Latency Performance of SOAP Implementations

Dan Davis † and Manish Parashar ‡
† Compaq Computer Corporation, Manalapan, NJ 07726, USA
‡ Department of Electrical and Computer Engineering
Center for Advanced Information Processing (CAIP)
Rutgers, The State University of New Jersey
Piscataway, NJ 08855-1390, USA
{dand,parashar}@caip.rutgers.edu

Abstract— **This paper presents an experimental evaluation of the latency performance of several implementations of Simple Object Access Protocol (SOAP) operating over HTTP, and compares these results with the performance of JavaRMI, CORBA, HTTP, and with the TCP setup time. SOAP is an XML based protocol that supports RPC and message semantics. While SOAP has been designed as an interoperable business-to-business protocol usable over the Internet, we believe that applications will also use SOAP for interactive web applications running within an intranet. The objective of this paper is to identify the sources of inefficiency in the current implementations of SOAP and discuss changes that can improve their performance. SOAP implementations studied include Microsoft SOAP Toolkit, the SOAP::Lite Perl module, and Apache SOAP.**

Keywords—**Performance study, SOAP, JavaRMI, CORBA, Network programming**

I. Introduction

Remote Procedure Call (RPC) and Remote Method Invocation (RMI) provide elegant and powerful models for programming distributed systems. These programming models typically include a protocol to exchange information, a language to describe an application's interface, and bindings that preserve the syntax of intra-computer communication for inter-computer communication.

The Simple Object Access Protocol (SOAP) and Web Service Description Language (WSDL) are a new protocol and an interface language that provide these same benefits for web services and web applications in peer-to-peer systems [1][2]. These technologies are capable of supporting a web service infrastructure [4] where SOAP accessible services are described using WSDL and discovered through service registries. In such an infrastructure, Web services can be directly accessed and can be combined and composed. For example, the practice of screen scraping [5], where relevant information is extracted from a web page and is reformatted and passed to another web page, can be avoided using these technologies. One service can directly invoke another to produce composite results.

SOAP and WSDL are increasingly accepted as the means for supporting distributed applications on the web. This has made the performance of SOAP critical. As SOAP will be used to support evolving interactive web applications, it should be fast enough for productive human computer interaction. Furthermore, web services will also be deployed on intranets where there are fewer hops and firewalls between the web service producing information and the consumer of that information.

In this paper, we analyze the latency performance of several SOAP implementations and compare these with results for JavaRMI, CORBA, TCP setup delay, and HTTP services using Microsoft Internet Information Services (IIS) and Apache Tomcat. Previous studies have shown the impact of XML parsing and formatting on SOAP performance [3]. The evaluation presented in this paper supports these results and identifies additional factors at the network level that can have a significant impact on SOAP performance. Based on the results, the paper discusses strategies in network programming that will improve the performance of Microsoft SOAP Toolkit, Perl SOAP::Lite, and Apache SOAP.

II. Protocol Overview

SOAP is a lightweight protocol for exchange of information in a decentralized, distributed environment. It is an XML based protocol that consists of three parts: an envelope that defines a framework for describing what is in a message and how to process it, a set of encoding rules for expressing instances of application-defined datatypes, and a convention for representing remote procedure calls and responses.

While SOAP doesn't specify a transport mechanism, most SOAP RPC implementations use HTTP. The SOAP request is the body of an HTTP POST request; the response is the body of the HTTP response. Figure 1 shows a request sent to a SOAP service. The trace was generated using Apache SOAP with the Tomcat application server.

WSDL provides an interface definition language for SOAP. WSDL is not required, but the use of WSDL with SOAP is a de facto standard. WSDL supports the definition of complex structures. WSDL defines the request and response messages for a number of ports, which correspond to methods for an RPC service. The definition of a message specifies the names and XML types for each parameter. WSDL can also describe messages and documents for non-RPC SOAP. SOAP implementations use WSDL to dynamically bind to web services. However, SOAP implementations differ in their support for binding to application-defined datatypes. In general, associating high-level structures is not as easy for the programmer as with JavaRMI or CORBA.

