HHFR: A new architecture for Mobile Web Services Principles and Implementations

Sangyoon Oh and Geoffrey C. Fox
Community Grids Laboratory
501 N Morton St. 222, Bloomington, IN, 47408, USA
{ohsangy, gcf}@indiana.edu

Abstract

By combining mobile computing and Web Services technologies, pervasive computing expects more portability and location transparency for accessing information in anytime from anywhere. Though, a direct integration of two technologies imposes performance limitations because of XML's verbose nature and physical limitations of mobile computing. We present our new architecture design and implementations, Handheld Flexible Representation (HHFR) in this paper. The architecture provides alternative representations other than XML-based SOAP and fast communication transport options. The negotiation between two end-points using SOAP message sets up characteristics of following stream of messages. The benchmark results show the performance advantage of using the architecture when a session composes a sequence of messages.

KEYWORDS

Mobile Computing, Web Service, Stream of Message, Binary Representation

1. Introduction

Mobile computing gives a pervasive computing the way to access information *any-time* and *any-where* by its portability and remote connectivity. And Web Service technology gives a pervasive computing the way to interoperate remote resources and diverse services. Because of their importance in the pervasive computing, it is a no-big surprise that there are so many recent researches of adopting mobile computing as one of flexible platform of Web Service technology.

Even a half a decade ago, there is just handful of people who access remote information from their mobile device. The information access from mobile devices, however, has became easier than ever recently with the help from advanced mobile devices and widespread availability of packet-switched, always-on cellular phone networks. There are many projects that try to adopt smart-phones and cellular phone with data connections as major elements in Web Services, since huge synergy effect of interoperability and removing physical-location constraints are expected.

However, the verbose nature of current XML-based SOAP [1] approach imposes performance limitations in integrating mobile computing applications and conventional Web Services directly. SOAP achieves ubiquity by using highly universal XML as a form of data exchanging between disparate and distributed computing resources. Though, XML-based SOAP possesses three major characteristics that may affect SOAP performance. First, the in-memory data model must be converted to textual format to build a SOAP message object and to extract information from it. Secondly, because of inevitable mobile computing characteristics – high latency, narrow bandwidth, limited computation, and small memory space, SOAP message processing consumes valuable resources [2]. Finally, mobile communications suffer from a larger data size by XML's descriptive tags and structure. It is usually

not a problem on the powerful wired networks, although the bandwidth is pricey in mobile networks.

High performance SOAP encoding is an open research area [3], [4], and [5]. Web Services in mobile environment is a benefited of the researches, since it also need to overcome the performance limitations because of its characteristics above. Even the small size, regular frequency message exchanges could cause performance overheads in such an environment.

In this paper, we present our new architecture design to achieve an optimized communication using binary message stream and a SOAP negotiation as well as the prototype implementation of the architecture and the performance benchmarks.

We organize this paper as follows: we describe an overview of *Handheld Flexible Representation* architecture design in section 3. And we illustrate detail implementations of the prototype in section 4. Section 5 presents a performance benchmark results. Conclusion is described in section 6.

2. Background

We see several notable projects from industry and academia that try to overcome performance limitations of current Web Services approach. Extreme! Lab researched the limits of SOAP performance for scientific computing where large data sets including arrays are common and the design of a SOAP implementation suitable for systems with small memory and bandwidth [3], [4]. Throughout the experiments, the result of research shows the major improvements from using schema-specific parser mechanism for arrays, persistent connection, and streaming of messages to prevent full serializing objects to determine the length. It also shows that the most serious overhead is conversion to textual form from in-memory float numbers. To resolve the limitations, they recommend using multiple communication protocol incorporating with a binary representation and fast protocols other than SOAP. The condition they are facing with the conventional Web Services is similar to the constraint of mobile computing because of its limited computing environment characteristics. Both need to overcome performance limitations of SOAP.

The report of the W3C Workshop [6] on Binary Interchange of XML Information Item Sets (Infoset) [7] 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], [8], and [9]. The purpose of the workshop is to study methods to compress XML documents and transmit pre-parsed and schema specific object. It identified the requirement of *binary XML Infoset*, for example 1) maintaining universal interoperability, 2) a generalized solution that is not limited to a specific application domain, 3) reducing process time including a data binding time, and 4) negotiation - fall back to XML/SOAP text format if receiver can't understand binary. The discussion leads W3C form XML Binary Characterization Working Group for further researches. Sun's Fast Web Services [5] and Fast Infoset project [10] specifies a representation of an instance of SOAP Infoset using binary encoding. They use Abstract Syntax Notation (ASN). 1. [11] to abstract encoded messages that may be encoded using it. The higher level protocols (WSDL [12] for contract definition of service etc.) remain unchanged, thus you could use standard SOAP-XML for development, and have a switch that turns on the binary protocol for production deployment.

