A Comparative Study of DCOM and SOAP

Alexander Davis
Intel Corporation
1900 Prairie City Road
Folsom, CA 95630
Alek.davis@intel.com

Du Zhang
Department of Computer Science
California State University
Sacramento, CA 95819-6021
zhangd@ecs.csus.edu

Abstract

Simple Object Access Protocol (SOAP) is a recent technology, which aims at replacing traditional methods of remote communications, such as RPC-based Distributed Component Object Model (DCOM). Being designed with a goal of increasing interoperability among wide range of programs and environment, SOAP allows applications written in different languages and deployed on different platforms to communicate with each other over the network. While SOAP offers obvious benefits in the world of interoperability, it comes at a price of performance degradation and additional development efforts required for implementation of features missing from SOAP, such as security and state management. This paper reports the outcome of a comparative study of SOAP and DCOM in terms of features, development effort, and application performance. The results indicate that SOAP performance on a homogeneous platform is consistently worse than that of DCOM. Depending on the operation and the amount of data transferred, SOAP performance degradation can range from minor (e.g. two or three times slower than DCOM) to major (e.g. twenty or more times slower). Based on lines of code statistics, SOAP application development effort should be slightly (~10%) higher. If an application has to rely on the features such as state management or security, which are missing from SOAP, then SOAP implementation will require substantial design efforts and in certain cases, may not be feasible at all. We recommend application developers to consider these issues, when preparing to move their applications to SOAP.

Keywords: *distributed computing, DCOM, SOAP, performance analysis.*

1. Introduction

Remote communication allows applications running on different computers to interact with each other across the network. Several remote communication technologies have been available to application developers since the introduction of distributed computing. These technologies include RPC [9], DCOM [11], CORBA [8], EJB [17], and the others.

In this paper, we consider DCOM as a mature remote communication technology. As an evolution of RPC-style programming, DCOM has become a predominant methodology in distributed computing on Windows platform. Even though it makes programmers' job somewhat easier by hiding many complexities of client-server application development, DCOM fails in two areas. First, although DCOM has been ported to other platforms, it has only achieved broad reach on Windows [2]. Second, DCOM applications are difficult to deploy in a corporate environment where communication must be performed across firewalls [26]. Though there is a way to overcome this limitation, the solution is regarded as weakening security and normally not accepted by corporate information security and network administrators. These problems are addressed by a new framework, which is called Simple Object Access Protocol [34], or SOAP.

SOAP was designed with a goal of increasing interoperability among wide range of programs and environments. It allows applications written in different languages and deployed on different platforms to communicate with each other over the network without the need of using custom-written adapters. Trying not to invent a new technology, the architects of SOAP based it on two common protocols: HTTP [23] and XML [18]. SOAP treats XML as an encoding scheme for request and response parameters of the method calls and uses HTTP as a transport layer [2]. Since its introduction in September of 1999, SOAP has been endorsed by the vast majority of hi-tech companies including Microsoft, IBM, Oracle, and Sun Microsystems and is rapidly becoming an industry standard.

Ironically, SOAP interoperability features, which lead to its wide adoption, have a potential to backfire. First of all, unlike platform-specific technologies (RPC, DCOM, CORBA, etc), which transfer data in binary formats, SOAP uses platform-neutral XML formats to encode information passed to and returned from remote method calls. This means that parameters converted to XML strings will bloat data payloads and require additional processor cycles to be spent on XML parsing. Also, since SOAP objects are intrinsically stateless, SOAP applications relying on state information will have to perform additional steps to implement state management. It would be reasonable to assume that these factors can adversely affect SOAP performance.

Although abundance of information about SOAP is widely available in numerous magazine articles, on-line resources, and technical reviews, these publications mostly focus on SOAP benefits. There is no research data publicly available, at the time of this writing, which would assess SOAP performance.

To fill the existing gap, this study aims at comparing SOAP performance to DCOM and determining whether a SOAP-based application is slower than a compatible DCOM-based application. In addition, this research will attempt to compare development efforts required to implement the same functionality in SOAP and DCOM-based applications. Based on these results, this paper will set the expectations (in terms of changes in application performance and required development efforts) for application developers planning on converting their DCOM-based programs to SOAP-based, and provide relevant recommendations.

We use two Web-based, database-driven applications in out study, both are written in the same languages and implement the same sets of functionality. One application uses SOAP as a means of remote communication, while the other uses DCOM. To compare application performance, we conduct a number of compatible load tests. To measure the development effort, we adopt the number of lines of code required to implement the same functionality in a DCOM and a SOAP-based application.