```
POST /soap/servlet/rpcrouter HTTP/1.0
Host:  192.168.1.1
Content-Type:  text/xml; charset=utf-8
Content-Length:  408
SOAPAction:  "urn:test:soap"
Cookie:  JSESSIONID=hypwiuar12
Cookie2:  JSESSIONID=hypwiuar12

<?xml version='1.0' encoding='UTF-8'?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=
"http://schemas.xmlsoap.org/soap/envelope/"
xmlns:xsi="http://www.w3.org/1999/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/1999/XMLSchema">
<SOAP-ENV:Body>
<ns1:getIntegers xmlns:ns1="urn:test:soap"
SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/
soap/encoding/">
</ns1:getIntegers>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Fig. 1. HTTP POST request with SOAP request

III. SOAP Implementations and Related Work

Several implementations of SOAP are maturing rapidly. Web sites facilitate interoperability testing and provide tutorials for rapid learning [6][7]. Implementations differ in their support for class binding, ease of use, and performance.

A. Apache SOAP and Apache Axis

Apache SOAP was developed by IBM alphaWorks and donated to the Apache Software Foundation [8]. It provides SOAP support for Apache's Tomcat application server. Apache Axis is planned to replace Apache SOAP. To increase performance, Apache Axis uses SAX for XML parsing and processes messages in stages.

B. SoapRMI

Indiana University's Extreme Laboratory wrote SoapRMI (now called XSOAP) to study how SOAP might be applied for high performance technical computing [9]. They compared the performance of SOAP with JavaRMI, Nexus RMI, and HPC++ and also analyzed the performance limitations of SOAP. They found that the encoding and decoding time for SOAP was greater than for protocols such as Java's object serialization. SoapRMI uses their own parser, the XML Pull Parser (XPP), to improve performance on large arrays and data structures. SoapRMI excels because the interface is nearly identical with JavaRMI. XSOAP 1.3 plans to increase performance using HTTP keep-alive (persistent connections) and chunked encoding.

C. SOAP::Lite module for Perl

Paul Kulchenko has written this module so Perl may be used for a web service server or client [10]. A server may be run as a stand-alone HTTP server, as a CGI script, or as an Apache module through `mod_perl`. Both the server and client support non-basic types using Perl's associative arrays.

D. Microsoft SOAP Toolkit

Microsoft SOAP Toolkit provides a wizard that exports the methods of a COM object using SOAP [11]. The COM objects may be written using Visual C++ or Visual Basic. The wizard generates deployment files including the Web Service Description Language (WSDL) file that defines the interface.

E. JavaRMI and CORBA

JavaRMI and CORBA should be compared with SOAP because Java makes it easy to invoke these protocols from Java Applets. JavaRMI's advanced features include code distribution, graph serialization, and distributed garbage collection [12]. CORBA is designed for interoperability and a model-oriented software process [13].

F. Peer-to-peer performance studies

Web services are often associated with peer-to-peer software. The composition expected of web services resembles the forwarding common to Gnutella and Freenet peers. The performance studies of peer-to-peer small world networks suggest that if such a model evolves for web services, then the network of composite web services is likely to have short paths [5].

IV. Experimental Design

For each protocol or SOAP implementation we tested, we implemented a remotely accessible server with this interface:

- `void doNothing()` – Our tests call this method to determine the overhead associated with a SOAP call.
- `void setSize(int size)` – Our tests call this method to set the size of the string and array returned by `getString()` and `getIntegers()`. The time required for this call is not measured.
- `String getString()` – Our tests call this method because the encoded response to each `getString()` call has the same XML elements regardless of the size of the string returned.
- `int[] getIntegers()` – Our tests call this method so we can compare the latency performance as additional XML elements are encoded and decoded.

Each protocol or SOAP implementation is tested as a black box. This allows our approach to generalize to commercial implementations such as Microsoft SOAP Toolkit, but also limits our ability to analyze the precise cause of performance problems.

