# Mobile Web Service Architecture Using Context-store

**Sangyoon Oh[1], Mehmet Aktas[2] and Geoffrey C. Fox[3]**
[1] WISE Lab, Div. of Information and Computer Engineering, Ajou University, Suwon, Korea
[e-mail: syoh@ajou.ac.kr]
[2] Information Technologies Institute, TUBITAK-Marmara Research Center, Turkey
[e-mail: mehmet.aktas@bte.mam.gov.tr]
[3] Community Grids Lab, Indiana University, Bloomington, Indiana, 47404, USA
[e-mail: gcf@indiana.edu]
*Corresponding author: Sangyoon Oh

*Received June 30, 2010; revised July 27, 2010; accepted August 10, 2010; published October 30, 2010*

## Abstract

Web Services allow a user to integrate applications from different platforms and languages. Since mobile applications often run on heterogeneous platforms and conditions, Web Service becomes a popular solution for integrating with server applications. However, because of its verbosity, XML based SOAP messaging gives the possible overhead to the less powerful mobile devices. Based on the mobile client's behavior that it usually exchanges messages with Web Service continuously in a session, we design the Handheld Flexible Representation architecture. Our proposed architecture consists of three main components: optimizing message representation by using a data format language (Simple_DFDL), streaming communication channel to reduce latency and the Context-store to store context information of a session as well as redundant parts of the messages. In this paper, we focus on the Context-store and describe the architecture with the Context-store for improving the performance of mobile Web Service messaging. We verify our approach by conducting various evaluations and investigate the performance and scalability of the proposed architecture. The empirical results show that we save 40% of transit time between a client and a service by reducing the message size. In contrast to solutions for a single problem such as the compression or binarization, our architecture addresses the problem at a system level. Thus, by using the Context-store, we expect reliable recovery from the fault condition and enhancing interoperability as well as improving the messaging performance.

***Keywords:*** Mobile web service, XML, web service repository, handheld flexible representation (HHFR), web service framework

This research was jointly supported by Ajou university research fellowship of 2009 (S-2009-G0001-00054), the MKE under the ITRC support program supervised by the NIPA (NIPA-2009-C1090-0902-0003), the Korean government.

**DOI: 10.3837/tiis.2010.10.008**

## 1. Introduction

In an Internet era, it is popular to inter-relate distributed functionalities and resources to build an application. Since we can reuse existing functionalities, service reuse paradigm like Service Oriented Architecture (SOA) saves us the time to develop as well as increases the chance for Business to Business (B2B). Web Services have emerged as a de-facto standard for Service Oriented Architecture in recent years [1][2]. Web Services also profoundly affect the overall distributed computing area. Like their predecessors, such as CORBA, RMI and DCOM, the primary goal of the Web Services is to inter-relate distributed functionalities. But, unlike its predecessors, it achieves its goal in an elegant and neutral manner; it provides well-defined interfaces for distributed functionalities, which are independent of the hardware platform, operating system and programming language. So, distributed functionalities, or services, that are run on different hardware platforms, run on different operating systems, or written in different programming languages, can communicate through Web Service interfaces. Web Service may be the best candidate for machine-to-machine (process-to-process) interaction technology because of its strong interoperability.

While the Web Service technology has become a standard to connect remote and heterogeneous resources, mobile devices have become a vital part of people's everyday life. People use mobile devices anytime and anywhere, such as cellular phones, smart phones and handheld game consoles. The Web Service technology recognizes mobile computing as an area that it should expand into [3]. Through integration, Web Services enable pervasive accessibility by acquiring mobility as they overcome the physical location constraint of conventional computing. Meanwhile, mobile computing also requires a technology that connects mobile systems to a conventional distributed computing environment. Since mobile applications runs on different platforms, we need the integration technology which is strong in interoperability. Web Services may be the perfect candidate for such connection, since a strong interoperability is the key requirement of the technology. This will be important for the success of Web Services when we consider the fact that the mobile computing environment is much heterogeneous in terms of hardware platforms, operating systems and programming languages. Thus, the integration of mobile computing with Web Services technology will yield many advantages for both sides [3][4].