W3C XML Protocol Working Group released the draft of Message Transmission Optimization Mechanism (MTOM) [13] and XML-binary Optimized Packaging (XOP) [14]. Combined together, the specifications are targeted to two data type - multimedia data that already have standardized formats, such as JPEG, GIF, and MP3 and data that includes digital signature. The XML encoding would damage the data integrity. XOP is an alternate serialization that looks like a MIME package. It avoids data binding overhead, though still preserves XML structure – tags. Thus, XOP and MTOM, which describes how XOP is layered into SOAP HTTP transport, still

possess a parsing issue inherited from SOAP/XML.

Cross Format Schema Protocol (XFSP) [9] is another project that serializes XML document based on schema. Initially it is motivated by the flexible definition of network protocols. It is written in Java and uses DOM4J model to parse the schema. With XML Schema-based Compression (XSBC) [15], XFSP provides binary serialization and parsing framework. Naval Postgraduate School provides lots of research on Streaming X3D documents in the XFSP framework.

Data Format Description Language (DFDL) [16] is a descriptive language that is proposed to describe a file or a stream in a binary format for Grid computing. Like Extensible Scientific Interchange Language (XSIL) [17], it is XML-based and comes with an extensible Java Data model. DFDL define the structure of data. 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. Also DFDL is designed to be processable through DFDL parser and data model. We designed the message format description of our Flexible Representation based on DFDL. In our Handheld Flexible Representation architecture, we define simple XML-schema based descriptive language and develop a language parser using XML Pull Parser (XPP) [18]. Our prototype implementation will not be in-depth like DFDL, though it will be enough to show advantages of our approach.

3. Architecture Overview

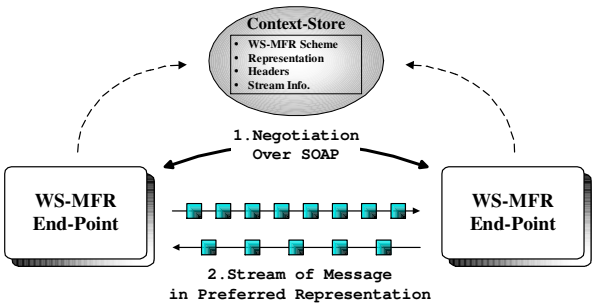

Figure. 1. Illustrated overview of HHFR architecture

In this section we present new software architecture, the Handheld Flexible Representation (HHFR) that is designed for an optimized and expandable communication in mobile web services. HHFR separates message contents from a SOAP message in *Angle-Bracket syntax* format, and service end-points in HHFR exchange separated-message contents in an optimized fashion. An overview of HHFR architecture design is depicted in Figure 1.

3.1. Design Overview

Major aspects of the architecture design are:

Optimized binary representation against verbose XML syntax: HHFR provides a communication options to exchange SOAP messages in a binary representation format by separating XML syntax of SOAP messages and SOAP message contents. Structured and typed conventional SOAP causes performance bottlenecks that are magnified in mobile environments. Separated data structure and types of SOAP body (payload) is negotiated in the beginning of a stream. XML Schema is used to characterize the syntax of SOAP Body.

Targeting a stream of messages: HHFR works best for the Web Services where two end-points exchange a stream of messages. Messages in a stream share the structure and type information of SOAP Body and the most of headers of messages are not changing in a stream session. Therefore, the structure and type as a form of XML Schema and headers can be transmitted only once and rest of the messages in the stream have only payloads. The 'streaming' of message is possible by introducing non-blocking message communications.

Context-store as a repository: In the HHFR architecture, a *context-store* module keeps static data from a message stream including unchanging-SOAP headers, XML Schema as a data representation, and stream characteristics that is captured in a *negotiation stage*. By saving SOAP headers and data representation to the

context-store and having optimized binary format messages, the architecture slims down a size of messages.

Interoperable Web Service architecture: Distinct from other ad-hoc solutions to the SOAP performance problem, the architecture doesn't change overall interoperability of existing Web Service standard. Our approach provides seamless integration of current Web Service applications by using conventional SOAP messages to setup an optimized representation and transport. When the other end-point claims it is not compatible, HHFR architecture fall back to the conventional SOAP communication.

3.2. Separation of Representation

The essential idea of HHFR architecture is an optimized representation and communication for two Web Service end-points while not sacrificing SOAP compatibility. We design HHFR architecture to provide an optimized data representation according to present communication environment. Options include binary and conventional SOAP representations.

3.2.1. XML and SOAP Infoset. The first thing to describe SOAP Infoset in our architecture is looking into what is XML Infoset, since SOAP is XML Based language and latest specification 1.2 is defined using the XML Infoset.

