Optimizing Web Service Messaging Performance in Mobile Computing

Sangyoon Oh and Geoffrey C. Fox
Community Grids Lab, Indiana University, Bloomington, Indiana, 47404, USA
Computer Science Department, School of Informatics, Indiana University
{ohsangy, gcf}@cs.indiana.edu

***Abstract*— The performance and efficiency of Web Services in conversational and streaming message exchanges can be greatly increased by streaming the message exchange paradigm. In this paper, we describe our design and implementation of a novel approach to message exchange optimization. This area is particularly important for applications in physically constrained mobile computing environments, but there is great potential for other applications. The verboseness of XML-based SOAP representation imposes possible overheads in mobile Web Service applications. In order to help minimize such overheads, we separate data content from the syntax and use streaming message exchanges. The redundant or static message parts are stored in a shared metadata space – the Context-store. Therefore, the streamed messages themselves are not self descriptive, but the combination of the message and the negotiation captured in the Context-store is self descriptive. We describe our architecture and evaluate our approach by testing the performance of the resulting system. The empirical results show that our framework outperforms conventional Web Services in both conversational and streaming message exchanges with mobile clients. We demonstrate how to find the breakeven point at which our methods overtake the conventional SOAP messaging for a particular application.**

***Index Terms*—Grid/Web Service, Quality of Service, Web Service Performance, Mobile Application**

I. Introduction

Web Service-based Service Oriented Architecture (SOA) have become a backbone of Grid computing because of its interoperability across diverse services/applications in a distributed environment. The Open Grid Services Architecture (OGSA) [1] [24] has defined the environment for offering Grid computing as a Web Service. Similarly, the simple interface and interoperability of Web Service architecture make mobile computing applications adopt Web Services as a model of communication.

But the verbose nature of the current XML-based SOAP [2] requires an alternative and more efficient solution for message exchanges in mobile environments, which have many physical constraints like limited processing, battery power, and slow or intermittent connections. The conventional SOAP communication model possesses major characteristics that may affect messaging performance. Serializing and de-serializing SOAP messages consume many resources. For example, floating point number or any other in-memory representation must be converted from and to the textual format of a SOAP message. This is an expensive process for limited mobile computing. Also, the message size is increased substantially by adding descriptive tags of XML syntax, and this can present a problem for narrow mobile connections.

High performance SOAP encoding is an open research area [4-6]. There have been many investigations attempting to address the performance issues of mobile Web Services and provide adequate solutions. But these proposals and solutions generally only tackle small pieces of the problem rather than provide a system level solution. In this paper, we describe our novel architecture that increases the performance of message exchanges in mobile Web Services environments. Our Handheld Flexible Representation (HHFR) architecture provides a complete system from the representation of the message to the use of a dynamic metadata repository for guaranteeing semantic consistency. The streamed messages are self descriptive when combined with the captured parts in the Context-store. Our system provides a logical binding between messages and the negotiation captured in the Context-store. As we show in Section 5, our framework outperforms the conventional SOAP messaging. We also demonstrate how to find the

breakeven point at which our method is more efficient than the conventional SOAP messaging.

We organize this paper as follows. In Section 2, we discuss background work. Section 3 reviews the HHFR architecture design. We illustrate implementation details of the system in Section 4. In Section 5, we discuss the performance evaluation based on our performance model. We conclude in Section 6.

II. BACKGROUND

The report of the W3C Workshop [7] on the Binary Interchange of XML Information Item Sets (Infoset) [8] is the result of the increasing demand for a binary form of XML-based communication. The report includes the conclusion of the workshop meeting on September 2003 as well as several dozen position papers from various institutes [5, 9, 10]. The purpose of the workshop was to study methods for compressing XML documents and transmitting pre-parsed and schema specific objects. It identified the requirements of binary XML Infoset, for example a) maintaining universal interoperability, b) producing a generalized solution that is not limited to a specific application domain, c) reducing process time including data binding time, and d) negotiation – i.e. a fall back to XML/SOAP text formatting if the receiver cannot understand binary. More recently Web Service performance has been reviewed at the 15th Global Grid Forum workshop (GGF 15) [11].

We divide the current approaches of expediting Grid/Web Service communication into the following categories. First, most proposals that follow the XML Binary Characterization of the W3C have the goal of producing a self-contained alternative to an XML message; it would be optimized for faster processing and smaller packet size. The approaches in this category replace a redundant vocabulary with indexes. Sun's Fast Infoset project [6], XML Schema-based Compression (XSBC) [12], and XML Infoset Encoding (XBIS) are examples of the category.

Secondly, there are non-self contained alternative approaches, such as Sun's Fast Web Services [5], the Indiana University Extreme! Lab's recommendation

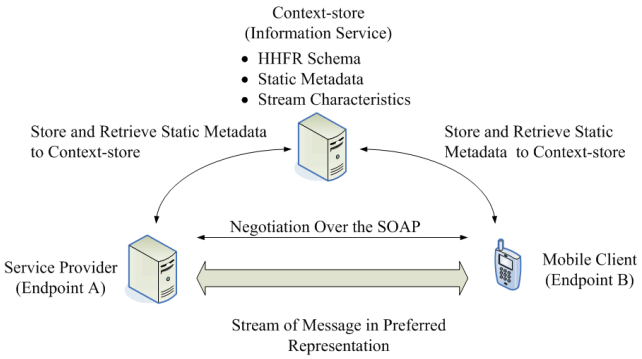

Figure. 1. HHFR Architecture Overview

[4], and the HHFR presented here.

The last category includes message compression approaches. Compressing an XML document reduces the size, but increases processing time. Even XML-specific compression like XMill [13], which achieves a better ratio than conventional compression utilities like GZip [14] (the experiment in Ref [13] shows XMill performs twice or better), doesn't improve performance much because of the additional layer of processing, i.e. compression and decompression.