However, despite the fact that the condition of mobile computing has much improved in recent years, there are fundamental differences between mobile and PC-like stationary environments such as limited processing power and the battery-life problem on the wireless side. Thus, applying the current Web Services communication models to mobile computing causes potential performance overhead that mostly come from XML's verbosity. The interoperability of Web Services mainly comes from their Extensible Markup Language (XML) based open specific standards. However, Web Services self-descriptive characteristic causes two problems in mobile computing environments. First, the encoding and decoding of verbose XML-based SOAP messages consumes resources. Therefore, Web Service participants, particularly mobile clients, may suffer from poor performance. Also, a large portion of a message contains static information that is the same for known participants. This causes an increase in the message size and consumes unnecessary processing time for redundant information.

Since the conventional Web Service communication framework does not adequately meet the needs of mobile computing as noted above, we need an optimized architecture (i.e.,

reducing message size to save wireless bandwidth, reducing parsing overhead and reducing communication latency caused by improper communication scheme) to prevent performance degradation in mobile computing as well as in conventional computing that is interacting with mobile applications. The optimized architecture should provide the following capabilities: 1) an optimized information representation to minimize the size of messages, 2) a streaming style message exchange that is clearly different from the request-response style of the current HTTP and 3) an online Web Service Repository where participants are able to store static or redundant parts of the message.

There have been many studies on addressing possible performance degradation. Such proposals widely range from technical approaches like binarization of XML messages [5][6] to user modeling approaches to maximize user experience (i.e., better application response time and application startup time) [4]. However, those studies are mostly providing a solution to a single specific problem rather than providing a system-level comprehensive architecture.

We designed and implemented a novel architecture called the Handheld Flexible Representation (or HHFR) for optimized Web Service messaging in mobile computing. The key design goal of HHFR architecture is to optimize Web Service messaging in mobile computing. To achieve the goal in a system-level, we adopt three design characteristics. Firstly, we separate the message content from its message syntax and let them have flexible representation based on their communication environment. When we serialize message content into the XML document, we need additional time for message processing to parse and more bandwidth for increased message size. Secondly, we adopt a streaming style communication scheme instead of using conventional request-reply based HTTP scheme. Thus, mobile Web Service clients exchange messages in streaming fashion. Finally, we introduce a negotiation stage at the beginning of Web Service interaction between Web Service and clients to set up the streaming channel and agree on the message representation. As empirical results shows in Ref [7], the HHFR architecture outperforms conventional Web Service architecture when the client application is running on less powerful mobile device, and the client and the Web Service exchange information in a session (i.e. exchanging similar messages in series ).

However, one essential capability of the HHFR architecture we listed above is addressed in an ad-hoc way. A Web Service repository can be used for storing static parts of the messages in a session to save bandwidth and parsing time. The static parts can be retrieved anytime it is needed. To address the Web Service repository issue, we improve HHFR to optimize messages better in interoperable fashion. For the extended HHFR architecture, we design and implement the Context-store [1]in UDDI and WS-Context compliance [8].

The two contributions of the paper are as follows: the main contribution is to design an interoperable information repository for mobile Web Service based on UDDI specification. Adopting WS-Context and UDDI compliant Context-store will make the system a more interoperable environment. In the current Web Service environments, there are many Web Services and clients that are capable of understanding WS-Context or UDDI specification. Especially for mobile clients, HHFR architecture provides not only message optimization capabilities but also a context repository (i.e. an online Web Service database). The other contribution is modeling the use of Context-store as well as conducting the comprehensive experiments. The purpose of the experiments is to show that 1) accessing time to the Context-store is nominal, 2) the use of Context-store reduces bandwidth and 3) our

[1] We will use both Web Service Repository and Context-store interchangeably throughout the paper. To be precise, a Context-store is a HHFR specific name for Web Service Repository

Context-store can serve enough number of requests from the service and clients. We expect that the extension makes the HHFR architecture interoperable with other UDDI and WS-Context compatible applications.

This paper is organized as follows. In Section 2, we discuss related works that address the issues of mobile Web Services. We present an overview of our HHFR architecture design in Section 3 and illustrate the extended version of HHFR with Web Service Repository in detail in Section 4. In Section 5, we show our empirical results and we discuss and conclude in Section 6.