Each client implementation makes a tunable number of calls to one of the timed functions, `doNothing()`, `getString()`, or `getIntegers()`. The client prints the time required for these calls excluding a warm-up period and one-time protocol costs. So, for CORBA and RMI, the

times we report exclude the lookup time in the ORB registry. Similarly, for Apache SOAP, the times we report exclude the setup time to create the objects that correspond to each method. This experiment uses a single client to access the server.

These protocols and SOAP implementations are included in all tests:

- **JavaRMI** – The client and server are written in Java and compiled and run with Sun's JDK 1.3.1_02 for Microsoft Windows.
- **CORBA** – The client and server are written in Java and compiled and run with Sun's JDK 1.3.1_02 for Microsoft Windows.
- **Microsoft SOAP Toolkit SP2** – The client and server are written in Visual Basic with Visual Studio 6.0. The client uses the Win32 call `GetTickCount()` to get a timer with comparable millisecond resolution to Java's `System.currentTimeMillis()`. We used an ASP listener for these tests instead of an ISAPI listener. Microsoft Internet Information Services (IIS) provided the web server.
- **SoapRMI/Java 1.1** – The client and sever are written in Java and compiled with Sun's JDK 1.3.1_02. SoapRMI can be run with a registry or with an HTTP endpoint for interoperability. Tests performed for normalization show little difference between these two options.
- **SOAP::Lite module in Perl** – The client and server were written using ActiveState distribution of Perl 5.6.0
- **Apache SOAP 2.2 and Axis Alpha 3** – The client and server were written in Java and compiled and run with Sun's JDK 1.3.1_02 for Microsoft Windows. Tomcat 4.0.2 provided the web server; xerces and xalan are included with Tomcat 4.0.2 for XML parsing.

We performed each test with both client and server on the same host, and then repeated the test with separate client and server hosts joined by a small LAN. The LAN was a 10 Mbps LAN with only these two hosts. For the client machine, we used a laptop with a Pentium-III 550 MHz processor and 256 MB RAM, running Windows 2000 Professional. For the server, we used a desktop with an AMD Athlon 700 MHz processor and 160 MB RAM, running Windows 2000 Server. The desktop was used for both client and server when both client and server were on the same host.

Since each time measurement represents 200 calls and the timer's precision is 10 ms, the precision of our measurements is +-0.05 milliseconds.

A. void doNothing()

In order to seperate the overhead contributed by TCP, HTTP, and SOAP, we added these experiments to the `doNothing()` case:

- **TCP setup time** – How long does it take in our environment to connect and then close a TCP connection using Java?
- **HTTP to Apache Tomcat** – How long does it take to get the response to a Servlet request of Tomcat? Tomcat is the Web Server for Apache SOAP.

TABLE I
DONOTHING() WITH SERVER AND CLIENT ON SAME HOST

System	Language	Latency (ms)
JavaRMI	Java	1.2
CORBA	Java	1.5
MS SOAP Toolkit	Visual Basic	16.8
SoapRMI	Java	19.5
SOAP::Lite	Perl	42.0
Apache SOAP	Java	23.4
Apache Axis	Java	15.6
TCP setup time	Java	1.9
HTTP to Tomcat	Java	3.4
HTTP to IIS	-	1.1

- **HTTP to Microsoft IIS** – How long does it take to get a cached file from Microsoft Internet Information Services (IIS)? Microsoft IIS is the web server for Microsoft SOAP Toolkit.

Just as with the other experiments, we repeated these tests with the client and server on the same host and with separate client and server hosts.

B. String getString()

Tests were performed for strings of length 200, 400, and 800. Since the strings contain no XML tags, parsing the message is independent of the string length. We expected the latency performance with each of these lengths to reflect additional transfer time. Since all of these are under the minimum packet size of the network, we expected the `getString()` results to be similar to the `doNothing()` results.