The Global Grid Forum's Data Format Description Language (DFDL) [15] is a descriptive language that is proposed to describe a file or a stream in a binary format for Grid computing. Like the older Extensible Scientific Interchange Language (XSIL) [16], it is XML-based and comes with an extensible Java data model. DFDL architecture defines three primary layers: the lower layer (Mapping), the central layer (abstract data model), and the upper layer (API). The mapping layer defines the mapping between concrete representation and information content. For example, it defines a number format of data, whether it is a big-endian or little-endian, and a complex data format such as an array. The data model layer defines the data structure independently of their physical representation. It supports most of types that XML Schema Definition does [17].

Approaches described in this section are focusing on optimizing message representations to improve performance. Our HHFR architecture described in following sections is an overall framework which uses a Web Service database (i.e. context-store) to reduce a size of message as well as support a streaming style message exchange, not only optimize

message representations. So our approach is complementary not competitive to approaches described in this section.

III. The Design Principles

The key design goal of the HHFR architecture is optimizing Web Service messaging. Smaller size messages reduce the transit time of messages, which is a big gain for high latency and slow wireless connections. Also, by simplifying the structure of messages, the HHFR runtime system expects to reduce parsing and serializing overheads imposed by the verbose nature of SOAP. We achieve an optimized representation of message content by separating the message content from its syntax and streaming them in a preferred representation. Figure 1 depicts the overview of the HHFR architecture.

3.1. Replacement of XML Syntax With An Optimized Representation

HHFR provides a message exchange option in a preferred representation other than the conventional SOAP. An XML Based SOAP message itself is syntax (structure and type), and its verbose nature could impose a performance bottleneck, which is magnified in wireless computing environments.

The SOAP message has an outer-most element SOAP Envelope in its XML Document and it is composed of optional headers and a body. The architecture handles static information (unchanging headers) of the messages and dynamic information (payloads and headers that are applied to the individual message) differently.

The redundant or unchanging headers are stored in the metadata repository, the Context-store. The application can store static information either in the negotiation stage or in the middle of the session.

The body element contains a payload that includes program instruction and/or data. Compared to the individual message conversion approach that converts SOAP messages into another self-descriptive message format, our message stream approach requires a data description written in a DFDL-style data descriptive language and an internal data model. So the data represented in the

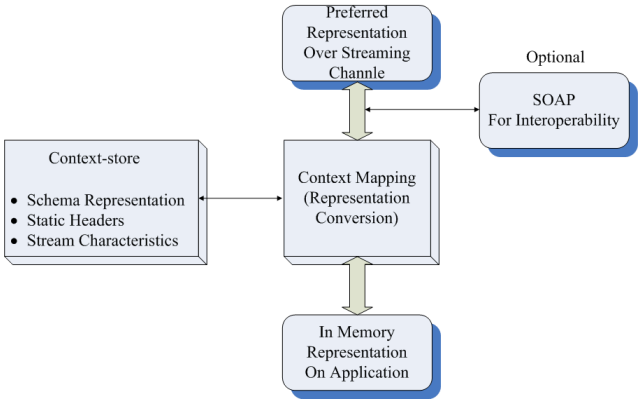

Figure. 2. Relationships between different forms of SOAP messages and their defining context

preferred format is not self-descriptive, but it is when it is binding with the data model which is gotten from the data description file. The relationships between data formats and representations are depicted in figure 2.

A binary representation increases the performance of the HHFR architecture for several reasons. First, we can save the bandwidth of message exchange. Since the descriptive tag of XML syntax increases the size of exchange data, having the content data in a binary format could demonstrate a savings as high as a factor of ten if the message structure is especially redundant – for example in the case of an array. A very simple message with a single text element can have its size reduced by half [3]. Secondly, the HHFR architecture can avoid a textual conversion; the process converts non-textual data into the text format and vice versa by adopting a binary representation. This is an expensive process, especially for relatively low-powered mobile devices.

3.2. Focus on Conversational Message Exchanges

HHFR works best for Web Services where two participating endpoints exchange a stream of messages like a conversation. For applications using a specific service, messages in the stream have the same structure and the same data information. Further, much of the message header is identical. Therefore the structure and type of SOAP message content in the HHFR schema can be transmitted once, and the rest of the messages in the stream only have updated payloads. To establish such a message stream, two endpoints should negotiate at the

beginning of the session. They negotiate the preferred representation (for example, a binary representation), transport characteristic (TCP or UDP), and quality of service issues (reliable messaging and/or security). The negotiation uses a conventional SOAP message, so that the two endpoints fall back to the SOAP message based Web Service communication if they fail to negotiate.

3.3. Negotiation

The negotiation stage is required by the architecture design to set up the stream characteristics. As discussed, the HHFR architecture uses the non-self-contained representation to exchange messages. So data, in the exchange of messages, should be paired with description information to be processed by the corresponding endpoint. Two participating endpoints should exchange each other's data description information at the beginning of the stream.

Negotiating the method of message exchanges is also essential during the negotiation stage. It is a well known fact that data streaming can increase message exchange performance in Web Services. So some researchers investigate this option using HTTP Persistent connection, SMTP, or asynchronous messaging service, such as MQSeries as a transport. K. Chiu et al. [3] suggest using chunk overlaying and a pipelined sending over an HTTP persistent connection; yet this is not always available as a network protocol implementation on all mobile devices and all cellular networks.

The HHFR architecture provides a fast communication channel. The channel can be either the same channel that is used for conventional SOAP message exchanges or a separate channel. If the fast communication option is implemented as a separate channel, it is efficient in its performance and also flexible in that it can be implemented in various ways including for asynchronous messaging schemes. But it also requires additional network resources – additional ports – and software modules to establish the connection. If the option is implemented as the same SOAP channel, it achieves an even higher interoperability, but it is also required to modify or adapt the transport layer implementation of a SOAP Server.

