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 can be greatly increased in conversational and streaming message exchanges by streaming the message exchange paradigm. In this paper, we describe our design and implementation of a novel approach to the message exchange optimization. This area is particularly important to applications in physically constrained mobile computing environments but potentially has other application areas. The verboseness of XML-based SOAP representation imposes possible overheads in mobile Web Service applications. We separate data content from the syntax and use streaming message exchanges. The redundant or static massage parts are stored in shared metadata space – the Context-store. The streamed messages 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 result shows that our framework outperforms the conventional Web Services in 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 became a backbone of Grid computing because of its interoperability across the diverse services/application 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 communicating.

But the verbose nature of current XML-based SOAP [2] requires an alternative and more efficient solution for message exchanges in mobile environment, which holds many physical constraints like limited processing and batter power and slow and intermittent connection. The conventional SOAP communication model possesses major characteristics that may affect messaging performance. Serializing and de-serializing SOAP message consumes lots of resources. In-memory representation, for example floating point number, must be converted from and to textual format of SOAP message, which is an expensive process for limited mobile computing. Also, the message size is increased substantially by adding descriptive tags of XML syntax and it is another problem for narrow mobile connections.

High performance SOAP encoding is an open research area [4-6]. There have been many investigations to address a performance issue of mobile Web Service and to provide solutions. But these proposals and solutions tackle small pieces of the problem, rather than providing the system level solution. In this paper we describe our novel architecture for increasing a performance of message exchanges in mobile Web Service environments. Our Handheld Flexible Representation (HHFR) architecture provides a complete system from a representation of the message to the use of dynamic metadata repository for guaranteeing semantic consistency. The streamed messages are self descriptive when it is 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 methods overtake the conventional SOAP messaging.

We organize this paper as follows. In Section 2, we discuss background works. Section 3 reviews 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 Binary Interchange of XML Information Item Sets (Infoset) [8] is the result of the increasing demand of binary form of XML-based communication. The report includes conclusion of workshop meeting on September 2003 as well as several dozens of position papers from various institutes [5, 9, 10]. The purpose of the workshop was to study methods to compress XML documents and transmit pre-parsed and schema specific object. It identified requirements of binary XML Infoset, for examples a) maintaining universal interoperability, b) producing a generalized solution that is not limited to a specific application domain, c) reducing process time including a data binding time, and d) negotiation - fall back to XML/SOAP text format if receiver can't understand binary. Web Service performance has been more recently reviewed at the 15th Global Grid Forum workshop (GGF 15) [11].

We divide current approaches of expediting Grid/Web Service communication into following categories. First, most proposals that follows the XML Binary Characterization of the W3C have the goal of producing a self-contained alternative to an XML message, 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], 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 [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] that achieves better ratio than conventional compression utilities like GZip [14] doesn't improve performance much because of the additional layer of processing, compression and decompression. Information external to a message is needed to interpret it.

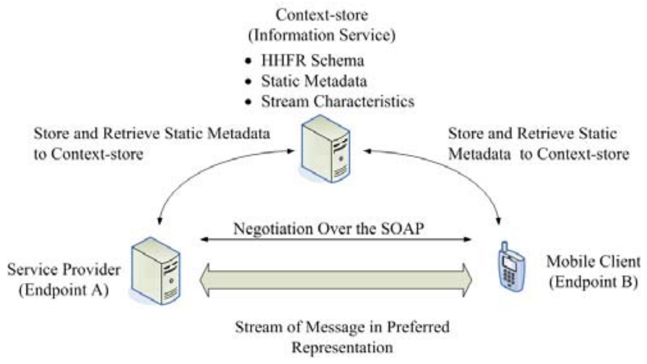
Context-store
(Information Service)
• HHFR Schema
• Static Metadata
• Stream Characteristics
Store and Retrieve Static Metadata to Context-store
Store and Retrieve Static Metadata to Context-store
Negotiation Over the SOAP
Service Provider (Endpoint A)
Mobile Client (Endpoint B)
Stream of Message in Preferred Representation

Figure. 1. HHFR Architecture Overview

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 independent of their physical representation. It supports most of types that XML Schema Definition [17].