C. int[] getIntegers()

Tests were performed for integers arrays containing 200, 400, and 800 integers. Because Visual Basic is unable to return arrays, the Visual Basic of `getIntegers()` returns an XML node list designed to produce responses of similar size and complexity. Unfortunately, that makes the `getIntegers()` results not quite an apples to apples comparison. To characterize how much difference this makes, we include an alternate implementation of `getIntegers()` for Apache SOAP that returns the same XML node list.

V. Experimental Results

When the client and server are run on separate hosts, SOAP performs very poorly. Table II presents results for the `doNothing()` call when the the client and server run on separate hosts. The network delay is calculted by comparing identical rows in Table I and II. TCP connection setup is about the same between the client and the server, yet Apache SOAP, the Microsoft SOAP Toolkit, and the SOAP::Lite Perl module all take roughly 200 milliseconds. Since this time is so surprising and so constant, we expected this was due to something other than SOAP processing.

TABLE II
doNothing() with separate client and server

System	Language	Latency (ms)
JavaRMI	Java	0.8
CORBA	Java	1.3
MS SOAP Toolkit	Visual Basic	200.9
SoapRMI	Java	37.7
SOAP::Lite	Perl	200.1
Apache SOAP	Java	205.9
Apache Axis	Java	13.1
TCP setup time	Java	1.2
HTTP to Tomcat	Java	2.6
HTTP to IIS	-	1.5

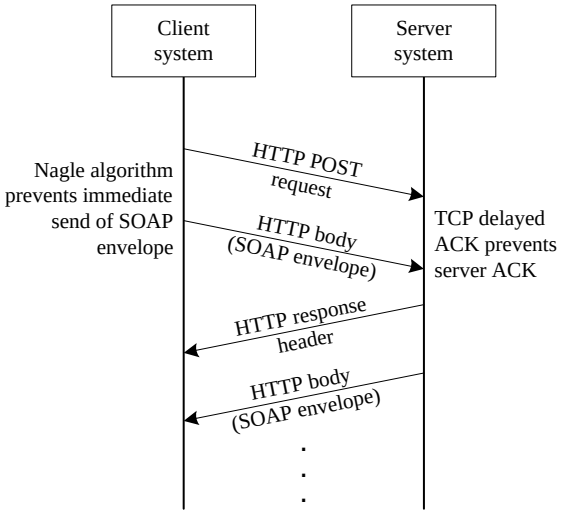

Fig. 2. Event trace of SOAP communication

To explain the delay, we compared network traces of the communication for Apache SOAP and SoapRMI. Figure 2 shows the network packets exchanged for Apache SOAP. We found that the HTTP POST request was divided into two packets, one containing the HTTP headers and one containing the HTTP body including the SOAP envelope. For Apache SOAP, the second packet in the HTTP request was delayed about 170 milliseconds from the first request. Once the second packet is received, the server processes the SOAP call. Then, at least two packets are sent for the HTTP response, one containing the headers and additional packets for the body with the SOAP envelope. The delay in the request is caused by the interaction between the Nagle algorithm and TCP delayed ACK algorithm in the operating systems for the client and server. The Nagle algorithm is controlled using the `TCP_NDELAY` socket option [14]. Both the Nagle algorithm and TCP delayed ACK algorithm are designed to reduce the number of small network packets from telnet like applications.

Microsoft SOAP Toolkit has a different pattern of network communication, shown in Figure 3. The Nagle algorithm and delayed ACK cause the same delay, but this time, the server waits for the client's ACK. The HTTP POST request and body are sent in the first packet after the TCP connection is built. Microsoft SOAP Toolkit sends an HTTP response with continue status (status code 100) before sending the HTTP message with the SOAP response. The purpose of the added continue is to reduce network communication [15]. If the server will reject an HTTP request, it can do so before receiving the body of the request. In this case, the header and body are in the same packet, and so the reason for the continue status is unclear. The server doesn't send the SOAP response until after the client has acknowledged the HTTP continuation packet. In our tests, the client has no more data to send, and so the TCP delayed ACK algorithm applies.