Two participating endpoints negotiate all the issues above: the data format by exchanging a description file, the fast communication channel setting up the connection, and any related quality of service issues.

3.4 Context-store (Information Service)

WS-Context [18] compliant Information Service is a message-based interface to a DB. As we discussed, the use of a Context-store reduces the bandwidth usage, but it also ensures the system is semantically consistent. Note that the streamed messages are not directly self descriptive. However the combination of the message and the negotiation captured in the Context-store is self descriptive. For example, it provides a fault-tolerance feature, which means that when the service endpoint is out-of-service, the service gets the required context from the Context-store after recovery. The use of the Context-store guarantees that the system and the participants stay semantically consistent: if there is a third party who audits a session, it also gets contexts from the Context-store and understands a session by filling up the missing SOAP parts saved in the store.

IV. IMPLEMENTATION

To demonstrate the effectiveness of the HHFR architecture, we have implemented a prototype mobile Web Service framework based on the architecture. The HHFR architecture consists of the HHFR Schema and processor, a fast communication channel with flexible representations, and the Context-store. Steps of the normal session are as follows: 1) HHFR-capable endpoint sends a negotiation request to the intended endpoint. The negotiation request is a conventional SOAP message that includes characteristics of the following session. 2) In the negotiation message, a service client endpoint – a negotiation initiator – sends an input data description written in the HHFR schema, which we describe later in section 4.2, and a service endpoint – a negotiation responder – sends an output data description. 3) Two endpoints use a second transport channel for the message exchange where

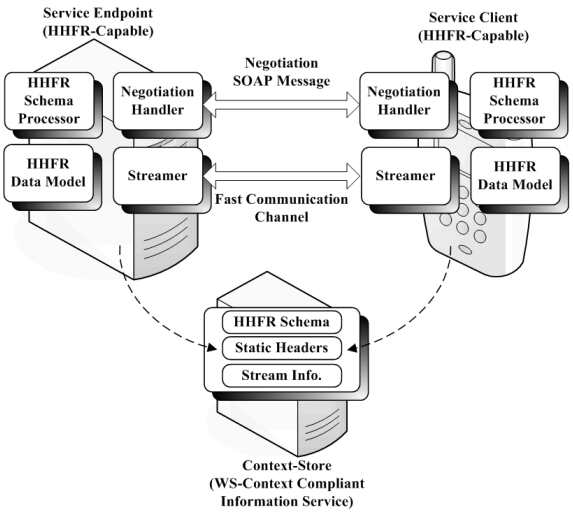

Figure. 3. Simple Overview of Implementation

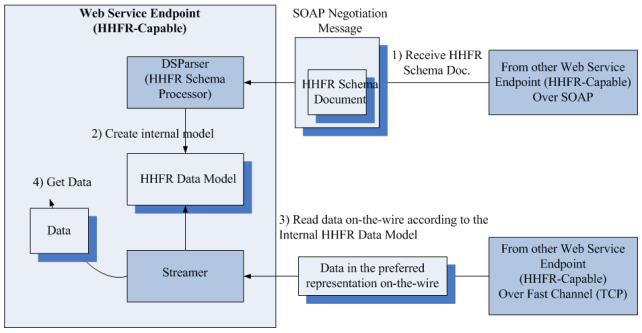

Figure. 4. HHFR Schema processing and interactions between related modules.

they stream messages. Messages in the stream are in the form of the negotiated representation. 4) The redundant or unchanging message parts – static metadata – are stored into a dynamic metadata repository, the Context-store, during the session.

To focus on investigating the optimization of message exchanges, we use existing efforts to address and implement issues that are out of our research interests, such as a Web Service container, a SOAP parser for the mobile environment, and a metadata repository. The overview of implementation is depicted in figure 3.

4.1. Negotiation Scheme

A normal HHFR session starts with a negotiation stage, where two endpoints exchange negotiation SOAP messages. By design, a negotiation stage is essential to establish the characteristics of the following stream. During this stage, a service endpoint returns characteristics that are suggested by a negotiation initiator and are selected and confirmed by the service endpoint. In the prototype implementation, this stage simply starts when the initiator sends a SOAP request to an intended service endpoint and ends when the initiator receives a response from the service.

The negotiation stage distinguishes whether the service endpoint is HHFR-capable or not. Since the negotiation stage is performed over the conventional SOAP protocol, this interoperable method enables the service endpoint (the negotiation responder) to reject a HHFR session and use a conventional SOAP based Web Services communication. The client (the SOAP initiator) must fall back to the conventional Web service messaging if it receives a SOAP fault or a SOAP error, which mean the responding service doesn't have the proper (exported) method in it and doesn't understand the negotiation SOAP message.

4.2. HHFR Schema: Data Description Language

To map non-XML based data – separated message content and XML data – SOAP message or any preferred representation, we define a DFDL-style data descriptive language, the HHFR schema. It is a small subset of XSD with some additions. The architecture of the HHFR schema is similar to that of DFDL: the HHFR Schema describes the data format, a Schema Processor (`DSParser`) builds a HHFR data model, and the Streamer converts data content from and to a preferred presentation format data.

HHFR Schema defines a subset of XSD components: simple type definition, complex type definition, element declaration, and attribute declaration. HHFR Schema defines a limited number of simple type built-in to the XML Schema. They are `string`, `int`, `byte`, `float`, `Boolean`. The current version of HHFR Schema doesn't support user-defined `sympleType`. The `complexType` element of the HHFR Schema can have mixed content, but cannot have simple content or empty content. So we declare `complexType` element is without `mixed` attribute. The following is an example:

```
<xs:element name="HHFR">
```

```
  <xs:complexType>
    <xs:element name="String1" type="string"/>
    <xs:element name="String2" type="string"/>
  </xs:complexType>
</xs:element>
```

The HHFR Schema processing involves several modules as depicted in figure 4. The HHFR Schema processor, `DSParser`, gets a HHFR Schema, which is contained in the negotiation request and response SOAP messages as depicted in step 1) of figure 4, as an input and produces an internal HHFR data model as an output as depicted in step 2). The relation between the HHFR Schema and HHFR data model is similar to the relation between an XML Document and its Java DOM Object.

