web services are emerging as the dominant application on the Internet. The Web is no longer just a repository of information but has evolved into an active medium for providers and consumers of services: Individuals provide peer-to-peer services to access personal contact information or photo albums for other individuals; individuals provide services to businesses for accessing personal preferences or tax information; Web-based businesses provide consumer services such as travel arrangement (Orbitz), shopping (eBay), and e-mail (Hotmail); and several business-to-business (B2B) services such as supply chain management form important applications of the Internet.

Although these services are provided through static or active Web pages, they are evolving into Extensible Markup Language (XML)-based Web services designed for programmatic rather than human access. For example, MapPoint.Net provides maps and location services for incorporation into other Web sites and applications. Thus, XML Web services are the building blocks for constructing a new generation of Web applications that leverage existing investments in Web technology.

One of the key requirements for the success of Web services is universal availability. Web services tend to be accessed at all times and in all places. People use a wide range of devices including desktops, laptops, handheld personal digital assistants (PDAs), and smartphones that are connected to the Internet using very different kinds of networks, such as wireless LAN (802.11b), cellphone network (WAP), broadband network (cable modem), telephone network (28.8-kbps modem), or local area network (Ethernet). Occasional to frequent disconnections and unreliable bandwidth characterize many of these

Caching XML Web Services for Mobility

networks. The availability of Web services is thus a significant concern to consumers using mobile devices and working in different kinds of wireless and wired networks.

A good solution to improve availability of Web services should be transparently deployable and generally applicable. Transparent deployment means that the solution must not require changes to the implementation of the Web services, either to the server and client-side

modules or to the communication protocol between them. The growth in the number of Web services has been phenomenal; hence, applying changes to existing Web services is impractical. For the same reason, the solution should be scalable and general enough to apply to all Web services. Building specialized components to handle disconnections for each Web service would be extravagant. A good solution, instead, would be applicable to all Web services and would involve interposing storage and computation transparently in the communication path of the client and the server without modifications to Web-service implementations on the client or the server.

Continued access to Web services from mobile devices during disconnections can be provided by a client-side request-response cache that mimics the behavior of Web services to a limited extent. Caching satisfies both the required characteristics of transparent deployment and general applicability. Caches are transparent to both the client and server components of the Web services. Hence, caching would require no changes to the implementation and the communication protocol of the Web service and

```
<?xml version="1.0" encoding="utf-8"?>
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:m="http://schemas.microsoft.com/hs/2001/10/myContacts" xmlns:c="http://schemas.microsoft.com/hs/2001/10/core"
xmlns:mp="http://schemas.microsoft.com/hs/2001/10/myProfile" >
  <s:Header>
    <licenses xmlns="http://schemas.xmlsoap.org/soap/security/2000-12">
      <c:identity> <c:kerberos>3240</c:kerberos> </c:identity>
    </licenses>
    <path xmlns="http://schemas.xmlsoap.org/rp/">
      <action>http://schemas.microsoft.com/hs/2001/10/core#request</action>
      <to>http://microsoft-m3we4f.microsoft.com</to>
      <fwd><via /></fwd><rev><via /></rev>
      <id>b55528a4-5d63-49f1-87a2-5fab8d76f658</id>
    </path>
    <c:request service="myContacts" document="content" method="insert" genResponse="always" >
      <key puid="3240" instance="1" cluster="1" />
    </c:request>
  </s:Header>
  <s:Body>
    <c:insertRequest select="/m:myContacts/m:contact[mp:name/mp:givenName = 'Joe']/mp:emailAddress" >
      <mp:email>joe@idontknow.com</mp:email>
    </c:insertRequest>
  </s:Body>
</s:Envelope>
```

FIG 1

Example SOAP message
carrying insert request for
MyContacts Web service.

can be applied to existing Web services that conform to World Wide Web Consortium (W3C) standards.