III. The Design principles

The key design goal of HHFR architecture is optimizing conventional SOAP messages. Smaller size messages reduce the transit time of message and this 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 overhead that is imposed from verbose nature of SOAP. The types and structures of a SOAP message is syntax, which make the message content descriptive. We achieve optimized representation of message contents by separating message contents from its syntax and streaming them in preferred representation. Figure 1 depicts the overview of HHFR architecture.

3.1. Replacement of XML Syntax With Optimized Representation

HHFR provides 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 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 optional headers and a body. The architecture handles static information (unchanging headers) of 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 is program instruction or data. Comparing to the individual message conversion approach that converts SOAP message 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 preferred format is not self-descriptive, but the Data Model gotten from the data description uses. The relationships between data formats and representations are depicted in figure 2.

Among alternative, a binary representation increases performance of HHFR architecture in several reasons. First, we can save the bandwidth of message exchanging. Since the descriptive tag of XML syntax increases the size of exchange data, having content data in binary format could save as high as a factor of ten if the message structure is especially redundant – for example in the case of array. A very simple message with a single text

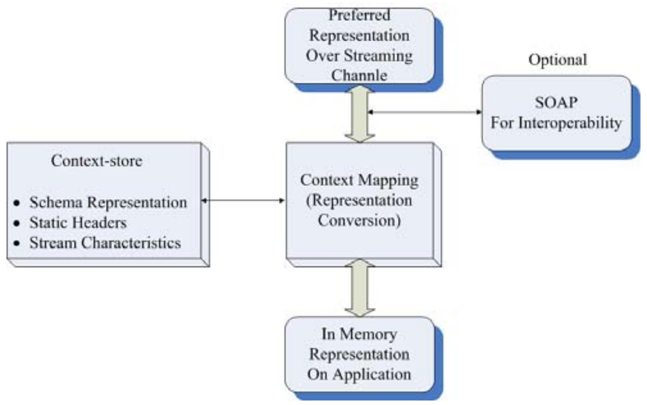
Preferred Representation Over Streaming Channle
Optional
SOAP For Interoperability
Context-store
• Schema Representation
• Static Headers
• Stream Characteristics
Context Mapping (Representation Conversion)
In Memory Representation On Application

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

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 expensive process, especially for relatively low-powered mobile devices.

3.2. Focus On Conversational Message Exchanges

HHFR works best for the Web Services, where two participating endpoints exchange a stream of messages like a conversation. For applications using a specific service, message 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 contents in HHFR schema can be transmitted once, and rest of the message in the stream has only 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 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, HHFR architecture uses the non-self-contained representation to exchange messages. So data, exchanging 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 the essential role of the negotiation. It is well known fact that Data streaming can increase the message exchange performance in Web Services. So some investigate 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 HTTP persistent connection, which is not always available for a network protocol implementation on all mobile devices and all cellular networks.

The HHFR architecture provides fast communication channel. The channel can be either the same channel that is used for conventional SOAP message exchange or a separate channel to the one. If the fast communication option is implemented as a separate channel, it is efficient in its performance and also flexible to be implemented in various ways including asynchronous messaging scheme. 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 is achieved in higher interoperability, but also required to modify or adapt transport layer implementation of a SOAP Server.

Two participating endpoints negotiate all the issues above: let corresponding endpoint know the data format by exchanging description file, fast communication channel information to set up the connection, and related quality of service issues.