After these two steps, the HHFR runtime is ready to start a fast communication option, which is discussed in the following section, and to process input data through `streamer`. The `streamer` is an interpret-style stub object [19], which is a popular design style in many data marshalling implementations. Compared to the more efficient compiled-style stub [19], which is popular in many client and server RPC implementations, the interpret-style stub is more flexible to a dynamic representation of input data. The stub doesn't need to be re-complied to different data representations. The stub reads and writes message packets through switch statements; a message packet is a unit of message in a preferred representation. In the prototype, a binary representation that is a sequence of bytes is a default representation format for the message packet.

4.3. Data Streaming

Data streaming is the key feature of our HHFR Prototype design, and it enables the system to achieve efficient message exchanges in mobile Web Services environments. Through streaming, message exchanges overcome the wireless network problems such as high latency and slow connection. Especially in flexible representations, we shorten the message transit time and reduce the bandwidth usage.

A fast communication channel of the HHFR Prototype provides an asynchronous and optimized alternative to the default HTTP communication method. As described, the negotiation response from the service must contain the endpoint address (IP and port number). The second communication channel is initiated by the service client.

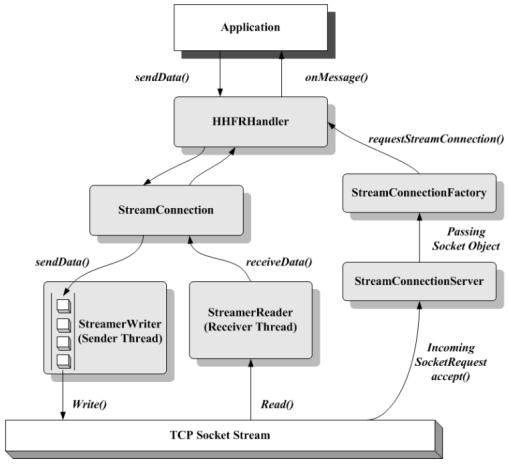

Figure. 5. Fast Communication Channel Layer

The fast communication channel layer for the TCP receptor is shown in figure 5. On the service provider side, `StreamConnectionFactory` waits for an incoming connection on a server socket and creates a `StreamConnection` that holds all streaming related classes, such as a `StreamReader`, `StreamWriter`, and `Streamer`. Data that an application attempts to send is queued in a `StreamWriter`. The path includes `HHFRHandler` and `StreamConnection`. Received data follows the opposite path and is delivered to the `onMessage` method.

4.4. Context-store

The redundant message parts may be treated as metadata and placed in a metadata store. We adapt an Information Service (metadata catalog system) for storing the transitory metadata needed to describe distributed shared information. The Information System, Fault Tolerant High Performance Information Service (FTHPIS) [20 - 21] which uses and extends the WS-Context Specification [18], is developed by Community Grids Laboratory of Indiana University and is being used as a third party transitory metadata store to store redundant parts of the SOAP messages, which are being exchanged between two endpoints. This way, the size of the SOAP messages is being minimized to make the service communication much faster. The redundant

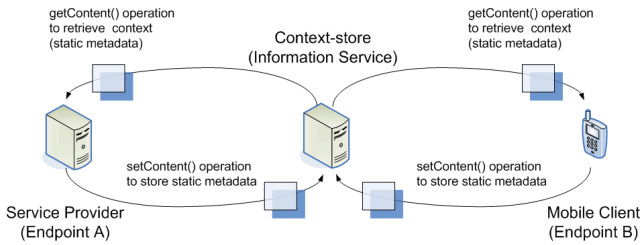

Figure. 6. Context-store operation overview

parts of a SOAP message can be considered as XML fragments which are encoded in every SOAP message exchanged between two services. These XML elements are stored as context into the Information Service. Each context is referenced by a system defined URI where the uniqueness of the URI is ensured by the Information Service. The corresponding URI replaces the redundant XML elements in the SOAP messages so that the message size can be reduced for faster message transfer. Upon receiving the SOAP message, the corresponding parties interact with the WS-Context compliant Information Service to retrieve the context associated with the URIs listed in the SOAP message.

As depicted in figure 6, the two primary WS-Context related functionalities of Information Services are `getContent()` and `setContent()` methods, which provide access and store operations method that can be called up whenever the endpoint needs to create, update, or retrieve a context in the Context-store (Information service). The Context-store client can do the following things: 1) create a `ContextServiceHandler` object with the Context Service URI, 2) store the given context of any type paired with an unique identifier, and 3) retrieve context. The `ContextServiceHandler` object is a wrapper class and provides `getContent()` and `setContent()` methods.

V. EVALUATION

We perform benchmark tests to evaluate our investigated framework. The comparison between the performance results of the conventional SOAP and the results of our system shows how much performance gains we have.

5.1. Performance Cost Analyze Modeling

We propose a cost analysis model for the HHFR runtime system. We assume the following basic system parameters to analyze the cost.

- t_1 : cost (time delay) per message for HHFR session
- t_2 : cost (time delay) per message for the conventional SOAP session
- O_a : overhead time for accessing Context-store
- O_b : overhead time for negotiation stage
- O_c : overhead time for Design HHFR Schema document.

Assume we have n messages in a session. The cost of message exchanges in a HHFR session consists of the message exchange cost ($t_1 n$) and overheads ($O_a + O_b + O_c$). Thus, the total cost is found by:

$$C_{hhfr} = t_1 n + O_a + O_b + O_c \quad (1)$$

The cost of message exchanges in the conventional SOAP session consists of the message exchange cost ($t_2 n$).

$$C_{soap} = t_2 n \quad (2)$$

t_1, t_2, and O_c are the parameters that depend on the size of message. O_a might or might not depend on the size of message.