Since DCOM is primarily a single-platform technology and SOAP is a multi-platform technology, the comparison between DCOM and SOAP in terms of application performance and development effort is confined, in our study, to single-platform applications. While there are various SOAP implementations from both Microsoft and other vendors, our work only assesses SOAP implementation provided by Microsoft SOAP Toolkit 2.0 [36], and does not make any claims about other SOAP implementations.

The rest of the paper is organized as follows. Section 2 describes the design of our study. Various issues in the tests are discussed in Section 3. Analysis results are summarized in Section 4. Finally Section 5 concludes the paper.

2. Design

We implement a simplified Web-based account management system as a test application, which maintains information about user accounts on the registered servers. There are two versions of this application: one version written using DCOM, the other SOAP.

Data are stored in a relational database and exposed to the middle tier components via a set of stored procedures. DCOM and SOAP COM servers implement business logic allowing communication between the back and front ends. Web clients are written using ASP technology.

Each test application can perform the following operations: add user, update user, delete user, find user, add server type, update server type, delete server type, find server type, add server, update server, delete server, find server, add user account, delete user account, and find user account.

In addition to the test applications, we write a test harness to compare the throughputs of DCOM and SOAP implementations.

To develop our test application and test harness, we utilize the following Microsoft tools and technologies. Database portion of the application is implemented using Microsoft SQL Server 2000. Microsoft ADO (as part of MDAC 2.6) provides data access services. Middle tier components are written as COM objects in Microsoft Visual Basic 6.0. Front end applications are designed as ASPs and deployed in Microsoft IIS 5.0. Microsoft SOAP Toolkit 2.0 provides SOAP development tools and runtime environment. Microsoft XML Parser 3.0 is used for XML manipulation.

Both SOAP and DCOM applications use the same database (AMSDB). Information in the database is manipulated using twenty two stored procedures.

2.1. Middle tier

Business logic of the test application is written as ActiveX DLLs using Microsoft COM technology [7]. There are two sets of COM components being used for DCOM and SOAP, respectively.

To implement business logic using DCOM, we need only one COM server DLL (DcomServer.dll) with a single COM component (DCOM.Server). This COM component is apartment-threaded.

The application logic of DCOM.Server is rather straightforward. Before performing an operation (such as adding a user or getting server records), a client must call Connect method to establish an ADO connection to the database. Connection object is saved as a member variable of DCOM.Server instance. After finishing performing the operations, the client calls Disconnect method to close database connection.

To implement business logic using SOAP, we need two COM server DLLs. First COM server (SoapServer.dll) is similar to DCOM server (DcomServer.dll), but unlike the latter, it encapsulates two COM components (SOAP.Server and SOAP.Session). SOAP.Session implements a pre-defined IHeaderHandler interface, which describes event handlers used to manage session state on the server side (middle tier). A second COM server (SoapClient.dll) is needed to handle session state on the client side (front end), which contains a single COM component (SOAP.Client), also derived from IHeaderHandler interface. All SOAP COM components are apartment-threaded.

Additionally, SOAP implementation of the middle tier requires the following components:

- A WSDL [40] file (AmsSoap.wsdl), identifying the services provided by the server and the set of operations within each service that the server supports. For each of the operations, the WSDL file also describes the format that the client must follow in requesting an operation.
- A WSML file (AmsSoap.wsml), providing mapping of the operations of a service (as described in the WSDL file) to specific methods in the COM object (SOAP.Server). The WSML file determines which COM object to load to service the request for each operation.

SOAP version of the application will also need a SOAP listener handling incoming SOAP requests. SOAP listener can be implemented either using a custom ASP or standard ISAPI [24] filter (soapisapi.dll). Since SOAP Toolkit's documentation claims that ISAPI handler gives better performance, this will be our choice; otherwise, we would also need an additional ASP file (In comparative tests, ISAPI implementation of the SOAP listener proved to be about 10% faster than ASP implementation.).

