Solutions for Interoperability Problems Faced in Web Applications
G. M. Tere\(^1\) and B. T. Jadhav\(^2\)
\(^1\)Thakur College of Science and Commerce, Mumbai- 400101, India
\(^2\)Y.C. Institute of Science, Satar - 415001, India
\(^1\)girish.tere@gmail.com, \(^2\)btj21875@indiatimes.com

ABSTRACT
Day by day we are using more automation. Future applications will be totally web based and will be using cloud computing technology. These applications will use heterogeneous platform and hardware. The main issue in these applications will be solving interoperability problem. We need to solve the interoperability problem efficiently, in such a way that computer resources are used efficiently. In past some of the solutions used are – RPC, RMI, CORBA, COM/DCOM. These solutions are having their own limitations. To overcome these limitations, technologist developed SOAP techniques. W3C standardized SOAP. But, since SOAP is XML document, the processing of SOAP need lot of computer resources and thus the performance of system goes down. This paper discusses these limitations and gives some useful solution to overcome these problems. We are proposing one more solution to reduce interoperability problems, based on "Contract-First" design method.

KEYWORDS
Contract-First, Interoperability, SOAP, Web service, XML.

1. INTRODUCTION
Day by day we are using more web applications. Web applications like Railway Reservation System, Airline Reservation System, Banking systems, Library Systems are developed with heterogeneous technology. Web applications are developed for different platforms which will work on ubiquitous devices. These applications deal with different databases like SQL Server, Oracle, MySQL and so on. The important point in developing such distributed web application is Interoperability. Web services are independent application components published on to the Web in such a way that other Web applications can find and use them. They take the Web to its next stage of evolution, in which software components can discover other software components and conduct business transactions. Examples of Web services are
- a credit card service that processes credit card transactions for a given account number,
- a market data service that provides stock market data associated with a specified stock symbol, and
- an airline service that provides flight schedule, availability, and reservation functionalities.

The Simple Object Access Protocol simplifies information exchange across a variety of platforms because it is not tied to any specific object model. SOAP began as XML-RPC. As the name suggests, XML-RPC was to perform remote procedure calls using XML. Existing RPC protocols, such as the Distributed Component Object Model (DCOM) and the Internet Interoperable Orb Protocol (IIOP), were not well suited for Internet use. These protocols require a significant amount of dedicated runtime support, and many firewalls do not permit access by non-HTTP protocols. DCOM and IIOP are tied to specific object models that do not interoperate. Eventually, its developers changed XML-RPC’s name to the more generic Simple Object Access Protocol. IBM developed a modified version that reflected the W3C XML Schema specification, and submitted it to the W3C in early 2000 as SOAP 1.1. The W3C standardized SOAP. The Simple Object Access Protocol has garnered a lot of interest lately, especially since the W3C undertook its standardization and Microsoft incorporated the protocol into its products. SOAP offers interoperability [11][19] across a wide variety of platforms. It enables interoperability by providing a generalized specification for invoking methods on objects and components using standard hypertext transfer protocol (HTTP) calls and Extensible Markup Language (XML) data formats. SOAP is not tied to a specific object model, so a client written in Microsoft VisualBasic, for example, could use SOAP to access a method in a Common Object Request Broker Architecture (CORBA) object running on a Unix platform.

2. WEB SERVICE ARCHITECTURE
A typical Web services architecture consists of three entities:
- service providers who create Web services and publish them to the outside world by registering the services with service brokers;
- service brokers who maintain a registry of published services;
- service requesters who find required services by searching the service broker’s registry. Requesters then bind their applications to the service provider to use particular services.

Fig. 1 shows basic Web service architecture. It shows the interaction between service providers, service brokers, and
service requesters in the publication, discovery, and consumption of Web services.

Fig 1: Basic Web Service Architecture
Fig. 2 illustrates protocol stack, used in Web service.

Web services should be used when we need to use software developed with different platforms. Interoperability problem can be solved using web services. We have to Web services with precautions, otherwise because of XML performance of overall web application may decrease. Reuse of legacy code to enable J2EE, .Net we can use web services.