XML Infoset specification is released to help to define other languages that are based on the XML data model, yet an instant help from the specification goes to the application designing and developments, which manipulate data model with XML APIs. The model defined by XML Infoset is not tied up with any specific XML API, such as Document Object Model (DOM), Simple API for XML (SAX), and XML Pull Parser (XPP). Thus, the application development sets to free as far as it follows XML Infoset Specification. One of the possibilities that XML Infoset Specification opened is to have a parser that supports the binary form of XML.

In our architecture, we defined the data model based on SOAP Infoset. Consequently, the HHFR architecture is able to separate a representation – XML/SOAP syntax and SOAP message content without losing any message contents.

3.2.2. Binary Representations of SOAP Message. From the separation of representation, the architecture provides options to choose appropriate message representations such as binary and conventional SOAP representation for optimized Web Service communication environment.

The binary representation is a critical option to improve overall performance of HHFR architecture in several reasons. First, it reduces the size of exchanging message by removing verbose SOAP syntax – *Angle-Brackets*. The saving is maximized as a factor of 10s when a document structure is inevitably redundant – Array. A very simple message with a single text element can be reduced in its size in half. Hence it is always good to have reduced size document to exchange, even in a conventional computing environment, however mobile computing really need it because of its narrow bandwidth connection.

Also, by using binary format message, HHFR is able to avoid textual conversions. The architecture removes conventional encoding/decoding stage where the in-memory format is converted into a text format and vice versa. It is an expensive process especially for relatively low-powered mobile devices to convert non-textual data into and from textual data, which is required in SOAP syntax. Among non-textual data, floating point number conversion is the most costly one [3].

Finally, another benefit to have a binary representation of SOAP message in the architecture is leaving out parsing for each message. Since SOAP syntax requires a structured data model that satisfies Extended Backus-Naur Form (EBNF), a parsing process is needed to get information from the given documents. The binary representation – byte array format of SOAP message contents is chunks of continuous information that is defined by SOAP Infoset, which doesn't need to be parsed in conventional way. Rather, architecture provides another kind of information retrieval scheme,

stream reader and writer. It is done by established internal Data Structure model and packet reader and writer that reads and writes data to byte stream according to the Data Structure.

For the replacement of XML Syntax that we separate, the architecture utilizes XML Schema Definition (XSD). It is a recommendation of the World Wide Web Consortium (W3C) about how to describe the elements in an XML Documents formally. Originally it is for checking the validity of XML Documents; however it is also an abstract representation of an XML Document's relationship – the structure and characteristics including elements and data type. Nevertheless, there is an issue to be considered to use the XML Schema approach to define the syntax of SOAP for message stream. XML Schema doesn't guarantee to capture a static view of XML Document, such as an order of elements or attributes. For that reason, HHFR expects the initial sender end-point sends the fixed order data model that is expected on an ultimate receiver side. However, this is less concerned when the service is defined in RPC style because the operation, which is the service, has a function signature of the ordered parameters. Except the ordering issue, XML Schema nicely replaces the XML syntax in data interchanges.

3.3. Negotiation of Characteristics

A few design issues motivates an introduction of a negotiation stage: 1) to have alternative representation of SOAP messages, the representation of message and leaving out-SOAP Headers should be transmitted at the beginning of the stream 2) to setup the fast and reliable way of communication, the architecture should negotiate characteristics of stream.

3.3.1. Supporting Alternative Representation of SOAP Message. A stream of messages shares the same representation, which means that messages share the structure and types of SOAP Body parts. Schemas which represents the separated structure and types of both request and response message should be conveyed initially to tell the representation of following binary format messages in the stream. As well as the representation, the headers of SOAP messages are not changing mostly. Needless to say, there are headers that apply to the individual message, such as reliability related headers. Such headers are processed at the corresponding handlers. Except those, rests of headers that are the majority in many cases can be transmitted only once and used by during the stream. Both the representation and headers are archived in the context-tore.

Since we preserve the message contents in SOAP Infoset data model, HHFR is able to apply various representations other than binary that defined in XML Schema; it is able to send/receive messages in binary format as well as traditional SOAP message in formal SOAP syntax.

3.3.2. Negotiating Characteristics of Stream. A connection of mobile device has narrow bandwidth and high latency. Because of it, chunk overlaying and pipelined-send over HTTP 1.1 is studied to improve its performance [3]. 'Persistent connection' or 'Keep-Alive' features that are required for it, however, is not available for network protocol implementation on some mobile devices and some cellular network.