Because of SOAP.Server's stateless nature, its application logic is more complex than that of DCOM.Server. While DCOM.Server can reuse a single ADO connection for multiple operations performed during the session, SOAP.Server must establish a new connection for each operation. To avoid the need of passing connection information (i.e. name of the SQL server, login ID, and password) to each method call, SOAP.Server implements a Connect method, which is used to verify connection information (by logging into the database) and - if login is successful - saving connection information values in the SOAP.Server's member variables. When SOAP.Server sends results of the operation back to the client, values of these member variables are passed via HTTP headers to the SOAP.Client object, which maintains their lifetime on the client side when SOAP.Server gets out of scope (i.e. between the method calls). When the client makes another SOAP.Server method call, SOAP.Client object passes saved connection information to SOAP.Session object, which is implemented in the same DLL as SOAP.Server object. SOAP.Session object sets the connection information values of the member variables when the new instance of SOAP.Server is created. Before performing a database operation, SOAP.Server detects that it does not have a database connection, so it will establish one using data values obtained from the member variables. While the approach of saving connection string information via HTTP headers works fine in a test application, it must be considered insecure in a real life scenario, because it results in passing sensitive information (i.e. such login IDs and passwords) back and forth in plain text.

2.2. Front end

For both DCOM and SOAP solutions, the client-side applications are implemented using two sets of almost identical ASPs and supporting files. The major difference between the two is that in DCOM case, the ASPs use DCOM.Server component, while SOAP ASPs invoke SOAP.Server.

2.3. Application architecture

After the user submits an HTML [20] form to the ASP, the page processes user inputs and calls appropriate COM object (DCOM.Server or SOAP.Server). The COM object connects to the database server using ADO connection, calls a stored procedure, and receives results as a single record or a recordset. If the stored procedure does not generate a record or recordset, the caller only processes returned result code and/or output parameter(s).

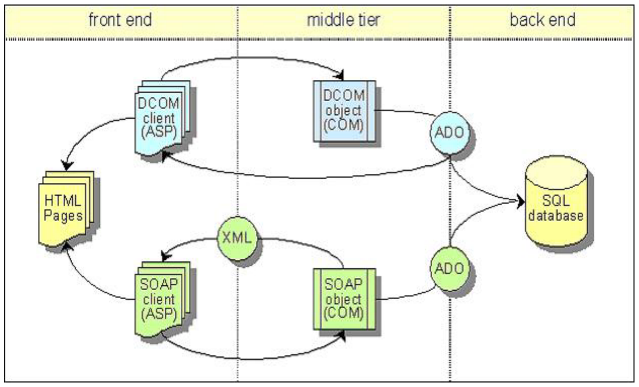

Figure 1. Application architecture.

In DCOM case, the ASP receives the original recordset generated by the stored procedure from DCOM.Server object and processes the returned records.

Unlike DCOM.Server, SOAP.Server cannot just pass the recordset back to the ASP, so it has two choices. In the most common scenario, SOAP.Server transforms the ADO recordset into a serialized XML stream (i.e. XML string) and passes this XML stream back to the ASP [12]. The ASP then creates a disconnected recordset from the received XML string, and processes it exactly as if it were coming from the database server. Alternatively, SOAP.Server can receive XML stream directly from the SQL server, thus bypassing ADO recordset-to-XML string conversion step. Then it would pass XML data to the ASP, which has to process returned records using XPath [41] instead of ADO. In this case, two data transformation steps can be avoided. To distinguish these two approaches we will call them *SOAP with XML/ADO* and *SOAP with XML/XPath*, respectively. Figure 2 describes the data transformations.

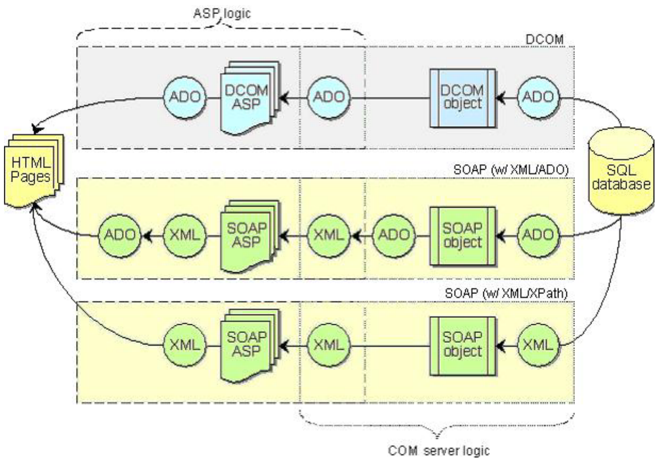

Figure 2. Data transformations.

SOAP method calls. Although there is a number of ways to achieve similar results, this is how SOAP calls are implemented in Microsoft SOAP Toolkit 2.0, which we use in our study (If we decided to implement SOAP calls from scratch, we would have to manually build SOAP packets, unpack results, handle errors, and do all the work, which is already implemented by the SOAP toolkit. In short, we would end up writing a component similar to MSSOAP.SoapClient.).