3. THE SOAP SPECIFICATION
The SOAP specification defines

- An envelope, a framework for expressing what is in a message, who should handle the individual parts of the message, and whether handling each component is optional or mandatory;

- Encoding rules, a serialization mechanism for exchanging instances of application-defined data types; and

- The SOAP remote-procedure call (RPC) representation, a convention for representing RPCs and responses.

4. REMOTE PROCEDURE CALLS
The SOAP 1.1 specification defines a standard for RPCs and responses. To invoke a method, software designers must include the target object’s URI, the method name, and the method’s parameters in the SOAP message. Optionally, programmers could include the method signature and additional header data. The SOAP body element carries both RPC method calls and responses.

4. SOAP APPLICATIONS
As noted before, SOAP is object model neutral. Object functions, such as method implementations and garbage collection, are left to the software infrastructure supporting the clients and servers. No distributed-object architecture is needed, because SOAP can send messages from one object to another across the Internet without knowing what type of object is sending or receiving the message. Also, using Internet wide standard HTTP for transport eliminates the firewall barriers that have hindered the acceptance of proprietary and platform-specific transport protocols. Users can implement SOAP in any programming language; current implementations include those in Perl, Java, and Visual Basic.

5. WEB SERVICE DESIGN METHOD
There are essentially two ways of designing web services, the contract-first [20] and the code-first way. These two methods are shown in Fig. 3.

5.1. Contract-First Method
The first step in this method is to define different SOAP messages, that should be used, in other words specify a WSDL [1] file. The next step is to be certain that Server and Client generate accurate SOAP messages based on WSDL file. In this method, "Stub Codes" of Server and Client are automatically generated from WSDL file. "Contract-First" method has high
interoperability level, since both Server and Client use the same reference which is WSDL file.

5.2. Code-First Method
The first step in this method is to design WEB service code [2][9] and its interface. Then WSDL file is generated from this code. In this method designer has no control on SOAP messages. Client codes could be written from scratch; but generation of correct SOAP messages has low chance in this situation. The better procedure is to generate "Client Stub" immediately from WSDL file. However, "Code-First" method generally makes more interoperability problems in comparison with "Contract-First" method. Regardless of type of design method, Server and Client must understand each other. "Contract-First" method improves the interoperability level, while the latter increases the productivity [20].

5.3. Interoperability Problems in WEB Service
One of the most common problems in both methods of WEB service design is that the generated code in Server/Client is unable to understand and manage exceptions. Therefore, by an exception taking place program would halt. Although error management in WEB service is approximately similar to that of other programs like JAVA, exception management is more difficult due to transparency of SOAP messages. For example suppose an error occurring in server. In this situation a "SOAP Fault Message" is sent to the Client. If Client application cannot parse and understand this error message, application is ended because of unhandable exception. This problem mostly happens when multi-vendor Client and Server are used; for example Server uses J2EE platform and Client uses .NET. To solve the problem a "custom SOAP Message Parser" [8][9] is required which must parse "Detail Field" in received SOAP message and generate a proper exception. "Detail field" in SOAP fault messages contains adequate information about occurred fault. In this case, client receives a correct exception and processes it properly. The interoperability problem in contract-first method is only this SOAP fault problem [16][19]. Because in this method WSDL files are validated, then Server and Client codes are generated from these files. "Code-First" method has additional problems, since Server and Client codes are generated in a different manner. One of the most important problems is "Encoding Style", depending on the structure of the SOAP messages and encoding. As an example, "Missing Array Structure" fault may occur in which generated code for Client is unable to parse the received data properly. Another problem is the conversion of various types of data to each other. There is no "One-to-One" map for different types of data. Although WEB service design is independent of software/hardware, some interoperability problems may appear especially when multi-vendor Server and Client are used. To solve these problems, WS-I (institute established by renowned companies like Microsoft and IBM) provides a standard named "Basic Profile" that proposed Solutions to Decrease WEB Service Interoperability Problems. In this paper different WEB service design methods are compared in terms of interoperability. Based on the results, a proper design method and some solutions to reduce interoperability problems are suggested. To have a better comparison between "Contract-First" and "Code-First" methods in terms of interoperability, both of them are tested by WS-I test tools; while Client and Server use .NET and J2EE platforms, respectively.