## 2. Related Works

The research issues in Mobile Web Service mostly fall into a performance overhead category. It is mainly caused by the verbosity of XML (i.e. XML is too heavy for mobile devices in most cases). To improve the performance of mobile Web Services, we have to design the Web Service system from two different perspectives: architectural and optimization view. We can use one of three architecture views to implement a mobile Web Service: a wireless portal network, the wireless extended Internet, or a peer-to-peer (P2P) network as Adacal and Bener described in Ref [9]. A wireless portal architecture uses a gateway between a client and a service. The wireless extended Internet architecture allows mobile clients directly communicate with services like conventional stationary PC clients. In P2P network, mobile clients can be a Web Service provider. In this paper, from the architectural perspective, we focus on the wireless extended architecture where a client can directly talk to the service. The optimization issue is the main focus of interest of this paper. We will delve into the issue throughout the paper.

In this section, we overview related works in two folds: first, we visit approaches which try to minimize communication and processing overheads caused by the verbosity of XML message. Then we describe UDDI and WS-Context shortly before we discuss the relation between optimizing mobile Web Service messaging and UDDI.

### 2.1 XML Message Optimizations

From the optimization perspective, approaches to improve the performance of mobile Web Services can be categorized into either naïve compression of messages or binarization of messages. The compressing message approaches utilize various compressing mechanisms to provide smaller size messages to reduce the bandwidth usage of constraint wireless communication channel. Tian et al. studied mobile Web Services environment and pointed performance concerns about XML messaging efficiency [10]. The experiment shows their dynamic compression algorithm performs well and can save bandwidth. According to ref [11], XML specific compression such as XMill and Millau [12] may perform much better on small messages. Developed by Dennis Sosnoski, XBIS [13] also uses a generic scheme for replacing repetitive words (a define-and-replace scheme). XBIS is similar to XMill in terms of how it replaces repetitive words with an index, but there is a difference between the two. XMill processes an entire document at once, whereas XBIS processing can encode a streaming input, so the transformation allows encoding and decoding to start on a partial document. XBIS forms all the components of an XML document in the same order they appear in the text. Like other repetitive words replacement schemes, XBIS defines each name as text only once, and it then uses a handle value to refer back to the name when it is repeated. However, these naïve approaches do not address the fundamental problems of mobile Web Services' environments

as well as adding one more layer of processing (compression-decompression) to less-capable mobile CPUs.

Another stream of studies on improving mobile Web Services' environments focuses on utilizing the binarization mechanism of XML. Whether or not it is self-contained (maintaining self-descriptive characteristics of XML), there has been a lot of studies and proposals [5][7][14][15]. First, Paul Sandoz and his team at Sun Microsystem proposed Fast Infoset specification to W3C workshop on Binary Interchange of XML information Item as an XML alternative to provide faster and more efficient Web Services in restricted computing environments. Serialized (i.e., binarized) XML document contains information items and their properties as well as the hierarchical structure of the XML Document [5]. The Cross Format Schema Protocol (XFSP) [16] is another project that serializes XML documents based on a schema. Initially it was created to provide a flexible definition of network protocols. It is written in Java and uses the DOM4J model to parse the schema. Combined with XML Schema-based Compression (XSBC) [17], XFSP provides binary serialization and a parsing framework. ExtremeLab of Indiana University presents studies on binary XML for scientific application (BXSA) [15] that has a new structure and layered format based on XBS [18].

### 2.2 UDDI and WS-Context

The UDDI Specification of OASIS [19] helps us to solve the problem of locating services of interest. It is an XML based online registry to list services on the Internet. It provides set of specifications for service description and discovery. UDDI also provides components to register catalog data (i.e. Yellow Pages) and technical information about the service (i.e. Green Pages) along with business registry information (i.e. White Pages: address and identifiers).

However, UDDI can be used as an online information service of Web Services. If it is WS-I [20] compliant and used for storing context of a Web application, Web Services and clients can share dynamic state information of the application session as well as static service information. Levenshteyn and Fikouras [21] use the WS-Conext compliant UDDI service to correlate the work of participants within the same activity by disseminating additional information (i.e. context). The context contains a unique identifier that allows a series of operations to share a common outcome. Yet, no one tries to use UDDI and WS-Context compliant information service for improving mobile Web Service communication performance.