First of all, the client ASP does not call SOAP.Server object directly. Instead it creates a MSSOAP.SoapClient COM object (which comes with the toolkit) and uses it as a proxy to call SOAP.Server methods. During initialization, MSSOAP.SoapClient takes the URLs of the WSDL and WSML files (AmsSoap.wsdl and AmsSoap.wsml) and retrieves the information describing the service provided by SOAP.Server from these files. This information includes names of the methods, method parameters, return values, etc. When the client makes a SOAP call, MSSOAP.SoapClient passes this call via SOAP runtime to the endpoint defined in the WSML file, which, in our case, is SOAP.Server COM object. Before the call reaches SOAP.Server, it has to go through the Web server (IIS), SOAP listener (soapisapi.dll which comes with SOAP Toolkit), and XML parser. After SOAP.Server processes the operation, results along with the error codes are passed back to the caller passing through the same SOAP runtime components. Figure 3 shows SOAP call stack.

SOAP session state management. Because SOAP objects are intrinsically stateless, we must implement session management logic programmatically. As it is already described earlier, session state information is passed between the client ASP and SOAP.Server using HTTP headers and managed on the client and server sides with the help of two COM objects: SOAP.Client (client side) and SOAP.Session (server side).

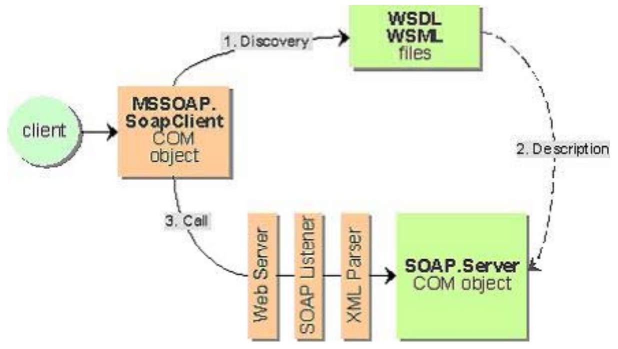

Figure 3. SOAP call sequence.

After creating MSSOAP.SoapClient object, the ASP calls its HeaderHandler method passing an instance of the SOAP.Client component. This instructs MSSOAP.SoapClient to pass HTTP headers through SOAP.Client. In our case, SOAP.Client writes the database server name, SQL login ID, and password, (which are used to connect to the database server) to the HTTP headers. On the server side, these headers are processed by SOAP.Session object, which is defined as a default header handler for SOAP.Server in WSML file. SOAP.Session object retrieves SQL login information from the HTTP headers and assigns them to the corresponding data members of SOAP.Server object, so that SOAP.Server could establish connection to the database serve. Figure 4 illustrates this process in the diagram.

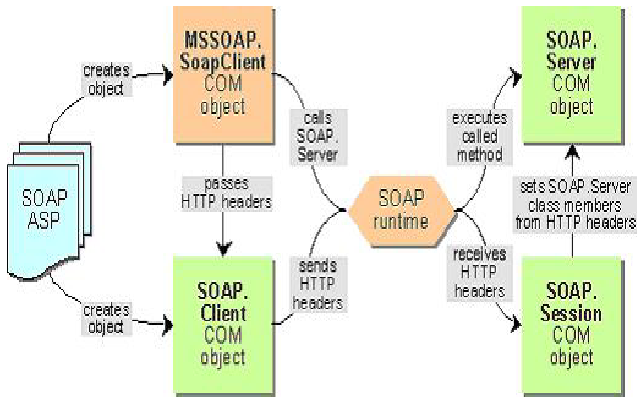

Figure 4. SOAP session state management.

2.4. Test harness

Test harness is used to measure the throughput of DCOM and SOAP implementations of the application for the key operations. It is written using ASP and DHTML [16] technologies. To run a test for a particular operation, the user has to fill out the corresponding HTML form providing details pertaining to the test, such as number of times the operation must be executed, number of records to be returned, etc. After the user submits the form, the ASP executes requested operation on either DCOM.Server or SOAP.Server COM object and returns intermediate page to the user's browser with current operation execution results. Returned page contains HTML form, which auto-submits itself so that the ASP can keep executing the operation a number of times as specified in the test settings. While executing the test, test harness measures the time it takes to perform the operation.