Because of above reasons, the HHFR architecture provides fast communication options as well as default HTTP transport, such as TCP and UDP where message contents in a binary representation is transmitted over in a stream fashion. Options provide asynchronous (also called non-blocking instead of HTTP callback mechanism) messaging scheme, so that the architecture can stream messages. The fast communication option looks similar to previous ad-hoc solutions to the Web Service Performance issue. Though, our architecture puts the fast transport implementation behind the SOAP communication transparently so that services in HHFR use conventional Web Service specifications seamlessly and loosely-coupled way without additional ad-hoc scheme.

The reliability is negotiated in the negotiation stage as well. For example, UDP transport is a simple high-performance datagram Internet Protocol, although UDP doesn't provide

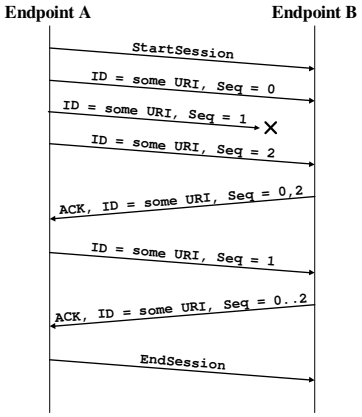

Figure. 2. Possible message exchange between two WS-RM end-points

reliability and ordering guarantee that TCP does. Datagram may be missing or arrived in out-of-order. Thus, the architecture design implements Web Service Reliable Messaging [20] (WS-RM) specification on UDP transport. Figure 2 depicts a possible message exchanges between WS-RM end-points over UDP or TCP transport.

The detail implementation of fast communication transport and reliability are presented in section 4.

3.3.3. Negotiation Stage. The issues discussed above are negotiated in an initial *Negotiation Stage* where a typical HHFR session starts with. The negotiation uses a conventional SOAP message, which makes the negotiation stage compatible with existing Web Service framework. The architecture design defines each item from the issues as an incremental element in a *Negotiation Schema*. Thus, negotiation handler

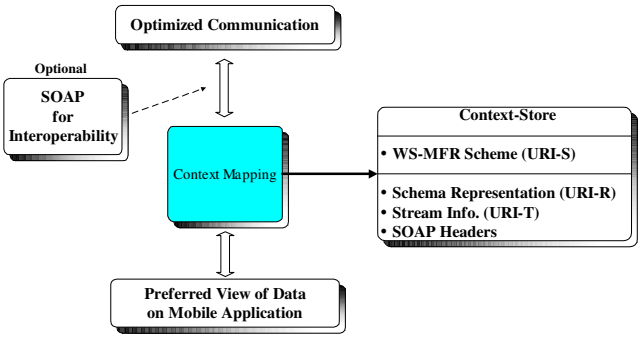

Figure. 3. Relationship of different forms of SOAP messages and their defining context

receives a negotiation SOAP message and it prepares a response SOAP message for negotiated items. A successful end of negotiation stage is lead to the message streaming stage, where streams reader and writer are set up for incoming and outgoing messages.

The negotiation stage is the only notable overhead we have in our architecture and it prevents us to use the architecture scheme in short-lived sessions, which have few messages exchanged.

3.4. Message Handling

The SOAP message has an outer-most element, SOAP envelope in its XML document, which is composed optional headers and a body – payload which contains a program instruction or data. The headers contain additional information for SOAP message, such as parsing instructions, security information, and routing/reliability information. The architecture handles static information of messages in the stream (un-changed headers) and dynamic information (payload and headers for individual messages) differently.

3.4.1. Handler for SOAP Header Processing. As discussed, the static/unchanged headers of SOAP message in the session (the stream) are archived in negotiation stages. Those headers applied to individual message are processed by appropriate handlers, which are transmitted as a part of optimized representation of SOAP message. The intension of using this possible additional information in the stream is negotiated as a part of the schema that represents a data structure of exchanging message format in the stream. When message in given representation enters or leaves HHFR architecture, the handlers process headers. The popular example can be a WS-RM header of SOAP message that marks sequence numbers or ACKs.

3.4.2. Conversion process. Comparing to the individual message conversion approach where the each converted into another self-contained binary format message, the message stream approach requires an internal *Data Structure*

Object that holds the representation of messages in the session. The architecture builds a data structure object by processing a captured XML Schema from the negotiation message. Figure 3 shows abstract process of message conversion process.

In a session that uses a binary representation as an optimized communication data format, stream reader and writer in the architecture read/write in-memory format data. A reader does a sequence of typed reading of each SOAP Infoset instances from the network stream according to the internal data structure, while a writer does a sequence of typed writing.

Originally, Data Format Description Language (DFDL) was included in the architecture design to define a binary format and its library is planned to be utilized to read/write binary format data. However, the schedule of its library implementation is not matched ours. Instead of it, we define a simple schema specification for a prototype implementation, which is modified a little from W3C's recommendation. One of examples of modification we've made is adding an array element data type, such as `<element name= "MyArray" type="array" primitives="float" value="90">`.