### 2.3 Discussions

There have not been many studies that address mobile Web Service's performance issue from the system level. Rather, they focus on a single problem (e.g. utilizing binarization mechanisms) at a time. Especially, the study on integrating a Web Service repository to improve mobile Web Service's performance has not yet been done. In this paper, we propose the extension of HHFR architecture and we present a detailed study about utilizing an online Web Service repository in a mobile Web Service environment.

## 3. Mobile Web Service Architecture: HandHeld Flexible Representation

In this section, we present a software architecture designed to optimize communication in mobile Web Services – the Handheld Flexible Representation (HHFR), which distinguishes the semantics of messages from their representation. In the beginning of an HHFR message stream, two participating nodes negotiate the characteristics of the stream. Once this

negotiation is complete and the stream is established, the two nodes exchange the message content, which is a combination of semantics and representation, in an optimized fashion. An abstract diagram of the HHFR architecture design appears in **Fig. 1**. The early version of the HHFR implementation is presented in [7].

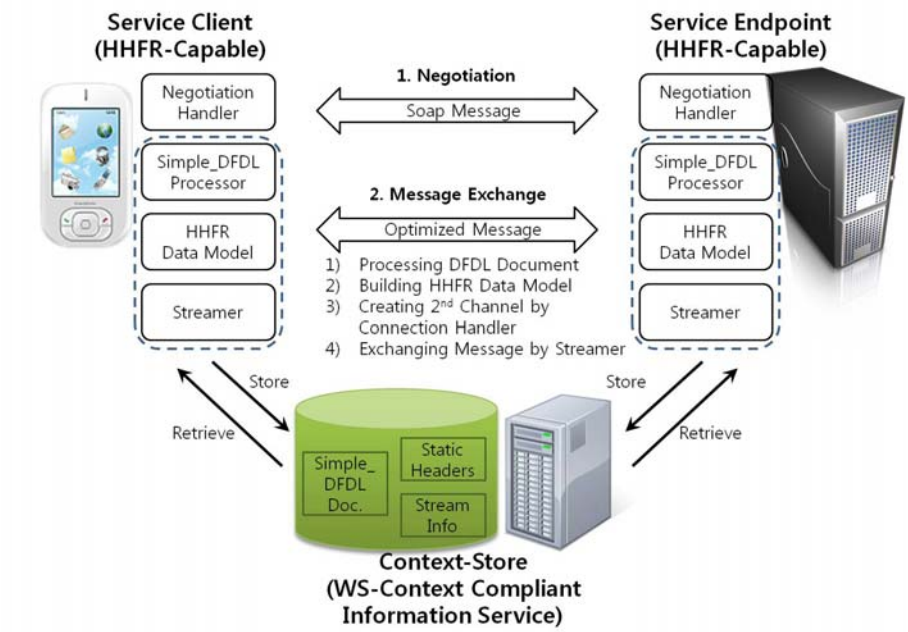



**Fig. 1**. Overview Diagram of HHFR

### 3.1 Motivation and Overview of Design

In many cases, mobile applications interact with a Web Service in a session style. That is, they exchange messages continuously and messages are similar or the same. For example, a mobile client in a ubiquitous software system may send context-aware information such as temperature, intensity of illumination and GPS position numbers. In this case, the mobile client and the Web Service exchange messages with the same message format and the only context values are changing. A mobile client in a Video/Audio conference using Unified Communication environment [22] receives and sends voice and image data in a streaming manner with the same message style. We define the term *a session* for consecutive messages between a service and a client.

When a mobile client and a service exchange messages in a session, most of the messages are in the same representation (i.e., structure and format), except the starting message and ending message. Messages in the middle have the same structure, but values are changing as new information is produced. Also, messages in a session may include some information represented during the session.