3. Test

Most enterprise applications, which are being or will be converted to SOAP, are most likely to fall under the categories of either traditional or Web-based client-server applications. In a typical three-tier client-server system, information flows from a database (back-end) to a middle tier responsible for business logic implementation and then from the middle tier to a front-end client, which then presents the data to the user. In this scenario, the most common operations regarding transporting data to and from a database can be categorized using the following groups:

1. *Transporting a single record to the database*, i.e. SQL INSERT or UPDATE operations affecting one record. Since INSERT operations normally transfer at least as much data as corresponding UPDATEs, we limit our test cases to INSERT operations. Because the throughput here must be directly correlated to the number of inserted rows (e.g. if it takes M seconds to insert N records, it should take approximately 2×M seconds to insert 2×N records), this category can be represented by a single test case (e.g. inserting a single record N times).
2. *Retrieving a single record from the database*, i.e. SQL SELECT operation affecting one record. For the same reason as in previous group, this test type can also be represented by a single test case (e.g. selecting a single record N times).
3. *Retrieving multiple records from the database*, i.e. SQL SELECT operation affecting more than one record. Unlike previous two categories, the performance of this type of operation may depend on the number of rows returned with each recordset, i.e. time may not be linearly correlated to the number of records returned with each request. Therefore, we have to execute this test with different number of returned records, e.g. 100, 1000, 10000.
4. *Deleting one or more records in the database*. Because SQL DELETE operations normally do not result in massive data transfers, we will not be using them in our tests.

When comparing SOAP to DCOM, we test how long it takes to perform each of the first three types of operations for the same amount of data using either methodology, and determine whether conversion to SOAP is likely to cause performance degradation.

Because there is no difference among database operations of the test applications for evaluating performance purpose, we only consider those pertaining to user records, such as inserting a user record or selecting a single or multiple user records. Each user record passed to or retrieved from the database is approximately 40-60 bytes (ASCII characters) long (this does not include XML tags).

Since in SOAP implementation, converting ADO recordset to XML string on the server and XML string to ADO recordset on the client may cause performance degradation, we also include test cases that do not require data conversion. In these test cases, the database server returns records as XML instead of ADO recordset. Returned XML data are passed directly to the client, which processes them using XPath.

When appropriate, we distinguish the test cases that process returned records, from the ones that do not (note: by processing records, we mean iterating through all fields of all returned rows). This helps us assess the difference between simply transporting data and transporting and processing data using each technology. The following test cases are used:

1. Inserting a single record 100 times (DCOM; SOAP with XML/ADO). Since data is passed to the database server and not retrieved from it, this test does not involve any data transformations; therefore, we do not need to use SOAP with XML/XPath here.
2. Retrieving 1, 100, and 1000 records 100 times (DCOM; SOAP with XML/ADO; SOAP with XML/XPath).
3. Retrieving 10000 records 10 times (DCOM; SOAP with XML/ADO; SOAP with XML/XPath).

The test machine is a single 400 MHz Pentium II processor system with 128 MB of RAM. It runs Microsoft Windows 2000 Professional with Service Pack 2. To avoid network-related variable factors, all components (the database server, COM objects, Web server, and Web browser) are configured to run on the same computer.

We execute each test case three times and calculate the ratios based on the averages. We also alternate DCOM and SOAP test executions to eliminate external factors (such as background processes) from affecting test results of a particular test group. After executing the tests, we calculate the following ratios:

1. SOAP test execution time to DCOM execution time for a corresponding test case. This shows whether, and if so how much, the SOAP implementation is slower than the DCOM.

2. SOAP test execution time with XML/ADO to SOAP execution time with XML/XPath implementation of the same operation. This indicates which SOAP implementation offers better performance.

When computing test execution times, we only count the time of test operations (e.g. inserting or retrieving records). Since the following operations are constant and do not differ between DCOM and SOAP, we *do not* include them in the calculations:

- displaying DHTML pages in the client's Web browser;
- data transfer from DHTML pages in the client's Web browser to the ASP on the Web server;
- ASP initialization (which includes retrieval of HTML form data, setting common variables, etc.);
- generating DHTML pages by the ASP;
- transferring DHTML pages from the Web server to the client's Web browser.

4. Result Analysis

4.1. Performance analysis

Test results confirm our hypothesis and indicate that SOAP performance is slower than DCOM for all test cases. Table 1 contains all test results (each type of test case was executed three times and ratios were calculated using the averages). The difference is especially apparent when operations affect single records, such as SQL INSERT and SELECT operations performed on one record multiple times.

Table 1. Performance test results.