A variety of systems have used caching on mobile devices in support of disconnected access to files, databases, objects, and Web pages. Most modern commercial Web browsers allow users to access cached pages while offline. Web-browser caches map URLs to HTML pages and need worry about only one operation—namely, the HTTP GET operation. They rely on directives provided by Web servers that indicate whether a page is cacheable and for how long. XML Web services present new challenges as a result of the diverse set of operations exported by such services, as well as their lack of involvement in the caching process.

STANDARDS FOR XML WEB SERVICES

A cache architecture must strictly conform to the XML-based standards for Web services developed by the W3C [see “Extensible Markup Language (XML) 1.0 (Second Edition),” by Tim Bray, Jean Paoli, C.M. Sperberg-McQueen, and Eve Maler, Oct. 6, 2000; www.w3.org/TR/2000/REC-xml-20001006]. This section provides an overview of such standards.

Web services consist of a service provider and multiple consumers based on the client-server architecture. Each Web service uses a custom communication protocol for the clients to access the servers. The most common access pattern for a Web service consists of requests and responses. The client sends to the server a request message that specifies the operation to be performed and all relevant information to perform the operation. The server performs the specified operation and replies with a response message. The actions carried out by the server might result in permanent changes to the state of the server.

Essentially, Web services provide the client with interfaces similar to remote procedure calls (RPCs). For example, MyContacts, one of the .NET My Services [see *Microsoft .NET My Services Specification*, Microsoft Press, 2001] is a Web service that allows users to maintain names, addresses, and phone numbers of their contacts. The MyContacts Web service exports operations to **insert**, **delete**, **replace**, and **query** portions of this contact information. Each of these operations takes input parameters (the query string) and produces output (query response or success status). Each Web service provides its own custom interface that could be vastly different from those provided by other Web services. For example, a travel Web service would provide operations to search for airfares, reserve and buy tickets, and look up itineraries.

The W3C has recommended a set of standards for Web services based on XML, with the support of several leading corporations including IBM and Microsoft. These XML-based standards provide globally recognizable protocols for discovering, describing, and accessing the custom interfaces of Web services. [For more information, see *Professional XML Web Services*, by P. Cauldwell, et. al. Wrox Press Ltd., 2001.] This standard consists of two important components: Simple Object Access Protocol (SOAP) and Web Services Description Language (WSDL). [For more information, see “Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI,” by F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana, *IEEE Internet Computing*, March-April 2002, Vol. 6, No. 2, pp. 86-93.]

SOAP. SOAP specifies a standard for sending messages between different entities of a Web service [see “SOAP 1.2 Part 1: Messaging Framework,” by Martin Gudgin, Marc Hadley, Jean-Jacques Moreau, Henrik Frystyk Nielsen, Oct. 2, 2001; www.w3.org/TR/soap12-part1]. SOAP messages are XML documents that are transported from one SOAP node to another. For Web services, the SOAP nodes could be either the client or the server. Each SOAP message consists of an outermost element called the *envelope*. The envelope consists of two elements: a mandatory *body* and an optional *header*. The body element carries the main content of the message. For a request message, it would carry the name and parameters of the operation to be performed. The header element consists of multiple header blocks, each containing meta-information for the receiver or intermediary nodes. The header blocks specify additional useful information such as a password for authentication.

The SOAP message in Figure 1 shows an example request message from the client to the server for the MyContacts Web service. SOAP messages are generally transported using Hyper Text Transfer Protocol (HTTP) as the application layer protocol because the SOAP-RPC recommendations are complete only for HTTP. Accordingly, the request for an operation from a client is carried by an HTTP request message, and the response from the server is carried by the corresponding HTTP response message.

WSDL. WSDL is a standard used to provide descriptions of Web services [see “Web Services Description Language (WSDL) Version 1.2,” by Roberto Chinnici, Martin Gudgin, Jean-Jacques Moreau, and Sanjiva Weerawarana, July 9, 2002; www.w3.org/TR/wsdl12/]. The WSDL document for each Web service completely describes the custom interfaces provided by that Web service to clients.