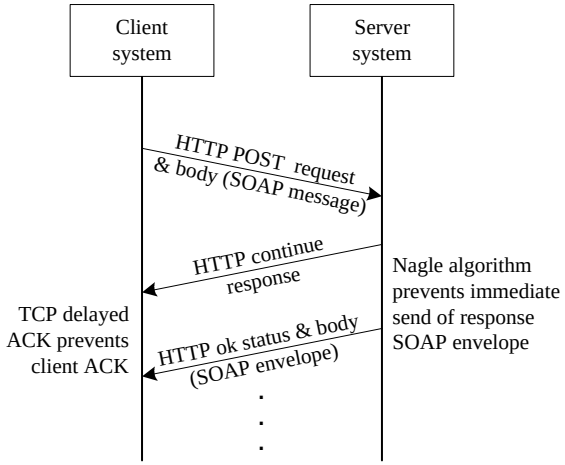

Fig. 3. Event trace of Microsoft SOAP communication.

TABLE III
getString() with server and client on same host

System	char[200] Latency (ms)	char[400] Latency (ms)	char[800] Latency (ms)
JavaRMI	1.3	1.2	1.3
CORBA	1.5	1.5	1.6
MS SOAP Toolkit	16.6	23.7	34.8
SoapRMI	19.3	19.6	19.9
SOAP::Lite	42.4	42.3	42.6
Apache SOAP	22.8	24.2	25.1
Apache Axis	17.1	19.0	19.6

The `getString()` results in Tables III and IV provide additional support for this analysis. The responses for `getString()` add one XML element (the result) and one XML attribute (the datatype) relative to `doNothing()`. So, a larger string adds little more XML overhead to a message. The `getString()` results are very similar to the `doNothing()` results.

When the client and server are run on the same host, the network delays are not significant. Under these conditions, XML parsing and formatting is the largest factor of the

TABLE IV
GETSTRING() WITH SEPARATE CLIENT AND SERVER

System	char[200] Latency (ms)	char[400] Latency (ms)	char[800] Latency (ms)
JavaRMI	1.2	1.6	2.3
CORBA	1.7	2.1	2.7
MS SOAP Toolkit	200.2	200.3	200.3
SoapRMI	44.1	41.4	41.6
SOAP::Lite	200.2	200.3	200.3
Apache SOAP	209.9	213.8	237.4
Apache Axis	14.3	14.7	15.6

TABLE V
GETINTEGERS() WITH CLIENT AND SERVER ON SAME HOST

System	int[200] Latency (ms)	int[400] Latency (ms)	int[800] Latency (ms)
JavaRMI	1.5	1.6	1.8
CORBA	1.5	1.9	2.0
MS SOAP Toolkit †	28.1	44.8	76.7
SoapRMI	22.2	24.4	31.5
SOAP::Lite	229.6	493.6	1810.6
Apache SOAP	65.8	117.3	190.7
Apache SOAP †	54.1	73.2	119.0
Apache Axis	73.2	150.2	229.4

TABLE VI
GETINTEGERS() WITH SEPARATE CLIENT AND SERVER

System	int[200] Latency (ms)	int[400] Latency (ms)	int[800] Latency (ms)
JavaRMI	2.3	3.2	4.7
CORBA	2.7	4.0	5.5
MS SOAP Toolkit †	33.3	55.8	104.1
SoapRMI	75.0	73.0	108.6
SOAP::Lite	400.3	600.8	1120.1
Apache SOAP	310.3	380.5	561.7
Apache SOAP †	293.7	285.0	449.2
Apache Axis Alpha 3	76.9	138.6	243.7

† Marked results are for XML node lists.

latency performance. Table I presents the results when the `doNothing()` call is made from a client to a server on the same host. For those results, the client and server are separate processes, and a loop-back socket is used rather than an in-memory call. Since Apache SOAP is implemented as a Tomcat servlet, the No-op Servlet results show what portion of the time for Apache SOAP is due to HTTP processing. Excluding the HTTP processing, SOAP and XML processing represents 20.2 milliseconds, 86% of the time, in an Apache SOAP call that does nothing. Similarly, the Microsoft IIS results in Table II show that Microsoft SOAP Toolkit takes 15.4 milliseconds, 94% of the time, for SOAP and XML processing. These numbers are illustrative, but might be different with a faster processor.