Test Cases / Test Types		INSERT 1 record 100 times (in seconds)		SELECT 1 record 100 times (in seconds)				SELECT 100 records 100 times (in seconds)				SELECT 1,000 records 100 times (in seconds)				SELECT 10,000 records 10 times (in seconds)			
				w/ parsing		w/o parsing		w/ parsing		w/o parsing		w/ parsing		w/o parsing		w/ parsing		w/o parsing	
DCOM		7		1		1		23		20		68		35		22		5	
		4	4	1	1	1	1	22	22	20	20	64	66	39	37	20	21	4	4
		1		2		2		22		19		67		38		20		3	
SOAP	using XML/ADO	37		37		34		87		56		158		103		112		100	
		41	42	36	36	34	34	95	92	53	53	159	156	106	106	111	110	100	100
		47		34		34		93		50		150		109		107		100	
	using XML/XPath			35		32		87		57		100		125		121		100	
				35	35	33	33	76	87	55	57	110	109	101	108	117	118	100	100
				35		33		98		59		118		99		117		100	
SOAP/DCOM performance ratio (times slower)	SOAP w/ XML/ADO		10.4		26.8		25.5		4.1		2.7		2.3		2.8		5.3		25.0
	SOAP w/ XML/XPath				26.3		24.5		3.9		2.9		1.6		2.9		5.7		25.0
ADO/XPath					1.0		1.0		1.1		0.9		1.4		1.0		0.9		1.0

Contrary to the expectations, using XML/XPath as opposed to transforming data from ADO recordset to XML string on the server and then back to ADO recordset on the client had very little impact on the results.

Since we have determined that conversion between ADO recordset and XML string should not cause SOAP performance degradation, we can attribute the slowdown to the following factors: (1) extra work related to state management; (2) increased data payloads, and (3) SOAP runtime overhead (e.g. XML parsing).

State Management. In our scenario, SOAP statelessness results in (1) additional SQL login operations taken in each test case, and (2) processing HTTP headers on both client and server sides. Because each SOAP method call requires an extra login to the database server (via ADO Connection.Open call) and database logins are considered "expensive" operations, it would be logical to assume that this should contribute to performance degradation. However, ADO - which we use to communicate with the database server – provides a connection pooling feature, which makes connection opening operations rather "cheap". To test how much slowdown additional SQL logins can cause, we modified DCOM test harness to make additional database connection opening and closing calls for each performed operation (thus simulating extra SQL operations performed in SOAP implementation), but the test results showed no noticeable difference from the original DCOM test results. Therefore, we can conclude that additional SQL logins do not affect performance results. Preprocessing HTTP headers, on the other hand, could potentially affect the performance but unfortunately there is no reliable way to detect whether it is true, and if so, how much difference would it make.

Increased Data Payloads. An interesting trend is noted here. In the DCOM case, the more records are returned with each call, the less percentage of operational time is spent to transfer them and the more time is spent to parse through the record fields. This is not the case in SOAP. We see that in the SOAP implementation, initial parsing time drops, but at some point (e.g. 10,000 rows) it starts to grow and the higher the number of returned rows gets, the longer it takes to transfer them as percentage of total operation. It is interesting to notice that at 10,000 records, the parsing time for DCOM and SOAP is approximately the same, which can probably be explained by the fact that in SOAP case all data to be parsed is local to the caller (and therefore should take less time to process), while in DCOM case, each fetched record is retrieved from the server. This confirms our hypothesis about the effect of the increased payloads on the processing time.

SOAP Runtime Overhead. Even though, there is no easy way of determining how much time is spent in which phase of SOAP runtime infrastructure, it is apparent that SOAP runtime overhead is more expensive than DCOM runtime overhead. In DCOM case, data sent from a client to a server must only go through a proxy-stub pair responsible for packaging data on the client side, sending it over the wire, and unpacking it on the server side. SOAP processing is more complex. As Figure 3 shows, before it reaches the destination object, SOAP data goes through MSSOAP.SoapClient, Web Server, ISAPI filter (SOAP listener), and XML parser. MSSOAP.SoapClient in its turn, calls WSDL reader to get description of the service, and invokes SOAP serializer to generate SOAP packets, and SOAP reader to receive and process returned results. On the SOAP server side, requests are handled by MSSOAP.SOAPServer COM object, which performs operations similar to the ones done by MSSOAP.SoapClient. If data validation has to be performed (as in many XML-oriented applications), additional time must be spent checking data, resulting in higher performance penalty (In our tests, we did not use XML data validation). Obviously, SOAP process is more complex and should require additional time.