Caching XML Web Services for Mobility

This document can be used by program development tools, such as Microsoft's Visual Studio .NET, automatically to generate proxy stubs that encapsulate the remote Web service as a local object on the client. The WSDL document lists the names of the operations provided by the Web service, as well as the format of SOAP messages used to communicate between the client and the server. The WSDL document also provides a complete description of the data types of the parameters to be passed to each operation or received as responses. Thus, WSDL provides adequate description of the varied interfaces provided by the Web service.

AN EXPERIMENT IN CACHING FOR DISCONNECTED OPERATION

To study the suitability of caching to support disconnected operation on Web services, we conducted an experiment in which a caching proxy was placed between Microsoft's .NET My Services and the sample clients that ship with these services. The .NET My Services were chosen for this experiment because, although they are not commercial services, they were publicly available at the time of the study, well documented, and are representa-

tive of non-trivial XML Web services that support both **query** and **update** operations.

The Microsoft .NET My Services Software Development Kit (SDK) contains a number of Web services such as: My-Contacts, which allows users to store and retrieve address and phone information; MyProfile, which allows users to store their personal information; and MyFavoriteWeb-Sites, which allows clients to manage favorite Web sites. Each Web service in the .NET My Services family exports four significant operations on shared databases:

- **Query** lets users select portions of the database. This operation takes an XPath string as the query expression.
- **Insert** takes data to be inserted in the database, as well as an XPath string specifying the location for the insertion.
- **Delete** takes an XPath query string and deletes the specified entries in the database.
- **Replace** performs an atomic **delete** and **insert** in the database.

The SDK also comes with several sample applications that call on these services. By running the sample applications, we performed various operations on these services having the network connected, as well as deliberately disconnecting the network. Figure 2 illustrates the setup of our experiments.

We built an HTTP proxy server as defined in the HTTP protocol standard ["RFC 2616: Hypertext Transfer Protocol - HTTP/1.1," by R. Fielding, J. Gettys, J.C. Mogul, H. Frystyk Nielsen, and T. Berners-Lee, IETF, January 1997; www.ietf.org/rfc/rfc2616.txt] and deployed it on the client device. In this case, the client device was a laptop running Windows XP. All HTTP messages originating at the client, including those generated by Web clients and Internet browsers, were made to pass through the proxy server. The proxy server acts as a simple tunnel for all HTTP packets that are not SOAP messages.

We added a cache for storing SOAP messages to the proxy server. Integrating the Web-service cache into a proxy provides transparency. This cache stores SOAP messages received in response to SOAP requests. All cache policies for expiration and replacement were implemented as recommended in the HTTP standard. During the experiment, this cache was used only to store HTTP packets with SOAP messages as their entities. Whenever the network is connected, the received SOAP request is sent to the server. The received SOAP response is stored in the cache associated with the request, replacing the old response if it existed. When the cache contains a previous response for the same request, the new SOAP response is compared with the previous response, and the result of

the comparison is recorded in a log file. These comparisons provided valuable insight into what kinds of operations could be cached and what other operations affect the validity of the cached responses.

If the network is disconnected, the SOAP response stored in the cache (if present) is returned to the client and the SOAP request is stored in a write-back queue. If a request has no cached response, the client times out waiting for the response, as in the normal case when a server is unreachable. All requests are stored in the write-back queue for later replay, because the cache manager cannot determine which requests modify the service state and which are simply queries. The write-back queue is responsible for periodically checking the network for connectivity. Whenever the connection to the Web service is restored, the queued-up SOAP requests are played back to the server and the SOAP responses are stored in the cache.

This experiment clearly highlighted the benefits of employing a Web-service cache to support disconnected operation. Specifically, we were able to demonstrate that the .NET My Services set of Web services, such as MyContacts, can be used during disconnections with little awareness of the disconnection. The applications ran just fine while disconnected as long as the cache was preloaded and the cache manager could identify similar requests, even though neither the applications nor the services were designed or modified to accommodate offline caching.

ISSUES IN CACHING WEB SERVICES

Our experiment in caching exposed a number of issues that need to be handled to achieve a significant improvement in the consistency and availability of offline access to Web services. This section elaborates on problems associated with designing a client cache for Web services.