3.5. Context Store

One of essential components of the architecture that enables a stream of message approach is a context-store. In previous sections, we define static/unchanged information in conventional SOAP messages – such as information in headers. The context-store makes it possible to look-up the context information from a negotiation SOAP message – unchanged SOAP headers, separated representation, and characteristics of the stream. The HHFR scheme itself is also kept in the context-store as well.

In the HHFR architecture design, we define information in context-store with Universal Resource Identifier (URI). We notate the HHFR scheme itself as 'URI-S'. The current representation of the stream is 'URI-R' and the choice of transport protocol is 'URI-T'.

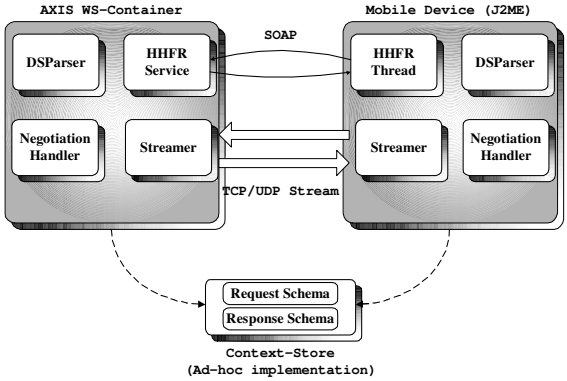

Figure. 4. Simple Overview of Prototype Implementation

4. Implementation

To demonstrate the effectiveness of the HHFR architecture, we have implemented a prototype Mobile Web Service framework based on the HHFR architecture. By implementing the prototype, the HHFR design becomes concrete. It provides separation of message contents and representation, fast communication transport options, and negotiation scheme. In this section, we detail the implementation of prototype.

4.1. Overview

The HHFR architecture design prototype, Handheld Flexible Representation (HHFR) is written in pure Java-based system. It implements major design points of the architecture to provide optimized communication for mobile web services without compensating interoperability. The prototype architecture overview is depicted in Figure 4. Since the architecture design doesn't include WS container functions, it needs to use an existing WS Container, such as AXIS of Apache Software Foundation (ASF).

4.2. Requirements of mobile Programming Environment

We present programming requirements, especially on mobile side, here.

4.2.1. Limited Programming Library. Because of limitations that mobile devices impose, its programming environment is not as much prosperous as typical wired computing

environments. J2ME has limited package supports than J2SE because of code size issues and limited instructions supported by a processor. Also, we have limited choice of XML and SOAP library. kSOAP and kXML [21] support rich APIs with tiny memory footprints; however they provide far less functionalities than conventional Web Service container, such as AXIS 1.x and 2, and .NET.

4.2.2. Wireless Network Connection. The connections of mobile devices have a narrow bandwidth and high latency, comparing to the conventional wired connection. European and Asian countries had launched the third generation cellular service (3G), such as UMTS [22] and W-CDMA [23], which is expected to have a connection speed of 300~500kbps for downloading and 56~90kbps uploading speed. The improvement of the connection speed is big, if we compare it to 2.5G – GPRS [24] or 2.75G – EDGE [25] services of up to 56kbps connection. Yet, the 3G installation in USA is the initial stage. It is serviced only in major Metropolitan areas and we need to wait few more years to use the same service that Europe and Asia use.

4.3. Implementation Details

The prototype implementation of mobile devices depends on kSOAP/kXML for SOAP/XML parsing and SOAP request/response call. And a mobile-side implementation is a service user only, while a service provider is implemented on conventional desktop machine using AXIS container.

4.3.1 Negotiation Scheme. As designed, the role of negotiation stage is essential to conclude the characteristics of following stream. The negotiation stage is an implemented method, *negotiation()* in HHFR service class *HHFRHandler*. Since the negotiation stage is implemented over SOAP protocol, the negotiation is acting different as a caller and a callee on each side. That is, the *negotiation()* method on a initiating side – a caller is a simple SOAP request message creation with desired characteristics of a following stream and it waits a return of kSOAP's *HttpTransport.call()*. And a receiver side-negotiation is implemented as a receptor of the SOAP request message and loads a SOAP response back to the initiator with negotiated stream characteristics information. The initiator – SOAP Request sender gets Boolean value as a return of *negotiation()*. The true value leads the streaming fashion communication. Initiator keeps using the conventional SOAP communication method, if the return value is false.