4.2. Development effort

Depending on the type of application, SOAP-based development can be anything from easier to harder than DCOM-based development. If the application has to support communication between multiple incompatible platforms or programs running on the internet (not within local area network or intranet), a single SOAP implementation is most likely to be easier to implement than separate versions of the application for each platform and adapters between them. On the other hand, for a single-platform development, SOAP implementation will probably require more effort, unless the application is trivial. The most common challenges for the application developers include implementing session state management and security.

State Management. As we already mentioned earlier, SOAP calls are stateless. Therefore, the applications relying on session information must implement state management programmatically, which may not be easy. In our test application, we pass state information using HTTP headers. Even though this approach is acceptable in a test environment, it would present a major security breach for production applications, because it passes sensitive data (such as usernames and passwords) over the network unencrypted. Designing and implementing a scalable, flexible, and secure state management solution for real-life applications is not a task for faint-hearted.

Security. Because SOAP is a simple wire protocol, it does not implement security. The only security-related option SOAP comes close to is support for SSL [37] over HTTPS [38], which handles encryption and decryption of data passed over the network [27]. While this may be enough for some applications, more complex systems relying on such DCOM features as *authentication* (identifying that you are who you say you are), *authorization* (verifying that you can do what you say you can do), and/or *identity* (impersonating another entity on someone's behalf), must also implement these options programmatically.

If we use the number of lines of code as a rough assessment of development effort, our SOAP version of the test application proves to take between 16% (for ASP development) and 23% (for VB development) more effort than DCOM version (Lines of code data for SOAP server do not include functions written for XML/XPath implementation, because these functions duplicate already existing functionality). The difference is due to the SOAP implementation of the state management and extra code used to perform ADO-to-XML and XML-to-ADO transformations.

5. CONCLUSION

Although SOAP offers a value of cross-platform interoperability, application developers must be aware that SOAP-based applications may suffer from a slower performance than applications using such platform-specific technologies as DCOM. Note that this conclusion is only pertinent to the single-platform applications and may not be true for the applications deployed in heterogeneous environments. As a rule, SOAP-based applications perform worst when (1) operations on single-record data are invoked (e.g. SQL INSERT or SELECT statements returning one row), or (2) large amounts of data (e.g. 1 MB or more) are transferred. Performance can be worsened even more by making SOAP calls over HTTPS instead of HTTP. SOAP-based applications, which do not rely on state information or such security-related features as authentication, should not take more effort to implement than comparable DCOM-based applications; however, such features as state management and security will require additional (and possibly non-trivial) development efforts. Table 2 summarizes the findings made by this research.

Table 2. Comparison of DCOM and SOAP features

Criteria \ Technology		DCOM	SOAP
Performance	single-platform	faster (2-25 times)	slower (2-25 times)
	multi-platform	not available	outside of scope of this paper
Development Effort (LOC)	single-platform	less (total 5513 LOC)	more (total 3687 LOC)
	multi-platform	not available	outside of scope of this paper
State Management	single-platform	intrinsic	not available
	multi-platform	not available	not available
Security	single-platform	authentication, authorization, encryption, identity	encryption (over SSL)
	multi-platform	not available	encryption (over SSL)

Even at the expense of performance, SOAP is probably still the best bet for the applications which must support cross-platform communication. With the right design, performance degradation can be diminished. Since SOAP is in its infancy, there is a hope that new tools (e.g. XML parsers) and frameworks (e.g. Microsoft's .Net) will become faster and thus help to improve SOAP application performance.

For the applications which run on a single platform within local area networks and do not communicate across firewalls, there is no obvious reason to use SOAP.

References

[1] "A Blueprint for Building Web Sites Using the Microsoft Windows DNA Platform." *MSDN Library*, Microsoft Corporation, January 2000.

[2] Box, Don. "A Young Person's Guide to The Simple Object Access Protocol: SOAP Increases Interoperability Across Platforms and Languages." *MSDN Magazine*, March 2000.

[3] Box, Don. "House of COM." *Microsoft Systems Journal*, July 1998.

[4] Box, Don. "House of COM." *MSDN Magazine*, January 2000.

[5] "Cascading Style Sheets, Level 1 Specification." http://www.w3.org/TR/REC-CSS1 World Wide Web Consortium, December 1996.

[6] Chappel, David. *Understanding Microsoft Windows 2000 Distributed Services*. Microsoft Press, 2000.

[7] "Component Object Model Protocol (COM) Specification 0.9" *Microsoft Developer Network*, Microsoft Corporation.

[8] "CORBA/IIOP Specification." http://www.omg.org/technology/documents/formal/corba_iiop.htm Object Management Group.

[9] "DCE 1.1: Remote Procedure Call. CAE Specification." http://www.opengroup.org/onlinepubs/9629399/ The Open Group.