For the application domain, which uses session style, we propose a novel mobile Web Service architecture HandHeld Flexible Representation that utilizes the characteristic of this repeating structure and information. In a mobile computing environment where mobile clients and services co-exist, the usage scenario of HHFR is as follows: Web Service participant initiates a stream, which is a series of message exchange using the same structure and type, by sending a SOAP request message to negotiate the characteristics of the following

communicated messages with another participant. If the negotiation is successful, which means that the other participant agrees to use the HHFR scheme, and then the two participants (i.e., endpoints) exchange messages in a preferred representation. The preferred representation is the negotiated format of messages, and it is not limited to SOAP-style, but rather supports many optimized formats. The message's semantic content is preserved while the syntax used to express the content is agreed upon in the negotiation stage, and HHFR uses this negotiation to establish a message stream. **Fig. 2** illustrates the usage scenario.

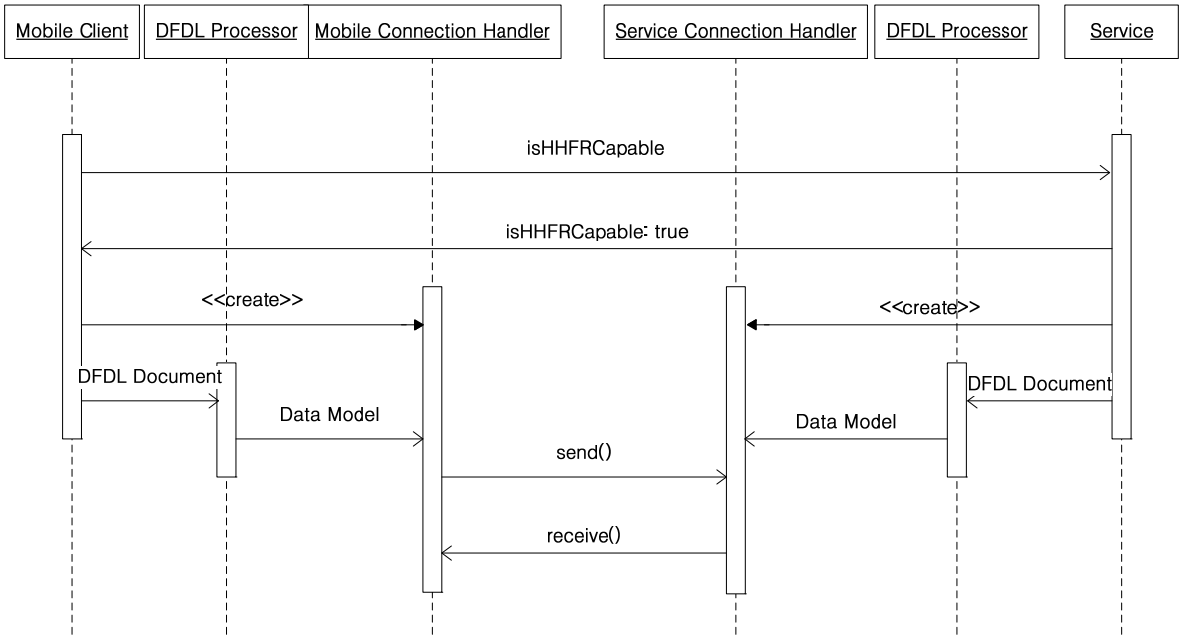



**Fig. 2**. Usage Scenario of Service Interaction using HHFR

There are three key design points of the HHFR architecture, which make the message exchanges in HHFR efficient. First, HHFR uses a Data Format Description Language (DFDL) [23]-style data description language, named the Simple_DFDL, to represent the message structure and type. HHFR distinguishes between message semantics and syntax, and the syntax is represented in the Simple_DFDL. Simple_DFDL will be briefly discussed in the following section, and detailed information about Simple_DFDL is in [7].

Second, in the HHFR, applications exchange messages in a streaming style. HHFR sets up a message stream (i.e. second channel) between the participants based on the characteristics negotiated. The message exchange is then freed from "waiting for response" by adapting an asynchronous messaging style.

Third, in HHFR, an online Web Service Repository module holds the static (within a particular stream) data of the messages: These include a) the unchanging or redundant SOAP message parts, b) the Simple_DFDL file as a data representation, and c) the negotiated characteristics of a stream. By storing the message fragment or meta-data of the stream as context, the application can exchange stripped down messages that contain only the vital part of the message content without losing the formal ability to produce the conventional SOAP message representation on demand.