SOAP is also orders of magnitude slower than JavaRMI and CORBA. In Table I, The performance of JavaRMI and CORBA are comparable, but SoapRMI, Apache SOAP, SOAP::Lite, and Microsoft SOAP Toolkit latency are much worse.

There are surprises when we compare the `getIntegers()` performance of the different implementations. Microsoft SOAP Toolkit's performance is exceptionally good when the client and server are on different hosts. In Table V, Microsoft SOAP Toolkit scales well, but is not far out of line with other results. In Table VI, Microsoft SOAP Toolkit is much better than we'd expect based on the network delays due to the Nagle algorithm.

The network delays we analyzed for the `doNothing()` call don't seem to apply to Microsoft SOAP Toolkit's `getIntegers()` performance. Microsoft SOAP Toolkit benefits because there is no translation from an XML node list to an integer array representation, but this doesn't explain the differences seen in Table VI. The Apache SOAP results for an XML element list demonstrates there's still a wide margin between Microsoft SOAP Toolkit and Apache SOAP. Analysis of the network communication confirms that the delays mentioned above are missing between packets. Suppose the response to a call for 200 integers writes enough data into the TCP connection's send buffer to override the Nagle algorithm on the server. If this is so, then the performance should degrade for an array whose XML format isn't as large. A simple experiment with a 20 integer array confirms that the latency per call is roughly 200 milliseconds for a 20 integer array. So, if the messages are large enough, the network delay is removed. Since an 800 integer array is roughly 30 KB for both Apache SOAP and Microsoft SOAP Toolkit, why doesn't this apply to Apache SOAP? For Apache SOAP, the Nagle algorithm applies to the HTTP POST request containing the call rather than the response

Another surprise with the `getIntegers()` results is the poor performance of the SOAP::Lite Perl module. The network packets generated by Perl show that Perl's integer array representation is much larger than that of Apache SOAP or Microsoft SOAP Toolkit. However, analysis of the representation shows that there are about the same number of XML elements, attributes, and namespaces. Each XML element representng an item in the array is named automatically and differently for SOAP::Lite, for instance, `gensym4819` is one element. Axis and Apache use an element named `item` for every integer in the array. SOAP::Lite automatically serializes associative arrays where the same object may be referenced more than once. My guess is that while this is very convenient, it has a large impact on performance. Another thing to remember is that Perl is a purely interpreted language. In contrast, Java's hotspot virtual machine may optimize the tight loops that perform our tests during the warming period.

Apache Axis is written for better performance, but that doesn't show in the `getIntegers()` latency. Axis uses a pipe-lined approach to SOAP processing, so it isn't surprising that latency suffers for larger messages. A pipe-lined architecture improves scalability and throughput.

SoapRMI was written to improve the performance for tests such as `getIntegers()`. So, it isn't surprising how well SoapRMI scales. SoapRMI scales well because it was designed to scale to large structures. SoapRMI also avoids the network delays that we see for other implementations. However, SoapRMI's performance is comparable with Apache SOAP in Table I, and both Apache SOAP and SoapRMI are orders of magnitude slower than JavaRMI. As discussed earlier, this is due to the overhead of SOAP and not due to the use of HTTP.

All of the implementations set the HTTP `Content-Length` header in their response. This header is difficult to pre-determine for dynamic content. To calculate the `Content-Length`, SOAP implementations may buffer the data. Such buffering may cause delays. One option to avoid buffering is to use HTTP chunking[15] as planned for XSOAP 1.3.

VI. Discussion

SOAP shows great potential for simplifying web service composition and the distribution of software using the Internet. Within Corporate Intranets, consolidated web applications and services also require fast, convenient protocols. Regardless of performance, business and system management concerns may make SOAP attractive for these applications. Therefore, SOAP's performance can and should be improved through designs for faster parsing and better network programming.