[10] "DCOM Technical Overview." *MSDN Library*, Microsoft Corporation, November 1996.

[11] "Distributed Component Object Model Protocol - DCOM/1.0" *MSDN Library*, Microsoft Corporation, January 1998.

[12] Dengler, Chris. "Returning ADO Recordsets With SOAP Messaging." *MSDN Library*, Microsoft Corporation, July 2000.

[13] Dedo, Doug and Nelson, Greg. "Integrate the Enterprise." *MSDN Library*, Microsoft Corporation, March 1997

[14] DeJesus, Edmund X. "SAML Brings Security to XML." *XML Magazine*, February 2002.

[15] "Document Object Model (DOM), Level 1 Specification, Version 1.0." http://www.w3.org/TR/REC-DOM-Level-1/ World Wide Web Consortium, October 1998.

[16] "Document Object Model (DOM), Level 3 Events Specification, Version 1.0." http://www.w3.org/TR/2002/WD-DOM-Level-3-Events-20020208/ World Wide Web Consortium, February 2002.

[17] "Enterpise JavaBeansTM Specification." http://java.sun.com/products/ejb/docs.html Sun Microsystems.

[18] "Extensible Markup Language (XML) 1.0 (Second Edition)." http://www.w3.org/TR/REC-xml World Wide Web Consortium, October 2000.

[19] Fedorov, Alex, Homer, Alex, et al. Professional Active Server Pages 2.0. WROX Press 1998.

[20] Gross, Christian. "Building COM Components on UNIX." *MSDN Library*, Microsoft Corporation, July 1998.

[21] "HTML 4.01 Specification." http://www.w3.org/TR/html4/ World Wide Web Consortium, December 1999.

[22] Horstmann, Markus and Kirtland, Mary. "DCOM Architecture." *MSDN Library*, Microsoft Corporation, July 1997.

[23] "Hypertext Transfer Protocol - HTTP/1.1. RFC 2616." http://www.ietf.org/rfc/rfc2616.txt, Internet Engineering Task Force.

[24] "ISAPI and the Web Application Architecture." *MSDN Library*, Microsoft Corporation.

[25] Loshin, Pete. "Web Services and the Simple Object Access Protocol." *MSDN Library*, Microsoft Corporation.

[26] Lowy, Juval. "Web Services. Hurdle the Firewall." *.Net Magazine*, Fall/Winter 2001, Vol. 1, No. 1.

[27] Mar-Elia, Darren. "Understanding the '.Net' in .Net." *Windows 2000 Magazine*, November 15, 2001.

[28] Patrizio, Andy. "SAML Advances Single Sign-On Prospects." *XML Magazine*, March 2002.

[29] Ramasundaram, Sethuraman. "WSDL: An Insight Out." *XML Magazine*, October/November 2001, Vol. 2, No. 5.

[30] "SAML Specification." http://lists.oasis-open.org/archives/security-services/200106/pdf00002.pdf Organization for the Advancement of Structured Information Standards.

[31] Sconnard, Aaron. SOAP: The Simple Object Access Protocol. *Microsoft Internet Developer*, January 2000.

[32] Scribner, Ken. The extensible Markup Language - A C++ Developer's Primer (Part 3). *Visual C++ Developer*, July 2000.

[33] Scribner, Kennard, and Stiver, Mark C. *Understanding SOAP*. SAMS Publishing, 2000.

[34] "Simple Object Access Protocol (SOAP) 1.1." http://www.w3.org/TR/SOAP/ World Wide Web Consortium, May 2000.

[35] Snell, James, Tidwell, Doug, and Kulchenko, Pavel. Programming Web Services with SOAP. O'Reilly & Associates, Inc., 2002.

[36] "SOAP Toolkit 2.0 Documentation." *MSDN Library*, Microsoft Corporation, 2001.

[37] "SSL 3.0 Specification." http://www.netscape.com/eng/ssl3/ Netscape.

[38] "Upgrading to TLS Within HTTP/1.1. RFC 2817." http://www.ietf.org/rfc/rfc2817.txt Internet Engineering Task Force.

[39] "Web Services and the Simple Object Access Protocol." *MSDN Library*, Microsoft Corporation.

[40] "Web Services Description Language (WSDL) 1.1." http://www.w3.org/TR/wsdl World Wide Web Consortium, March 2001.

[41] "XML XPath Language (XPath) Version 1.0." http://www.w3.org/TR/xpath World Wide Web Consortium, November 1999.