4.3.2. Fast Communication Transport Option. As discussed earlier, a fast communication transport option in the architecture design provides an alternative communication method in an asynchronous optimized fashion, other than the default HTTP. The TCP and UDP transports are provided as a transport client and a transport server on sender and receiver side, respectively. TCP transport classes provide a message streaming through a connection-oriented socket connection. For the service provider, *StreamConnectionFactory* class waits for the incoming socket connection on a server socket and creates a *StreamConnector* that holds all streaming related classes, such as data stream reader and writer, and streamer. UDP transport implementation has a similar class structure, except it doesn't have a `ConnectionFactory` design pattern because of its connectionless character. It just opens up the UDP port and receives datagram packets.

4.3.3. Queue on Sending/Receiving Thread. Other details on communication implementations are to place send method and receive method in different threads and introduction of queue on sender. The idea is used by communication implementations in many projects nowadays, because of its benefits with little complexity. To put *send()* and *write()* methods to a socket steam in a single thread makes one operation block the other and causes performance degradation. Even though Java 2 Standard Edition provides Non-Blocking I/O interfaces as a standard from version 1.4, mobile computing doesn't get benefits: the functionality is not provided for J2ME yet. Again, the high-latency of current

cellular connections makes it essential to separate *sender* thread with *writer* thread to be able to operate them concurrently.

Queue in the write thread adds more asynchronicity to the communication implementation. It decouples the message packet process performance from the performance of write communication thread. Write thread receives message packets and puts them to the queue. It de-queues a next message packet and tries to write to the socket whenever it is available. The idea works well specially for narrow bandwidth mobile connection where one big message packet can clog the whole transmission.

4.3.4. Streamer and Data Structure. The unit of message in the prototype implementation is a various length byte array, which we call a '*message packet*' as well. The separation of representation and message contents make the architecture have a scheme to extract/build information in a non-self contained way.

The *streamer* is an information extractor and builder with a sequence of switch statement for reading and writing byte format information which contains SOAP message contents – *message packet* in the order of internal *DataStructure* object, which is produced by a *negotiationHandler*. It parses through the received data representation (a schema of the other end point) with DSParser and returns the DataStructure.

5. Evaluation of Prototype Implementation

This section summaries performance benchmark results of the prototype implementation of HHFR architecture design. We develop two applications for the benchmark: a string concatenation service for benchmarking the pure text data and a float number addition service for text conversion required data domain. For comparisons, we develop conventional SOAP applications using AXIS and kSOAP toolkit for each test.

A simple test session is summarized as follows:

1. A service user (client) prepares a message with a given array size.
2. Send it to a service provider (in AXIS)
3. Web Service (the service provider) processes the message and returns a result message to the service user.
4. Repeat step 1 – 3 for the number of messages

5.1. Benchmark Configuration

We configure a benchmark environment as follows: for service providers, AXIS Web Service container runs on Sony VAIO notebook with mobile Intel Pentium 4M CPU 2GHz and 768MB DDR RAM, where Windows XP professional with Service Pack 2 operates. And mobile applications (service users) run on Treo 600 with ARM processor and 32MB RAM, where Palm OS 5.3 operates on as well as MIDP 2.0 and CDLC 1.1.

The time stamps are measured on mobile side (a session initiator) using `System.currentTimeMillis()` of MIDP 2.0 - CLDC 1.1 that returns 10 milliseconds precision time stamps.

Both applications use TCP transport as a choice of fast transport for the benchmark.

5.2. String Array Concatenation

The first benchmark application is a *string array concatenation service* that produces a single concatenated string of all string in a message. We measure a Round Trip Time (RTT) of a session: if there is a five message in the session, we measure entire five concatenation processing time as a RTT. It includes a communication set-up latency, transmission overheads, and concatenation operation time. Additionally, the benchmark of the HHFR prototype implementation contains negotiation overheads.

The benchmark focuses on the performance effect on runtime system by changing a number of messages in a session and a size of array in a message, and comparing them with SOAP's.

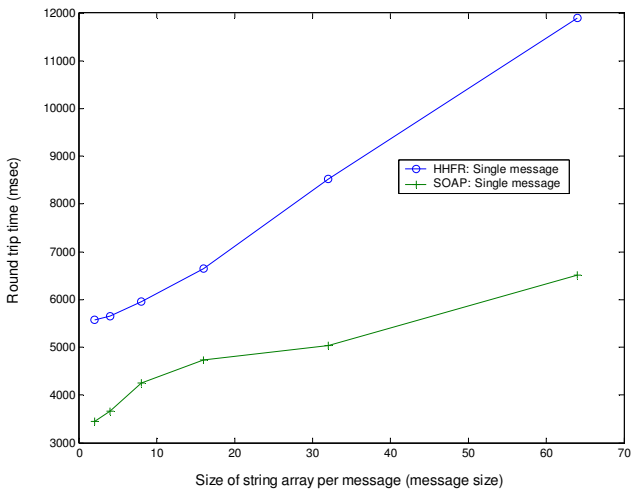

Figure. 5. String concatenation round trip time measurements of a single message in a session with various array sizes (message size). With a single message, a conventional SOAP out-performs HHFR – prototype implementation.