3.4 Context-store (Information Service)

WS-Context [18] compliant Information Service is message-based interface to a DB. As we discussed, the use of a Context-store reduces the bandwidth usage, but it also ensure the system semantically consistent. Note that the streamed messages are not directly self descriptive. However the combination of

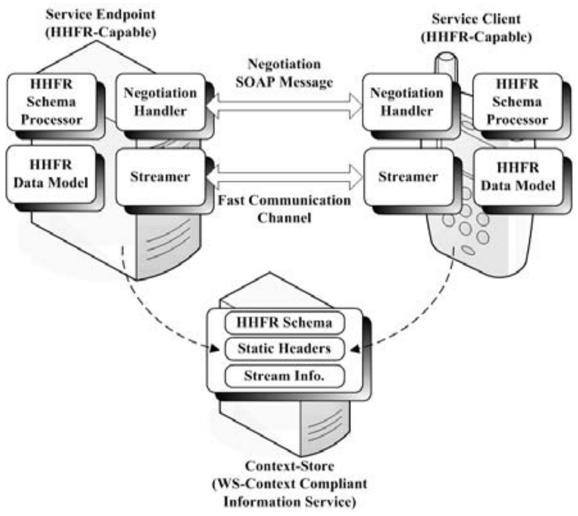

Figure. 3. Simple Overview of Implementation

the message and the negotiation captured in the Context-store is self descriptive. For example, it provides a fault-tolerance feature: when the service endpoint is out-of-service, the service gets the required context from the Context-store after the recovery. The use of the Context-store guarantees the system and participants semantically consistent: if there is a third party who audits a session, it gets contexts from the Context-store and understands a session by filling up the missing SOAP parts saved at 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 HHFR Schema and processor, 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 this chapter, and a service endpoint – a negotiation responder – sends an output data description. 3) Two endpoint use second transport channel for message exchange where they stream messages. Messages in the stream are in the form of 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 optimizing 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 mobile environment, and a metadata repository. The overview of implementation is depicted in figure 3.

4.1. Negotiation Scheme

A normal HHFR session is starting with a negotiation stage, where two endpoints exchange negotiation SOAP message. By design, a negotiation stage is essential to establish characteristics of following stream. During the stage, a service endpoint returns characteristics that are suggested by a negotiation initiator and selected and confirmed by the service endpoint. In the prototype implementation, the 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 the 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 uses a conventional SOAP based Web Service communication. The client (the SOAP initiator) must fall back if it receives a SOAP fault, which means the responding service doesn't have 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 contents 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 HHFR schema is similar to that of

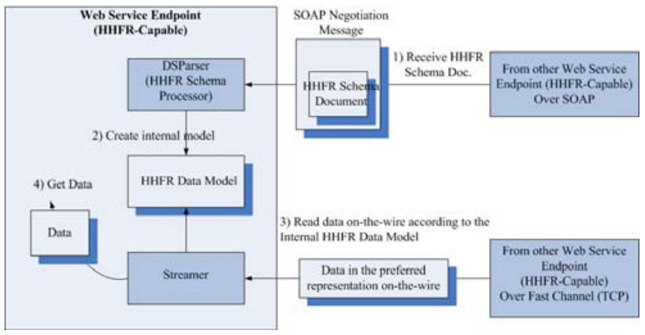
Web Service Endpoint (HHFR-Capable)
SOAP Negotiation Message
DSParser (HHFR Schema Processor)
HHFR Schema Document
1) Receive HHFR Schema Doc.
From other Web Service Endpoint (HHFR-Capable) Over SOAP
2) Create internal model
HHFR Data Model
4) Get Data
Data
3) Read data on-the-wire according to the Internal HHFR Data Model
Streamer
Data in the preferred representation on-the-wire
From other Web Service Endpoint (HHFR-Capable) Over Fast Channel (TCP)

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

DFDL: the HHFR Schema describes 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 limited number of simple type built-in to XML Schema. They are `string`, `int`, `byte`, `float`, `Boolean`. Current version of HHFR Schema doesn't support user-defined `sympleType`. The `complexType` element of the HHFR Schema can have mixed content, but can not have simple content and empty content. So we declare `complexType` element 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 message as depicted as step 1) of figure 4, as an input and produces a internal HHFR Data Model as an output as depicted as 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 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 popular design style in many data marshalling implementation. Compare to more efficient compiled-style stub [19], which is popular in many client and server RPC implementation, the interpret-style stub is more flexible to dynamic representation of input data. The stub doesn't need to be re-complied to different data representation. The stub reads and writes message packets, which is a unit of message in a preferred representation, through switch statements. In the prototype, the binary representation that is a sequence of byte is a default representation format for the message packet.