Even though we define the overhead for designing the HHFR Schema, it is zero value for the current framework because it is implemented as an ad-hoc method.

5.2 Configurations

The host running our benchmark applications is installed on the server machine and uses Axis as a Web Service container. HHFR Clients are installed on the Treo 600 machine.

`System.currentTimeMillis()` call of MIDP [22], which has 10 millisecond precision, is used for timing measurements. Table 1 contains a summary of testing environments.

Table 1 Summary of Machine Configurations

Axis and Context-store: GridFarm 8	
Processor	Intel® Xeon™ CPU (2.40GHz)
RAM	2GB total
Bandwidth	100Mbps
OS	GNU/Linux (kernel release 2.4.22)
Java Version	Java 2 platform, Standard Edition (1.5.0-06)
SOAP Engine	AXIS 1.2 (in Tomcat 5.5.8)

Service Client: Treo 600	
Processor	ARM (144MHz)
RAM	32MB total, 24MB user available
Bandwidth	14.4Kbps
OS	Palm 5.2.1.H
Java Version	Java 2 platform, Micro Edition CLDC 1.1 and MIDP 2.0

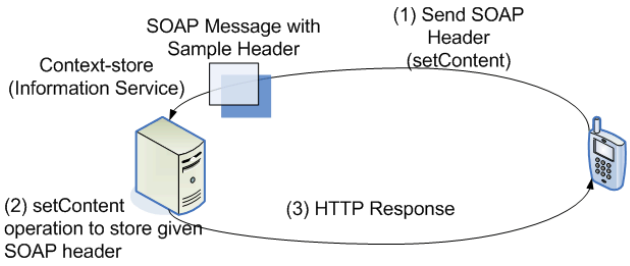

Figure. 7. Context-store operation overview

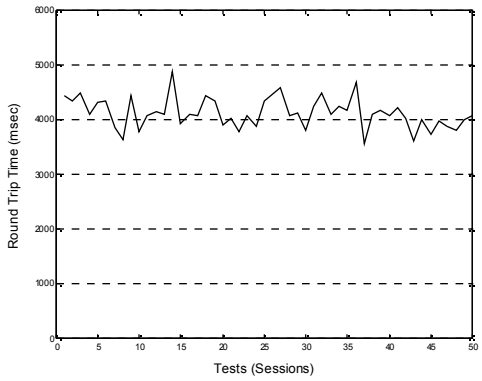

Figure. 8. Round Trip Time of Context-store Accession Tests

5.3 Parameter Evaluation

We perform tests to get O_a by measuring the Round Trip Times (RTTs) of Context-store

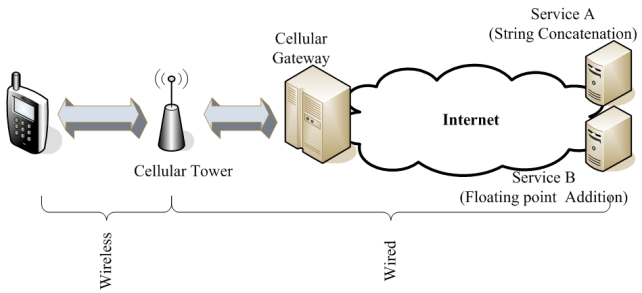

Figure. 9. Connection Setup for Performance Evaluation

(Information Service) accession. The message used in the test is a sample SOAP header document in a WS-Reliable Messaging [23] Specification. The size of the WSRM header is 847 bytes and the size of the entire SOAP request message for accessing the Context-store is 1.58KB.

The test setup is depicted in figure 7, and figure 8 shows our results.

To get t_1 and t_2 parameters, we measure a total session time for the given applications. We benchmark two application performances. The first application is a string concatenation service. It produces a single string, concatenated from all strings in an array – a pure-text data domain. The second application is a floating point number addition service that calculates the summation of all floating point numbers in an array in a message. The floating point numbers represent a float data domain, where the conventional Web Services message processing includes a float-to-text conversion that consumes many process cycles. We also measure the conventional SOAP-based Web Services performance of two given applications with the same setup, and this gives us the t_2 parameter measurement.

As depicted in figure 9, the connection setup for the test is partly wireless and partly wired. The test scenario is as follows: 1) a service client prepares a message with a given array size. 2) it sends one message to a service provider. 3) the service provider processes the message and returns a result in a message to the client. 4) repeat step 1-3 for each

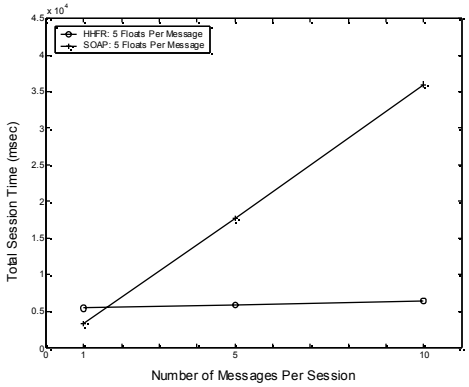

Figure. 11. Comparisons of floating point number addition test results (5 Floats Per Message)

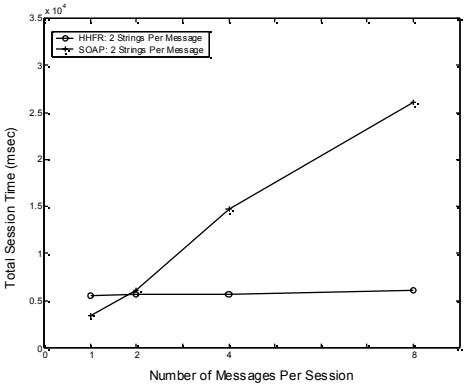

Figure. 14. Comparisons of string concatenation test results (2 Strings Per Message)