### 3.2 Replacing XML Syntax with Optimized Representation: Simple_DFDL

By separating message semantics from syntax, the proposed architecture provides mobile applications options to choose the appropriate message representations (i.e., a binary or a conventional SOAP representation) for a given Web Service communication environment. The binary representation is a critical option to improve the overall performance of the HHFR architecture for several reasons. First, it reduces the size of an exchanged message by removing the verbose SOAP syntax. The message size can be reduced by up to a factor of 10 if a document structure is especially redundant (e.g., with an array) [24]. A binary message representation also helps the HHFR architecture to avoid textual conversion. The architecture simplifies the conventional encoding/decoding stage[2], in which the in-memory representation is converted into a text format and vice versa. This is an expensive process, especially for the relatively low-powered mobile devices that are required by SOAP syntax. Among data conversions, floating point number conversion is the most costly one [25].

Finally, another benefit to having a binary representation of the SOAP message is that it does not need to be parsed in a conventional way. Since SOAP syntax requires a structured representation, we need to parse a given document to get information. A SOAP message in binary representation (i.e., in a byte array format of contents) exists as chunks of continuous XML information items that don't need to be parsed in a conventional way. Rather, the architecture offers another information retrieval scheme: Stream reader and Stream writer. The latter enable the applications to read and write information item data to and from byte stream by using Simple_DFDL to distinguish message semantics, i.e., information, message syntax and the message generated from the given Simple_DFDL document.

Simple_DFDL that is a simple restricted XML Schema Definition (XSD) is our method of defining the XML syntax of a message. While we design Simple_DFDL, we constrain the XML Schema definition to achieve a single structure by parsing the XML Schema document itself (i.e., a Simple_DFDL document should be a single XML Schema document rather than multiple documents.). Therefore, the HHFR architecture can use a Simple_DFDL document as a representation of both structures and types. Other constraints are as follows:

- There can be no reference in the Simple_DFDL definition using fragment identifiers or an XPointer.
- The Simple_DFDL supports only limited Built-in simple types, such as string, float, double, integer, boolean and byte.
- The Simple_DFDL does not support facets like minInclusive and maxInclusive to restrict the valid values.

Since we preserve the message semantics in the SOAP Infoset data model, HHFR is also able to handle various representations other than binary. We are able to send and receive messages in binary format as well as in the traditional SOAP syntax. Here is an example of Simple_DFDL to illustrate usage. An example array declaration follows:

```
<xs:element name="HHFR">
    <xs:complexType>
        <xs:element name="arraySize" type="i" value="10"/>
        <xs:element name="array" type="f"/>
    </xs:complexType>
</xs:element>
```

[2] They are called marshalling/unmarshalling in some projects,

Because individual messages in the HHFR architecture are not self-contained, the architecture builds an internal data structure that contains the names of element information items, attributes and child properties by parsing a Simple_DFDL document. Also the parsed structure of the Simple_DFDL document represents a serialized structure of the SOAP body. In combination, the internal Data Structure object and the HHFR message packet, which has an optimized representation, can be transformed back to the original from the SOAP message.

In our architecture design, each message representation is optimized according to the characteristics that are negotiated during the negotiation stage and the principles that are predefined by an architecture specification. A binary format is the optimized representation in most cases. However, in some conditions, views other than a binary representation can be preferred. In the negotiation stage, a handler responds to the negotiation requestor with message representations supported by the handler. Suppose the handler supports view A and view B, but it prefers B. Despite the fact that the service prefers message format A, the service may process a received format B message if the conversion process overhead is higher than the threshold defined in the HHFR design specification.

In a conventional Web Service environment, XML is the representation format that provides interoperability to the heterogeneous participating nodes. Yet, in a constrained computing environment, processing an the XML format message becomes a performance bottleneck because of its verboseness. The preferred representation concept of the HHFR architecture can provide an optimized representation of the mobile and conventional application and the given network characteristics.