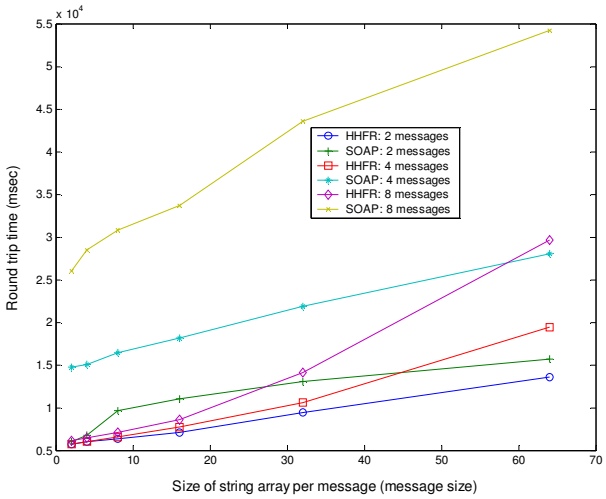

Figure. 6. String concatenation round trip time measurements with various array sizes (message size). This shows different number of message in a session graphs in one. With multiple messages in a session, HHFR needs less time to perform in every case.

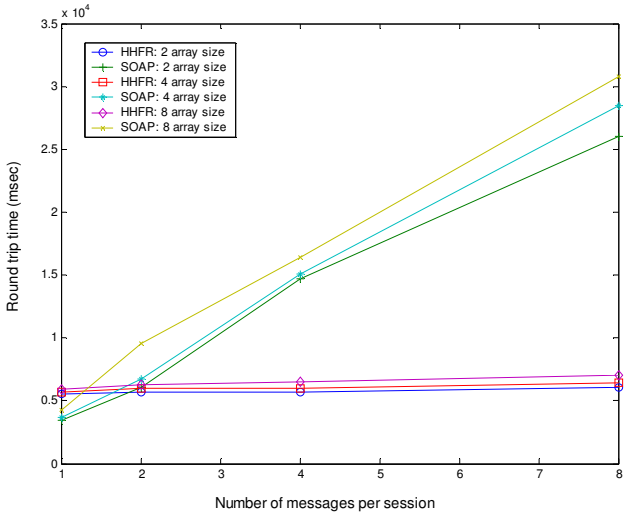

Figure. 7. String concatenation round trip time measurements with various numbers of messages. This shows different message size graphs in one. HHFR out-performs, except a session that consists of a single message.

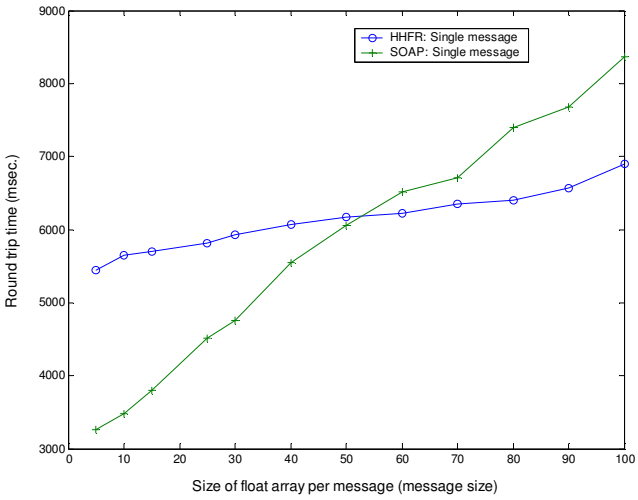

Figure. 8. Float number addition round trip time measurements of a single message with various array sizes (message size). A break-even-point is located between 50~60.

5.3. Floating Number Array Addition

The second application we benchmark is a float number addition service that returns a summation of all float numbers of an array in a message. The benchmark scenario is similar to the string concatenation service. Though, RTT of the SOAP application contains an OS level float-to-text conversion overhead, while HHFR doesn't. Like string concatenation service benchmark, we change a size of array and a number of messages in a session to observe a performance state change in the system, while comparing it with SOAP's.

Figure 8 shows a linear increase of RTT on both, HHFR and SOAP. Assume the formulas for HHFR and SOAP curve as `y = p1*x¹ + p2` and `y = p3*x¹ + p4` respectively. We see `p1 < p3` and `p2 > p4` as well as a break-even point is located between 50 and 60 of array size (`x`) with given input parameters.