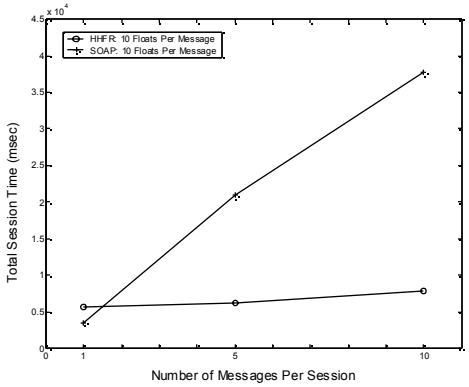

Figure. 12. Comparisons of floating point number addition test results (10 Floats Per Message)

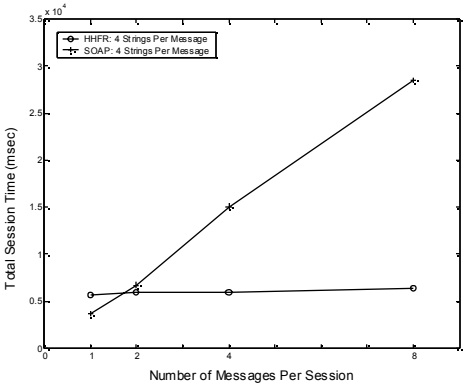

Figure. 15. Comparisons of string concatenation test results (4 Strings Per Message)

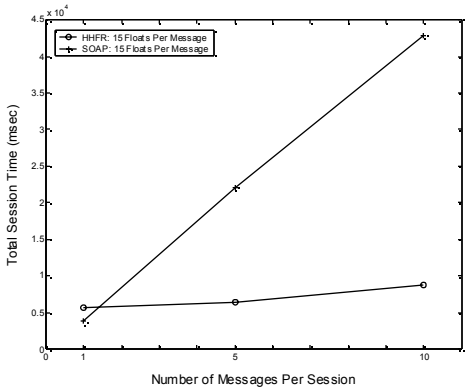

Figure. 13. Comparisons of floating point number addition test results (15 Floats Per Message)

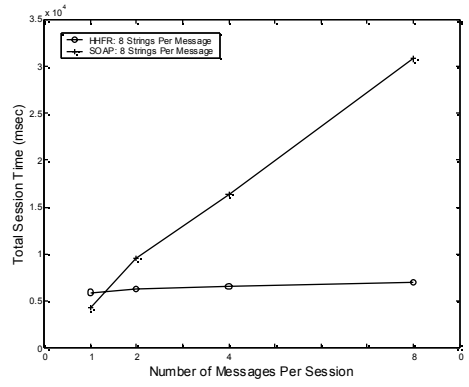

Figure. 16. Comparisons of string concatenation test results (8 Strings Per Message)

Table 2 Total Session Time (msec) of String Concatenation Application Tests over HHFR

Message Size	Number of Messages Per Session			
	n = 1	n = 2	n = 4	n = 8
2 Strings	5580	5710	5730	6100
4 Strings	5660	6010	6030	6470
8 Strings	5950	6310	6530	7070
64 Strings	11890	13570	19730	29660

Table 3 Total Session Time (msec) of String Concatenation Application Tests over SOAP

Message Size	Number of Messages Per Session			
	n = 1	n = 2	n = 4	n = 8
2 Strings	3440	6060	14720	26040
4 Strings	3670	6710	15050	28490
8 Strings	4260	9590	16390	30790
64 Strings	6510	15640	28020	54200

Table 4 Total Session Time (msec) of Floating Point Number Addition Application Tests over HHFR

Message Size	Number of Messages Per Session			
	n = 1	n = 5	n = 10	n = 100
5 Floats	5440	5790	6500	20360
10 Floats	5650	6210	7870	22780
15 Floats	5700	6500	8790	24510

Table 5 Total Session Time (msec) of Floating Point Number Addition Application Tests over SOAP

Message Size	Number of Messages Per Session			
	n = 1	n = 5	n = 10	n = 100
5 Floats	3260	17620	35960	330750
10 Floats	3480	20890	37720	353340
15 Floats	3800	22100	42790	387840

message.

Table 2 and 3 show the benchmarking results of the string concatenation application. The comparisons of the results are depicted in figure 14, 15, 16. Table 4 and 5 show the benchmarking results of the floating point number addition application. The comparisons of the results are depicted in figures 11, 12, and 13. The total session time in the tables and graphs include the negotiation overhead (O_b). We plot only the partial results of the total session time of the SOAP test on the graph since some results are too big and would make the comparison graphs less meaningful.

In addition to measuring t_1 and t_2, we also measure RTTs to get the O_b or negotiation overhead.

The size of the HHFR Schema used in the test is 326 bytes, and the size of the entire SOAP request message for negotiation is 1.07 KB. The results are shown in figure 10. As table 6 shows, values for overhead parameters are similar. Presumably, the similarity comes from the fact that the physical constraints in the mobile environment are the major factors in the overhead. So we can differentiate the overhead parameters of mobile environments and conventional Web Services environments as O (mobile) and O(ws). For example, O_a(ws), derived from measurements independent from our experiments in this paper shows O_a(ws) = 20 msec.

Table 6 Overhead Parameters and Values

Parameter	Value
O_a(mobile)	4120 (msec)
O_b(mobile)	4800 (msec)

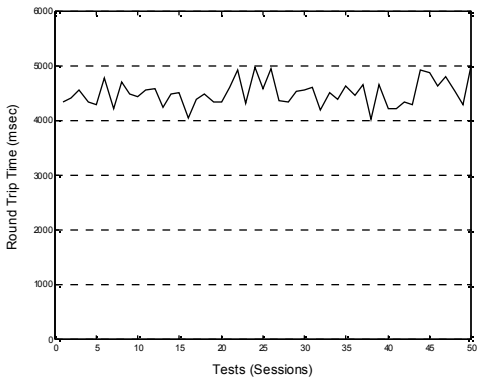

Figure. 10. Round Trip Time for the Negotiation Stage

5.4 Performance Comparisons

We compare the performance of HHFR and conventional SOAP message exchanges using our proposed performance model and the parameters we get from the benchmark tests. Table 6 shows the overhead parameters and values. We get t_1 and t_2. Then we use those numbers to calculate the breakeven points n_{be}.