4.3. Data Streaming

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

A fast communication channel of the HHFR Prototype provides an alternative to the default HTTP communication method that is asynchronous and optimized. 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.

The fast communication channel layer for the TCP receptor is shown in figure 5. On the service provider, `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.

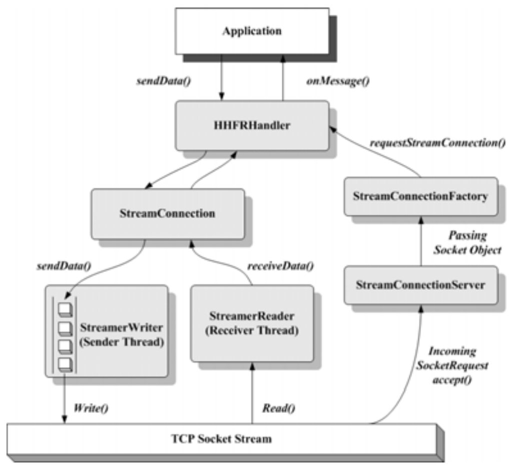

Figure. 5. Fast Communication Channel Layer

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 transitory metadata needed to describe distributed shared information. The Information System, Fault Tolerant High Performance Information Service (FTHPIS) [20 - 21] which use and extend 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 endpoint. This way, the size of SOAP messages is being minimized to make the service communication much faster. The redundant parts of a SOAP message can be considered as XML fragments which are encoded in every SOAP message exchanged among two services. These XML elements are stored as "context", i.e. metadata associated to a conversation", into the Information Service. Each context is referred with 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, which in turn reduces the size of the message 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

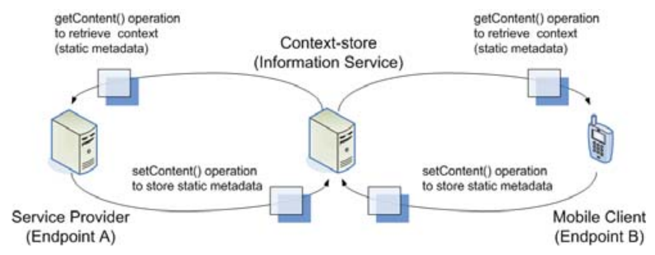

Figure. 6. Context-store operation overview

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 can be called whenever context needs to create, update, or retrieve context in Context-store (Information service). The Context-store client 1) first, create `ContextServiceHandler` object with the Context Service URl, 2) second, store given context of any type paired with an unique identifier, and 3) retrieve context. `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 HHFR runtime system. We assume following basic system parameter 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 HHFR session consists of message exchange cost (t_1n) and overheads ($O_a + O_b + O_c$).

$$C_{hhfr} = t_1n + O_a + O_b + O_c \qquad (1)$$

The cost of message exchanges in the conventional SOAP session consists of message exchange cost (t_2n).

$$C_{soap} = t_2n \qquad (2)$$

t_1, t_2, O_c are the parameters that depend on the size of message. O_a could depend on the size of message.

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

5.2 Configurations

The host running our benchmark applications are installed on the server machine and they use 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

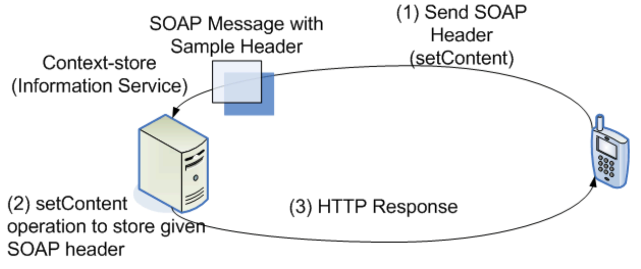
SOAP Message with Sample Header
(1) Send SOAP Header (setContent)
Context-store (Information Service)
(2) setContent operation to store given SOAP header
(3) HTTP Response