6. TEST ASSUMPTIONS
A simple WEB service is considered for the test, in which a Client with .NET platform and a Server with J2EE platform are connected to do SMS application. This application includes capabilities of "SMS transmission" and "delivery status reception" by the Client. In this service, Client and Server have unidirectional connection. This simple WEB service is designed with both "Code-First" and "Contract-First" methods.

6.1. Proposed Solutions
With the help of above tests, we suggest solutions to decrease interoperability problems:

- "Contract-First" method would be used in WEB service design. It means design would be started with definition of required SOAP messages or WSDL file.
- Only simple data structure which has "One-to-One" map in all environments could be used in data transmission. Obviously, processing of complex data structures may be difficult for receiver and may cause some interoperability problems.
- WSDL file would be validated based on WS-I "Basic Profile". This may solve most of the interoperability problems. On the other hand, it would better to be certain about validation of WSDL files firstly, and then compare SOAP messages with these files.
- Server and Client codes would be generated based on WSDL file.
- SOAP traffic is validated on the basis of WSDL file and WS-I "Basic Profile". "Monitor" must be used in order to save messages in a "Log File", and then these messages must be compared with the definitions in WSDL file, based on "Basic Profile".
- If SOAP messages are incorrect and have failed verification, the causes lead to errors could be discovered by detailed description in "Conformance Report". Then SOAP messages must be corrected and the errors must be solved.

7. WEB SERVICE PERFORMANCE BOTTLENECKS
Without a doubt, the three most common bottlenecks in today's web services-based solution are related to:

- Parsing of SOAP messages. The larger the message, the longer the time required to parse it.
- Marshalling and un-marshalling of Objects to XML and XML to Objects. The more complex the structure of a
message, the longer the time required to map between programming objects and XML elements.

- Processing of WS-Security capability that includes XML Digital Signatures and XML Encryption. Security between application endpoints isn't free and can add surprisingly long latency to the processing of service requests.

SOAP is potentially useful for various generic service-provider functions. For example, an application service provider could develop a SOAP implementation of a customer authentication function and then sell it to e-business operators, who would incorporate it in their websites. Below are a set of common practices that will direct your design to optimize your solution's efficiency with respect to the three bottlenecks itemized above. Most recommendations involve trade-offs for improving performance, but should be considered with respect to your overall set of requirements.

7.1. Payload size versus payload complexity

The obvious rule for optimizing your web service performance is to keep your payload small and simple. However, in the real world where you’re trying to solve real business problems, you do not always have the luxury of adhering to this rule. Long running business processes may require that XML documents be exchanged that capture not only the relevant business information, but also the state of the process. Larger messages result in longer parsing times while complex XML structures with nested elements result in longer times for the marshalling and unmarshalling of Objects and XML elements. The goals should be an awareness of these impacts and spending time architecting your programming objects to minimize the size and complexity of the XML message structures. However, you should choose to support a single invocation that includes a somewhat larger and more complex message versus supporting separate individual message transactions. Consider multiple messages when the business functions or performed tasks are truly independent, or when one task may significantly delay the processing of other combined tasks.

Document/Literal versus RPC/SOAP encoding

The Document/Literal messaging style should be used for your web services whenever possible for two reasons. First, it promotes interoperability in compliance with the Web Service Interoperability (WS-I) Basic Profile 1.0. The second reason falls within the scope of this article on performance.

Document/Literal results in smaller and less complex messages: the XML data in the body of the SOAP message does not have to be wrappered with a method name element, and no data type attributes are inserted into the XML elements. One other benefit of the Document/Literal messaging style is that today's Integrated Development Environment (WebSphere Application Studio Development) and runtime (WebSphere Application Server) support JAX-RPC serializers and deserializers routines for marshalling of Objects and XML elements that are optimized based specifically on the XML schemas included in WSDL, as opposed to serializers and deserializers associated with SOAP encoding.