Playback and cacheability. The diverse nature of Web services poses a major problem in identifying the semantics of the operations exposed by the Web service. In the case of file systems, the semantics of standard operations such as **read** and **write** are clearly understood. Results of the **read** operation can be stored in the cache, while the **write** operations need to be played back to the server upon restoration of connectivity. On the other hand, Web services have diverse interfaces that make it difficult for the cache to understand whether a certain operation needs to be played back to the server and whether an old response from the cache is acceptable to the client.

The Web-service cache needs to recognize at least two properties of an operation to function effectively. An

operation is said to be an *update* if its execution makes permanent changes to the state of the server. An operation is said to be *cacheable* if its subsequent execution with the same parameters produces the same response, provided that no update operation intervenes. Operations of a Web service could be both cacheable and updating, while others could be neither one or have one property without the other. For example, a request to **query** data is cacheable but generally is not an update. A **query** could also update the server state, if the server needs to maintain a log of all requests. A request to get the current time is neither cacheable nor an update.

Consistency. A fundamental challenge faced by caching schemes in general and compounded by the diverse nature of Web services is providing basic consistency guarantees. When operating in disconnected mode, a cache manager cannot provide strong consistency guarantees because it does not have access to updates performed by other users. It can at least strive to provide cache results that are consistent with a user's own actions, however. In particular, an operation performed by the local user could change some of the results of the earlier requests that are stored in the cache.

Consistency requirements may demand that responses for certain requests stored in the cache be invalidated as a result of the execution of later requests. For example, in the MyContacts Web service, a request to change the telephone number of a friend or delete the entry altogether should invalidate an earlier response querying the contact information of that friend. To maintain accuracy, the earlier response in the cache would have to be either correctly modified or deleted from the cache. Otherwise, the cache may return an incorrect response if the query is repeated during a network outage. For preexisting Web services, understanding the correct consistency requirements—that is, the interdependencies between operations—is an extremely challenging issue.

The ultimate effectiveness of a Web-service cache depends on how well it can support the consistency semantics of diverse Web services. Consider two operations performed in succession by a Web-service application: **request₁** and **request₂**. Let the response for **request₁** be currently stored in the cache. Suppose that **request₂** is being processed. Whether **request₂** invalidates **request₁** depends not only on the operations being performed by the two requests, but also on the parameters being passed to the two operations. The cache manager must correctly understand the condition for the invalidation of **request₁** by **request₂**.

Caching XML Web Services for Mobility

Going beyond invalidation, a cache manager could actually apply smarter transformations. Instead of specifying whether `request2` invalidates `request1`, a transformation could modify the cached response for `request1` to conform to the changes requested by `request2`. For example, when a `delete` request is received, a smart cache manager might modify the cached responses of earlier query requests by removing information.

User experience. An important criterion for evaluating a good technique to support disconnections is its effect on the user's experience. Ideally, a user should have the same experience when disconnected as when connected. Achieving the ideal goal, however, is not practically feasible, especially if the Web client is unaware of the existence of the cache.

There is a direct trade-off between the consistency guarantees assured by the Web-service cache and the quality of the user experience during disconnections. By providing only weak consistency guarantees, the Web-service cache can greatly improve the availability of Web services. For example, when the cache handles a request for an update operation, in addition to storing that request for future playback to the server, it could send a fake response to the Web client reporting the success of this operation. When the request is actually played back to the server upon reconnection, however, the server may decide to abort that operation for various reasons. On the other hand, guaranteeing strong consistency would not affect a user's experience but would prohibit the cache from employing certain techniques to enhance the service's availability.

Making the Web client application cache-aware can aid users during offline access to Web services. In particular, the Web-service client could appraise the user about disconnections and uncertainties in the execution of certain operations. Modifying Web clients to add this

reporting functionality may not be an easy task, however. Our experiment suggested the need to discover a standard mechanism for reporting important events to users that did not require substantial alteration of the Web client implementation.