Figure. 7. Context-store operation overview

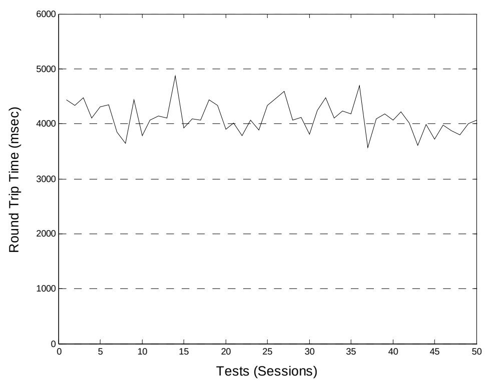

Figure. 8. Round Trip Time of Context-store accessing Tests

5.3 Parameter Evaluation

We perform tests to get O_a by measuring Round Trip Times (RTTs) of the Context-store (Information Service) accessing. The message used in the test is a sample SOAP header document in a WS-Reliable Messaging [23] Specification. The size of WSRM header is 847 bytes and the size of the entire SOAP request message for accessing Context-store is 1.58KB.

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

To get t_1 and t_2 parameters, we measure a total session time of 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

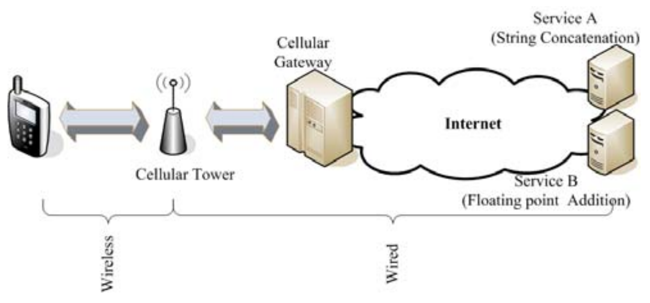
Service A (String Concatenation)
Cellular Gateway
Internet
Cellular Tower
Service B (Floating point Addition)
Wireless
Wired

Figure. 9. Connection Setup for Performance Evaluation

the summation of all floating point numbers of an array in a message. The floating point numbers are representing 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 Service performance measurement of two given applications with the same setup, which give 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 message.

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

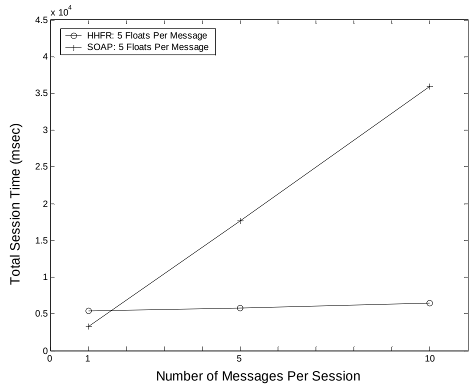

Figure. 11. Comparisons of floating point number addition test results (5 Floats Per Message)

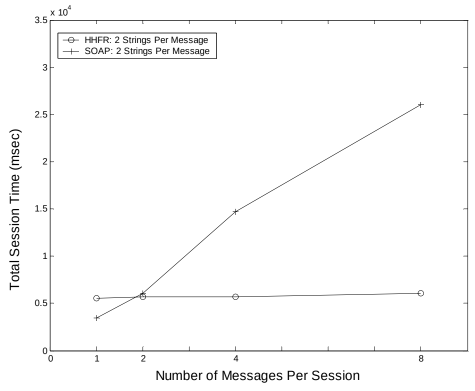

Figure. 14. Comparisons of string concatenation test results (2 Strings Per Message)

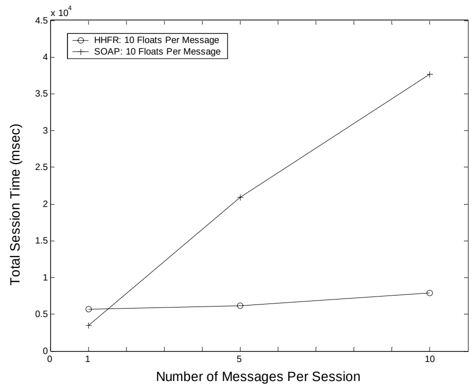

Figure. 12. Comparisons of floating point number addition test results (10 Floats Per Message)