The parameters for each application are different. So we calculate both breakpoints. Using the test result of 10 floats per message, we get the following

parameters:

$t_1 = 250$, $t_2 = 3780$, $O_a = 4500$, $O_b = 4800$, $O_c = 0$

Then the break even point is:

$$t_1 n_{be} + O_a + O_b + O_c = t_2 n_{be}$$
$$n_{be} = 2.88$$

Thus, if we have more than three messages per floating point number addition session, the HHFR performs more efficiently than the conventional SOAP.

Similar to the floats case, to calculate a breakeven point for the string concatenation application, we use the test result of 4 strings per message. Thus, we get the following parameters:

$t_1 = 120$, $t_2 = 3580$, $O_a = 4500$, $O_b = 4800$, $O_c = 0$

Then the break even point is:

$$n_{be} = 2.68$$

If we have more than three messages per string concatenation session, the HHFR performs more efficiently than the conventional SOAP.

VI. Conclusion

We investigate a novel approach to Web Services performance, in which the system 1) separates message content from XML syntax, 2) chooses a preferred representation, and 3) exchanges messages in a streaming fashion. This approach, implemented as an overall system framework, can increase the efficiency of message exchange, since applications can avoid textual conversion and conventional serializing/parsing. Having a reduced message size by storing the static parts of the message and having an optimal representation helps applications save network bandwidth. The streamed messages are not directly self descriptive. However the combination of the message and the negotiation captured in the Context-store is self descriptive.

Our presentation is particularly focused on applications in mobile computing environments, but the approach may be more general i.e. we expect our approach performs better than the conventional SOAP based Web Service messaging in a stream based application domain over standard internet connections. The bandwidth savings by reducing size of messages is not as big as mobile case's. However savings from avoiding conventional XML parsing/serializing processing is still significant. We also investigate this non-mobile case [25].

We compare our system with the conventional SOAP communication model, and as expected, empirical results based on our performance model show substantial performance gains by adapting the approach. As well, we demonstrate how to find the breakeven point at which our HHFR architecture overtakes conventional SOAP messaging for controlled applications. The breakeven points are different from application to application, but the general methodology can be applied to any application domain that defines its messaging style as conversational or streaming.

It should be noted that there is a binary XML group in W3C and the research we presented here is input to their deliberations. A key point of architecture is a negotiation architecture which allows multiple solutions that are agreed by a service provider and a client. At moment, we would say mobile access to Grid is not "standard" i.e. confined to research. Finally it doesn't require any changes to Axis2 as this supports SOAP Infoset and our approach is fully compatible with Infoset.

References

[1] I. Foster, C. Kesselman, J. Nick, S. Tuecke, " The physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," IEEE Computer Vol. 35, pp. 37-46, 2002.

[2] World Wide Web Consortium, " Simple Object Access Protocol (SOAP) 1.1," http://www.w3c.org/TR/soap/

[3] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", In *Proceedings of 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11 2002,* July 2002.

[4] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon, "Requirements for and evaluation of RMI protocols for scientific computing," In *Proceedings of Supercomputing 2000 (SC2000)*, Dallas, TX, Nov. 2000.

[5] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E Pelegri-Llopart, "Fast Web Services", Aug. 2003, http://java.sun.com/developer/technicalArticles/WebServices/fastWS/

[6] P. Sandoz and S. Pericas-Geertsen, "Fast Infoset @ Java.net," In *Proceedings of XTech 2005*, http://www.idealliance.org/proceedings/xtech05/papers/04-01-01/

[7] World Wide Web Consortium, "Report from the W3C Workshop on Binary Interchange of XML Information Item Sets", Sep. 2003, http://www.w3.org/2003/08/binary-interchange-workshop/

[8] World Wide Web Consortium, "XML Information Set", http://www.w3.org/TR/xml-infoset/

[9] J. H. Gailey, "Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation", Dec. 2002, http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx

[10] D. Brutzman, and A. D. Hudson, "Cross-Format Schema Protocol (XFSP)", Sep. 2003, http://www.movesinstitute.org/openhouse2003slides/XFSP.ppt

[11] Global Grid Forum 15 Community Activity, "Web Services Performance: Issues and Research" http://www.gridforum.org/GGF15/ggf_events_schedule_WSPerform.htm

[12] E. Serin and D. Brutzman, "XML Schema-Based Compression (XSBC)", http://www.movesinstitute.org/xmsf/projects/XSBC/03Mar_Serin.pdf

[13] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (SIGMOD2000), pp. 153-164, 2000.

[14] P. Deutsch, "GZIP file format specification version 4.3," May 1996, http://www.ietf.org/rfc/rfc1952.txt.

[15] M. Beckerle, and M. Westhead, "GGF DFDL Primer", http://www.gridforum.org/Meetings/GGF11/Documents/DFDL_Primer_v2.pdf

[16] R. Williams, "XSIL: Java/XML for Scientific Data", Jun. 2000, http://www.cacr.caltech.edu/projects/xsil/xsil_spec.pdf

[17] World Wide Web Consortium, "XML Schema Definition (XSD)," http://www.w3.org/XML/Schema

[18] B. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischinkinky, E. Newcomer, J. Webber, and K. Swenson, "Web Services Context (WS-Context)," http://www.arjuna.com/library/specs/ws_caf_1-0/WS-CTX.pdf

[19] L. Peterson and B. Davie, *Computer Networks: A System Approach 3rd Edition*. Morgan Kaufmann Publishers, 2003

[20] M. S. Aktas, G. C. Fox, and M. Pierce, "Managing Dynamic Metadata as Context," in Proceedings *of The 2005 Istanbul International Computational Science and Engineering Conference (ICCSE2005)*, Data Istanbul, Turkey, Jun. 2005.