One approach is for the cache manager to add an optional cache header to the SOAP responses. The cache header can be completely ignored by unmodified Web-service clients. Cache-aware Web clients can use the information provided by the cache header, for example, to pop up a window informing the user that the response was returned from the cache or that the request was stored in the playback queue to be communicated later.

Request and response messages. Understanding the format of the messages exchanged between a Web-service client and server can be another problem for a caching proxy. Despite using standard protocols, such as SOAP, Web services deviate considerably in the structure of their request and response messages. Even mechanisms for identifying the name of the operation being performed vary from service to service. For example, in Figure 1, which shows a SOAP request for the MyContacts Web service, the operation name, `insert`, is one of the attribute fields of the request header. The operation name may have to be identified in a completely different way for a different Web service.

Correct comprehension of the message structure is also required for other fundamental reasons such as comparing requests and sending default responses. In general, a cache manager needs to understand which elements of a request message should be used as the key for cache lookups. For example, each SOAP request message of the MyContacts Web service has a unique identifier in one of the header fields (see the `id` field in Figure 1). The cache manager must ignore this field during comparisons so it can correctly recognize similar requests. Otherwise, every request would be different and the cache would be rendered ineffective.

When a cache receives a request for an update operation during disconnection, it is expected to return a meaningful response to the client to pretend that the service is available. If this operation is also cacheable, the cache can return a response stored earlier; if not, it needs to generate a response that conforms to the message format of that service. Current WSDL specifications contain enough information to permit the cache manager to fabricate a properly formatted response, but lack information about reasonable default values for each element of the response. For example, in the response from an

insert request on the MyContacts Web service, the cache manager would need to set the status attribute to “success” to prevent the client from reporting a failure. The cache manager also needs to copy the value of the unique identifier from the request message to the **relatesTo** field of the response.

Prefetching. The effectiveness of a cache depends on the similarity of future requests to past requests. The cache can return stored responses only for those requests seen earlier by the cache. Prefetching or “hoarding” techniques that preload the cache with responses of requests that are anticipated in the future can significantly improve availability. Selecting the right requests for hoarding, however, requires the involvement of the user. Developing a standard mechanism for users to specify hoard requests that can be used by the Web-service cache is a challenge.

Security. Security is another important issue to be considered while building a cache for Web services. Web services often check the authentication of the messages and the authority of the users before performing operations. For example, the expiration of a Kerberos security-system ticket might prohibit a user from accessing certain information. The cache manager may need to include mechanisms to perform these authorization checks before responding from the cache during disconnections. Unfortunately, Web services use several disparate methods for ensuring security, thereby making it difficult to incorporate security in a cache implementation.

THE ULTIMATE GOAL: SEAMLESS ONLINE- OFFLINE TRANSITIONS

Mobile devices with wireless networking capability, such as laptops, handheld PDAs, and smartphones, are an increasingly popular platform for interacting over the Internet. Future applications on such devices will need highly available access to XML Web services, which are emerging as the key technology for integrating existing systems and providing new platform-independent services. The challenge faced by application designers is how to support mobile devices that have imperfect network connectivity, while incorporating Web services that reside on Internet

servers, export a rich set of operations, and are provided by a diversity of organizations.

Caching on mobile devices is needed to accommodate low-bandwidth, high-latency connections and to permit continued operation during periods of disconnection. Caching to improve availability of file systems and database systems is a well-explored and widely used technique. Web services differ considerably from these traditional distributed systems, however. Both file systems and database systems have well-defined client interfaces. For example, a file server exports standard operations such as **read**, **write**, **open**, and **close** that a cache manager can

implement. In contrast, each Web service exposes its own distinct interface to clients.

To deal with the diversity of Web-service operations, HTTP proxies can transparently cache request and response packets sent between applications and

Web services. Similar requests performed while disconnected can then be serviced with cached responses from the proxy. The proxy also needs to queue up certain operations performed while disconnected so that they can be later executed on the appropriate Web services when connectivity is restored. Using a proxy-based scheme allows caching to be performed transparently when accessing Web services designed with no support for client-side caching.