Binary XML encodings may also improve SOAP. Microsoft .NET uses a proprietary binary encoding for SOAP communication when both client and server understand that protocol.

For web applications operating in both WAN and LAN environments, there's no silver bullet. Web applications operating over a WAN would benefit from a limited number of larger messages, as long as code download doesn't take too long. For web applications operating over a LAN, many small messages would lead to smaller client applications.

For web services, JavaRMI may be encapsulated within HTTP or routed through firewalls using protocols like SOCKS. Since SOAP's overhead is greater than JavaRMI even discounting HTTP overhead, we expect that JavaRMI or CORBA will be faster over HTTP as well.

If a faster RMI system can provide the system management and programming benefits of SOAP, including interoperability, the ability to transfer XML DOM data, and the ability to call through firewalls, then such an RMI system should be preferred until SOAP matures.

VII. Conclusions

Our analysis points to areas where SOAP performance can be improved without changing the HTTP protocol. One large source of inefficiency in SOAP is the use of multiple system calls to send one logical message. Another source of inefficiency in SOAP is the XML parsing and formatting time. SOAP's performance may also be improved by using different capabilities of the HTTP protocol such as HTTP chunking.

References

[1] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer, "Simple Object Access Protocol (SOAP) 1.1," World Wide Web Consortium (W3C), May 2000, http://www.w3.org/TR/2000/NOTE-SOAP-20000508, visited Feb. 2001.

[2] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," World Wide Web Consortium (W3C), Mar. 2001, http://www.w3.org/TR/2001/NOTE-wsdl-20010315, visited Sep. 2001.

[3] Madhusudhan Govindaraju, Aleksander Slominski, Venkatesh Choppella, Randall Bramley, and Dennis Gannon, "Requirements for and Evaluation of RMI Protocols for Scientific Computing," in *Supercomputing*. 2000, IEEE.

[4] Williams Bordes and Johann Dumser, "SOAP: Simple Object Access Protocol," *TechMetrix Research-Trendmarkers e-Newsletter*, Dec. 2000, http://www.techmetrix.com/trendmarkers/tmk1200/tmk1200-3.php3, visited Jan. 2001.

[5] Theodore Hong, ed. Andy Oram, *Peer-To-Peer: Harnessing the Power of Disruptive Technologies*, ch. 14, OReilly and Associates, Inc., 2001.

[6] "A Quick-Start Guide for Installing Apache SOAP," *XMethods*, Feb. 2001, http://www.xmethods.com/gettingstarted/apache.html, visited Feb. 2001.

[7] James Snell, "Exposing Application Services with SOAP," *XML.COM*, July 2000, http://www.xml.com/pub/a/2000/07/12/soap/mssoaptutorial.html, visited Nov. 2000.

[8] Apache Software Foundation, http://xml.apache.org/, visited Feb. 2001.

[9] University of Indiana, Extreme Lab, http://www.extreme.indiana.edu/soap, visited Feb. 2001.

[10] Paul Kulchenko, SOAP::Lite Perl module, http://www.soaplite.com/, visited Feb. 2001.

[11] Rob Caron, "Develop a Web Service: Up and Running with the SOAP Toolkit for Visual Studio," *MSDN Magazine*, Aug. 2000, http://msdn.microsoft.com/library/, visited Feb. 2001.

[12] "Java Remote Method Invocation (RMI) Specification", Sun Microsystems, http://java.sun.com/products/jdk/rmi/, visited Dec. 2000.

[13] "The Common Object Request Broker: Architecture and specification" rev 2.0, Object Management Group, Feb. 2000, http://www.omg.org/, visited Feb. 2001.

[14] W. Richard Stevens, *UNIX Network Programming: Networking APIs (Second Edition)*, vol. 1, Prentice Hall, 1997.

[15] "Hypertext Transfer Protocol - HTTP/1.1," RFC 2616, Internet Engineering Taskforce (IETF), http://www.ietf.com/, visited Dec. 2000.