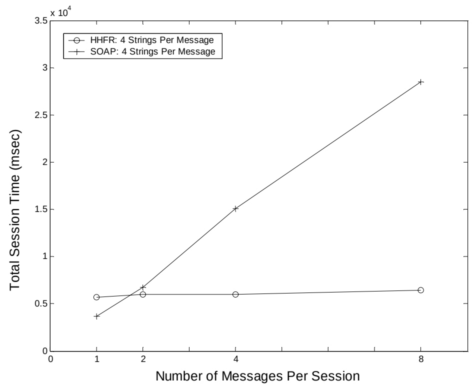

Figure. 15. Comparisons of string concatenation test results (4 Strings Per Message)

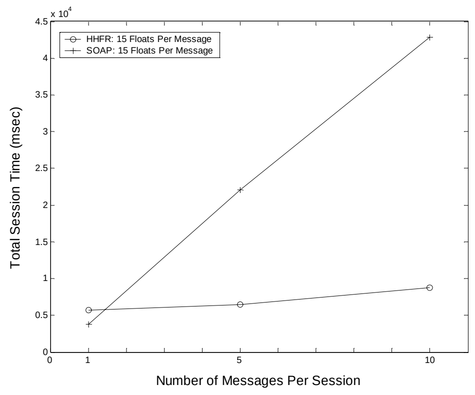

Figure. 13. Comparisons of floating point number addition test results (15 Floats Per Message)

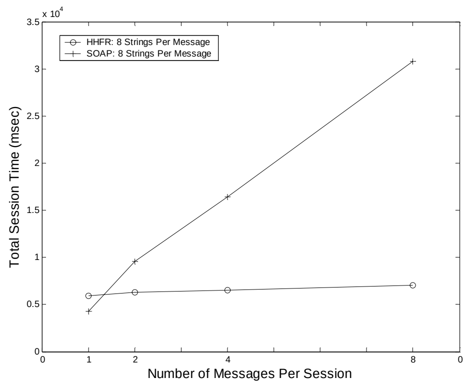

Figure. 16. Comparisons of string concatenation test results (8 Strings Per Message)

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

In addition to measuring t_1, t_2, we also measure the negotiation to get O_b overheads by measuring RTTs. The size of 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 physically constrained mobile environment is a major factor of the overhead. So we can differentiate overhead parameters of mobile environments and conventional Web Service environments as O (mobile) and O(ws). For example, O_a(ws) 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)

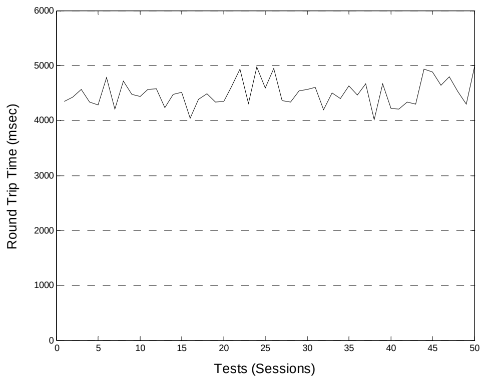

Figure. 10. Round Trip Time of Negotiation Stage

5.4 Performance Comparisons

We compare the performance of HHFR and conventional SOAP message exchange using our proposed performance model and parameters we get from the benchmark tests. The table 6 shows the overhead parameters and values. We get t_1 and t_2. Then using those numbers to calculate 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 following parameters:

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

Then 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 our floating point number addition session, HHFR performs more efficiently than the conventional

SOAP.

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

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

Then break even point is:

$$n_{be} = 2.68$$

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

VI. Conclusion

We investigate a novel approach to Web Service performance, in which the system 1) separates message contents from XML syntax, 2) chooses a preferred representation, and 3) exchanges messages in a streaming fashion. This approach implemented as a single complete system can increase efficiency of message exchanging, since applications can avoid the textual conversion and conventional serializing/parsing. Reduced message size by storing static parts of message and having optimal representation helps applications save network bandwidths. 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.

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

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.