5.4. Observations

From the figures, we observe a bigger performance advantages from a sequence of messages in a single session. Figure 7 of the string concatenation benchmark and figure 9 of the float addition benchmark show that HHFR streaming communication always out-performs a conventional SOAP and the gap is fast-increasing as the number of messages in a session grows. These performance gaps are mainly caused by a high network latency of cellular networks. Different from a default HTTP, TCP enable the system avoid network setup overheads during the session. As discussed in section 3, HTTP 1.1 persistence connection or HTTP 1.0 keep-alive option is out-of-considering of our benchmark, since its uncertain availability in a cellular network: proxy support for such a function is optional to the cellular service provider.

The second observation we did is an efficient memory space usage of the prototype implementation by avoiding a text conversion for building text-based SOAP messages in the float number adding service. During the benchmark, the runtime system of the prototype processes larger size array in a message. As equipped with a smaller memory space, OS on a mobile device needs the additional memory space for conversion.

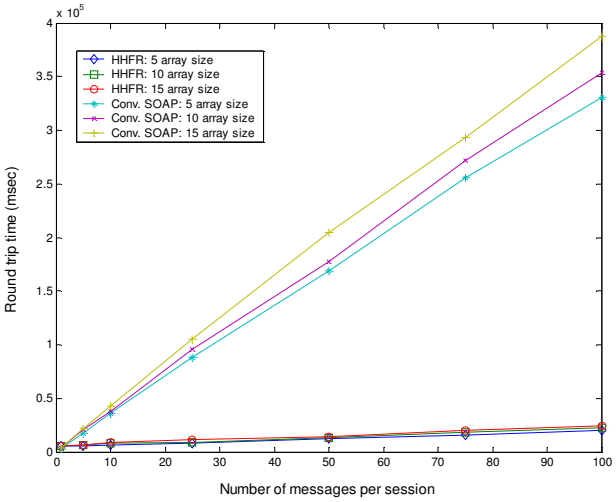

Figure. 9. Float number addition round trip time measurements with number of messages. This shows different message size graphs in one.

6. Conclusion

In this paper, we present new mobile web service architecture, HHFR as well as detailing the prototype implementation. Also, we evaluate the performance of the prototype while comparing conventional SOAP applications by measuring round trip times of given service sessions: the string concatenation service and the float number addition service. The evaluation result shows that our run time system out-performs a conventional SOAP system when the number of message in a session is multiple or the given message size is large. Our architecture helps session oriented web service applications that communicate in a stream of messages, such as multimedia or collaboration application.

7. References

[1] World Wide Web Consortium, "Simple Object Access Protocol (SOAP) 1.1", 2003, http://www.w3.org/TR/soap/

[2] J. Kobielus, "Wrestling XML Down To Size: Reducing The Burden On Networks And Servers",

Business Communications Review, Dec. 2003, pp. 35-38.

[3] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the Limits of SOAP Performance for Scientific Computing", **Proc. of 11th IEEE Int. Symposium on High Performance Distributed Computing HPDC-11 2002**, July 2002, pp. 256.

[4] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, D. Gannon, "Requirements for and Evaluation of RMI Protocols for Scientific Computing", **Proc. of SC2000**, 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] 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/

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

[8] 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

[9] D. Brutzman, and A. D. Hudson, "Cross-Format Schema Protocol (XFSP)", Sep. 2003

[10] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M. Hadley, and E Pelegri-Llopart, "Fast Infoset", Jun. 2004.

[11] International Telecommunication Union, "Abstract Syntax Notation One (ASN 1)", http://asn1.elibel.tm

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

[13] World Wide Web Consortium, "SOAP Message Transmission Optimization Mechanism (MTOM)", Nov. 2004, http://www.w3.org/TR/2005/REC-soap12-mtom-20050125/

[14] World Wide Web Consortium, "XML-binary Optimized Packaging (XOP)", Aug. 2004, http://www.w3.org/TR/2005/REC-xop10-20050125/

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

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

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

[18] A. Slominski, "XML Pull Parser (XPP)", **Extreame! Computing Lab,** http://www.extreme.indiana.edu/xgws/xsoap/xpp/

[19] World Wide Web Consortium, "XML Schema Definition (XSD)", Oct. 2004, http://www.w3.org/XML/Schema

[20] IBM, BEA, Microsoft, and TIBCO Software, "Web Services Reliable Messaging Protocol", Mar. 2004, http://www-128.ibm.com/developerworks/webservices/library/ws-rmdev/

[21] kSOAP and kXML, http://www.ksoap.org/

[22] Universal Mobile Telecommunications System (UMTS), http://www.iec.org/online/tutorials/umts/

[23] Wideband Code Division Multiple Access (W–CDMA)

[24] General Packet Radio Service (GPRS)

[25] Enhanced Data Rates for GSM Evolution (EDGE), http://www.ericsson.com/products/white_papers_pdf/edge_wp_technical.pdf