7.2. Parsing of SOAP messages

If a business function is to be exposed as an XML web service that leverages SOAP [7][12][19] for both internal consumption (EAI) and for external consumption by business partners (B2B), intermediaries such as gateways or service agents should avoid or minimize parsing of the SOAP Body. If a gateway component is used to centralize access of web services to the Internet, but no network transport or message manipulation is needed (such as SOAP/HTTP to RMI/IIOP), then the gateway should not perform parsing of the SOAP body. Many system management vendors today provide service agents that front-end the actual web services. These components rely on business context information within the SOAP body, such as business partner IDs, transaction correlators, message IDs, and authorization codes in providing their system management capabilities. Using the business context, the service agents provide statistics on business events, enforce business policies, and route requests to meet quality of service commitments. Recently, the Web Services Gateway in WebSphere Application Server V5.1 supports partial parsing of SOAP messages. Likewise, system management vendors have recently started to provide the capability to partially parse SOAP messages to minimize their impact on performance, so it's vital that these capabilities be utilized.

7.3. Marshalling and un-marshalling Objects and XML

The more complex your Objects and XML Schemas are for messages, the more processing will be required by both the client and service provider. The client will have to marshal their programming objects to XML elements before issuing a request, and the service provider will have to map the XML elements to programming objects in processing the request. Objects that are constructed with arrays of arrays or XML elements consisting of nested elements of nested elements will certainly be a bottleneck if conscious decisions are not made in architecting your data for both request and response messages. Today, many companies are standardizing on using Open Application Group Information System's (OAGIS) Business Object Documents (BOD). The XML schema for these XML documents include several levels of nested XML elements, so it is important that when using these BODs, you evaluate the impact they may have on your overall performance. The recommendation is to be selective in what BODs you use within your solution. It is also important to architect your messages so that you maximize the amount of real information that's being exchanged. Messages with many elements and attributes and little data are usually the result of complicated XML Schemas. Most often, parsing of SOAP messages is viewed as the major contributor to performance issues with web services. However, a complex message structure can also result in more than 50% of the processing time being associated with marshalling and un-marshalling of Objects and XML elements.

The following test were performed:
Testing performance difference between the XML parsers

The test has been performed on two personal computers with the following specifications:

Dell Inspiron Laptop 1:
- CPU: Intel Core 2 Duo, 2GHz
- Memory: 4GB DDR RAM
- Operating system: Microsoft Windows XP Professional, SP2

PC 2:
- CPU: AMD Athlon XP Processor, 1.67 GHz
- Memory: 1.6 GB DDR RAM
- Operating system: Microsoft Windows XP Professional, SP2

7.4. XML Parsers

There are three types of XML parsers: Streaming API for XML (StAX), Simple API for XML (SAX) and Document Object Model (DOM). There is a lot of performance difference between the types of parsers. In order to validate or reject this and, tests were conducted. The following components were involved in these tests:

- XMLGen: A benchmark data generator that generates XML documents of various sizes. The size of the XML document to generate can be provided by the user of XMLGen.
- Four different XML documents that were generated by XMLGen, varying in size:
  - xmark_0.00001.xml, 25KB
  - xmark_0.0001.xml, 42KB
  - xmark_0.001.xml, 122KB
  - xmark_0.01.xml, 1.1MB
- A SAX implemented parser.
- A StAX implemented parser.
- A DOM implemented parser.
- A Java class that did the actual comparison of the parsers.

The first tests that were performed were the actual parsing of XML documents of varying size mentioned above. We did the performance tests in two phases: first, the smallest document (25KB) was parsed by the StAX, SAX and DOM parsers in the warm up run. Next, all the documents generated by XMLGen were parsed by the three parsers. Partial results of these tests are shown in Fig. 4 and Figure 5.

8. CONCLUSIONS

There are three possible causes for low SOA performance: SOAP Encoding styles, Java SOAP bindings and XML parsers. By proper selection of these parameters during the implementation of an SOA environment and considering each of these possible causes, a large performance increase can be gained. SAX, StAX and DOM differ a lot in performance. When considering the differences in performance and implementation, using DOM would be the best for small documents. DOM is easiest to use and implement. However, as the size of the XML documents grow, it is recommended to look at other solutions, such as SAX or StAX. When dealing with large XML documents, it is better to use XML Binding techniques.

Though Web service is an important phenomenon in distributed applications field, it has many problems. The most significant one is the interoperability between elements, which is quite obvious when multi-vendor Client and WEB service are used. In general, WEB service design has two methods, "Contract-First" and "Code-First". Analyze of possible interoperability problems has shown that the Contract-First method has better interoperability level in comparison with the latter. Moreover, some solutions have been proposed in order to reduce the mentioned problems.
REFERENCES