An alternative to request-response caching is to replicate some or all of a Web service to the mobile device. Many Web services are front ends to SQL databases, and technologies exist for keeping device-resident data synchronized with server-resident databases. Maintaining a database on the mobile device has advantages over request-response caching in that users can have more control over the set of data available during disconnection and, more importantly, applications can run arbitrary queries on the local data.

Simply replicating Web-service data, however, is insufficient. Web services encapsulate their data in critical business logic. Indeed, the attraction of Web services is in allowing client applications to leverage this high-level logic. Replicating server code onto mobile devices with limited resources may not be feasible. More importantly, a Web service’s code is often treated as proprietary and is generally owned by parties different from those accessing the services. Thus, although data and code replication

Caching XML Web Services for Mobility

may work for closely coupled applications and services, it is not as generally useful as caching at the SOAP level.

Cache managers can be built directly into applications, rather than residing in proxies. This approach allows better integration with the application. For example, an application may know how to prefetch data for offline use and might provide visual cues as to what data is accessible when disconnected. Although handcrafted cache managers may work well for applications that access a small number of stable Web services, the drawbacks, of course, are the development costs and the need to evolve the caching code as the services themselves invariably evolve.

Our experiment with Microsoft's .NET My Services demonstrates that, using a caching proxy, you can achieve offline access with applications that were built without regard for mobility. That is, cache managers can be deployed transparently between client applications and servers without alterations to the Web-service implementations and communication protocols. Transparent, service-independent caching of all SOAP requests and responses works well for the large class of Web services that provide query operations only on fairly static data, such as mapping, currency conversion, and yellow page services. Nevertheless, especially for Web services that allow applications to update shared data, our experiment indicated that the Web-service cache could be more effective given a reasonable understanding of the semantics of the Web services.

Extensions to the WSDL standard are needed to customize proxy-based or application-embedded cache managers based on the semantics of specific Web services. Additional information placed in a Web service's WSDL description could indicate which operations are updates and which are cacheable, thereby increasing the effectiveness of caching schemes. For example, one might annotate the WSDL operation elements, which are used

to specify the request and response formats for each operation exported by a Web service. Such annotations extend the description of the Web service's interface. These annotations would not affect tools that automatically generate Web-service clients from WSDL specifications, but are simply used to adjust the behavior of cache managers. Annotations can be added to the WSDL description without requiring any modifications to the implementation of the Web service. These annotations could either be published by the service provider or by a third-party provider reasonably aware of the semantics of the Web service. The full set of WSDL extensions needed to facilitate caching remains to be explored and standardized. Extended WSDL specs would allow client-side caches to provide better cache consistency and, hence, a more satisfying mobile user experience.

The designers of future Web services should produce extended WSDL specifications that allow effective caching on both mobile and stationary devices for both disconnected operation and increased performance. Development tools driven by WSDL specs could then aid in the construction of mobile applications, a notoriously difficult endeavor. The ultimate goal is to provide seamless offline-online transitions for users of mobile applications that use emerging Web services. Q

DOUG TERRY is a senior researcher at the Microsoft Research Silicon Valley Lab. He is working on new technology in support of mobile users and architectures for accessing Web services from intermittently connected devices. Prior to joining Microsoft, Terry was the founder and CTO of Cogenia, which provided a replication platform for delivering contextually relevant information to mobile devices. Before Cogenia, he was chief scientist of Xerox PARC's Computer Science Laboratory, where he helped pioneer the notion of ubiquitous computing and led a number of research projects on weakly-consistent distributed systems. Terry has a Ph.D. in computer science from the University of California at Berkeley, where he worked on Berkeley Unix, developed the first version of the BIND DNS server, and later taught courses as an adjunct faculty member.

VENUGOPALAN RAMASUBRAMANIAN is a Ph.D. student in computer science at Cornell University, where he is working in the field of mobile computing and wireless networks. He was an intern in 2002 with Microsoft Research Silicon Valley Lab.

© 2003 ACM 1542-7730/03/0500 \$5.00

BIO