[21] Community Grids Lab, "Extended UDDI and Fault Tolerant and High Performance Context Service," http://www.opengrids.org

[22] Sun Microsystems, Mobile Information Device Profile (MIDP) http://java.sun.com/products/midp/

[23] R. Bilorusets et al., "Web Services Reliable Messaging Protocol (WS-ReliableMessaging), Feb. 2005, ftp://www6.software.ibm.com/software/developer/library/ws-reliablemessaging200502.pdf

[24] Global Grid Forum, Open Grid Services Architecture Working Group (OGSA-WG), https://forge.gridforum.org/projects/ogsa-wg1.

[25] Galip Aydin, Ahmet Sayar, Harshawardhan Gadgil, Mehmet S. Aktas, Geoffrey C. Fox, Sunghoon Ko, Hasan Bulut, and Marlon E. Pierce, "Building and Applying Geographical Information System Grids," in a special issue of Concurrency and Computation: Practice and Experience.

Ms. Ref. No.: FGCS-D-06-00104
Title: Optimizing Web Service Messaging Performance in Mobile Computing
Future Generation Computer Systems

Dear Dr. Peter Sloot,

I tried to address all the comments. The English grammatical and other errors are fixed and I've made a list of the explanations on a point by point basis how each comment from each reviewer has been addressed.

Reviewer 1's comments:
General Comments:

1. *Technically this work is very competent and of great interest, however, the written style detracts from the overall quality of the paper.*
2. *The English grammar is poor or careless throughout the paper.*
3. *The authors frequently miss out the definitive article, like an "a" or "the" before a word, and also add an "s" after words, when it is not necessary.*
4. *Throughout the paper various words, start with capital letters, though they are not the first word in a sentence or a name.*
5. *Sentence construction and wording often makes the readability of parts of the paper difficult. For example, the last paragraph of section 3.3 should be rewritten.*

Particular Comments:

1. *There are too many grammatical and other errors, such as missing or wrongly used word, to point them all out.*
2. *There is an interchange through the paper of the "Web Services" with "Web Service". Often the authors are referring to Web Services-based technologies, but instead refer to "Web Service". This mismatch should be resolved throughout the paper.*
3. *The first sentence in the abstract is poorly constructed "The performance and efficiency of Web Services can be greatly increased in conversational and streaming message exchanges by streaming the message exchange paradigm." This sentence should be reworded.*
4. *Words like "doesn't" and "can't" should be expanded to their full form.*
5. *Section V - Evaluation "How much performance gains we have." Poor English rewrite.*

I fixed the grammatical errors in the paper including the ones commented by reviewer 1. The revised manuscript is reviewed by a professional English reviewer.

Reviewer 2's comments:

1. The method proposed is not compared to any of the approaches listed in section 2, so I've no idea if the method is better or worse than those already out there.

Our approach is an "overall system framework approach" so the approach is not competitive, but complementary to other approaches which focus on optimizing message representations. We revised the section 2 of manuscript to have this information.

2. The method doesn't support all SOAP types.

HHFR design accommodates all SOAP types. However, the current version of HHFR prototype implementation supports limited SOAP type mapping.

3. The method requires alteration of both the Web Services server and client. This being the case, to be successfully adopted the solution would need to be put forward as some standard, so that when its utility is proven it could be implemented in all web service libraries. This doesn't appear to be the case, although some of the competitors are going down this route. How do you propose to get people to use this solution over the others out there?

It should be noted that there is a binary XML group in W3C and the research we presented here is input to their deliberations. A key point of architecture is a negotiation architecture which allows multiple solutions that are agreed by a service provider and a client. At moment, we would say mobile access to Grid is not "standard" i.e. confined to research. Finally it doesn't require any changes to Axis2 as this supports SOAP Infoset and our approach is fully compatible with Infoset. We revised the conclusion section.

4. I'd like to see how this method performs over standard internet connections too.

We expect our approach performs better than the conventional SOAP based Web Service messaging in a stream based application domain over standard internet connections. The bandwidth savings by reducing size of messages is not as big as mobile case's. However savings from avoiding conventional XML parsing/serializing processing is still significant. We also investigate this non-mobile case. We revised the conclusion section and added a reference 25.

5. One area that's not addressed at all is the impact of the proposed method on security. Some of the competitors do consider this.

Again, our approach is complementary to security. We are planning to extend our architecture to use WS-Secureconversation or any security specification which is supported by Web Service community.

Sincerely,

Sangyoon Oh

*** Biographies (Text)**

Sangyoon Oh (8128560751 ohsangy@indiana.edu)
Oh received the B.E in Mechanical Design from Sungkyunkwan University, Korea in 1995, and M.S. in Computer Science from Syracuse University, USA. He is currently a Ph.D. candidate in Computer Science Department and a research assistant at Community Grids Laboratory of the Pervasive Technology Laboratories at Indiana University. His main research interests include mobile computing, Web Service technology, Grid systems, and information representation.

Geoffrey Charles Fox (8122194643 gcf@indiana.edu)
Fox received a Ph.D. in Theoretical Physics from Cambridge University and is now professor of Computer Science, Informatics, and Physics at Indiana University. He is director of the Community Grids Laboratory of the Pervasive Technology Laboratories at Indiana University. He previously held positions at Caltech, Syracuse University and Florida State University. He has published over 550 papers in physics and computer science and been a major author on four books. Fox has worked in a variety of applied computer science fields with his work on computational physics evolving into contributions to parallel computing and now to Grid systems. He has worked on the computing issues in several application areas – currently focusing on Earthquake Science.