### 3.3 Negotiation of Characteristics

A couple of design issues motivate an introduction of the negotiation stage. First, to have an alternative representation of SOAP messages, the representation of messages should be transmitted at the beginning of the stream. Second, to set up fast and reliable means of communication, the architecture should negotiate the characteristics of the stream.

A stream of messages shares the same representation, meaning these messages share identical structure and type of XML fragments, i.e., SOAP Body parts. The applications on participating nodes negotiate a preferred representation and send messages in the preferred representation according to the exchanged Simple_DFDL representation. Together with the representation, the headers of the SOAP messages remain mostly unchanged in the stream. Thus, these unchanging headers can be archived in the Context-store, and the sender can avoid transmitting them with each message. Needless to say, some headers, like the reliability related headers, are unique to individual messages. Such headers need to be transmitted with each individual message and processed at the corresponding handlers. Unchanging headers, which are often the majority of headers, can be transmitted only once, and the rest of the messages in the stream can use saved-headers from the initial transmission.

The negotiation stage uses a single (or multiple, if necessary) conventional SOAP message[3] that makes the negotiation stage compatible with the existing Web Service framework. The architecture design defines each issue, such as reliability, preferred representation or security, as an individual element item in a Negotiation Schema. The process begins when an application on a participating node initiates a message stream by sending a negotiation request to a service node. The negotiation handler receives a SOAP negotiation message and prepares a response SOAP message containing the negotiated items.

[3] If the negotiation can be continued until the two participants reach a single agreed upon point.

Compared to the conventional Web Service communication method, the negotiation stage is an additional overhead[4] in the HHFR architecture, and this will discourage use of the scheme in a short message stream that has few exchanged messages. However, for larger message streams with many of redundant messages, the HHFR architecture's negotiation overhead is negligible.

## 4. Web Service Repository: Context-store in HHFR

In this section we present our extended version of HHFR with Context-store design and implementation in detail. We describe the Context-store design characteristics. In the following section, we provide our empirical experiment results using an implemented service to validate the design efficiency of the Context-store and its effect on the overall architecture.

### 4.1 Benefits and Design Characteristics of Context-store

As we discussed, a Context-store is an essential component of the architecture where we can store unchanging or redundant SOAP headers (e.g., namespace and encoding style information), a Simple_DFDL document as a message representation and the characteristics of the stream. The Context-store also archives the static context information from a SOAP negotiation message; the HHFR design specification scheme itself is also kept in the Context-store. By archiving, the context-store can serve as a meta-data repository for the participating nodes in the HHFR architecture.

By storing and retrieving redundant and static message parts, mobile clients can be benefit from the Context-store. Other than bandwidth saving, there are two more distinct benefits from the Context-store; namely, supporting a reliable session for Web Service applications and enhancing interoperability among the participating applications. Since a mobile client and a service provider can save the session configuration during their session, the session can be recovered from unexpected fault condition by retrieving stored configuration. They may re-play the session or jump into the last state. As we discussed earlier, adopting WS-Context and UDDI compliant Context-store will make the system as a more interoperable environment. There are tons of Web Service applications and clients which are capable of understanding WS-Context or UDDI specification and using the implementation. For them, HHFR architecture provides not only message optimizing capabilities but also a context repository (i.e. an online Web Service database). The overview of extended HHFR architecture and the benefits of the Context-store are depicted in **Fig. 3**.

The Context-store implementation could be either a local or a remote service. A local Context-store implementation is an internal module that keeps context. When it is a local service in the runtime environment, other components in the HHFR architecture make a method call to save the Context of the stream and to retrieve the context from the repository. It is simple and straightforward, and in this case, an individual node holds the context-store.

In this extension of the architecture, we choose a remote service like Domain Name Server (DNS) and our choice of a Context-store design is WS-Context specification [8]. The context of the stream contains shared information among Web Service participants and the HHFR specification itself. This is where the WS-Context specification is well suited. If the Context-store is implemented as a WS-Context server, then participating nodes can archive

[4] Others are a Repository (i.e., Context-store) accessing overhead and Simple_DFDL designing overhead.

and retrieve contexts of the stream with an identifier, e.g., Uniform Resource Identifier (URI). The HHFR architecture design defines information in the context-store with a URI.

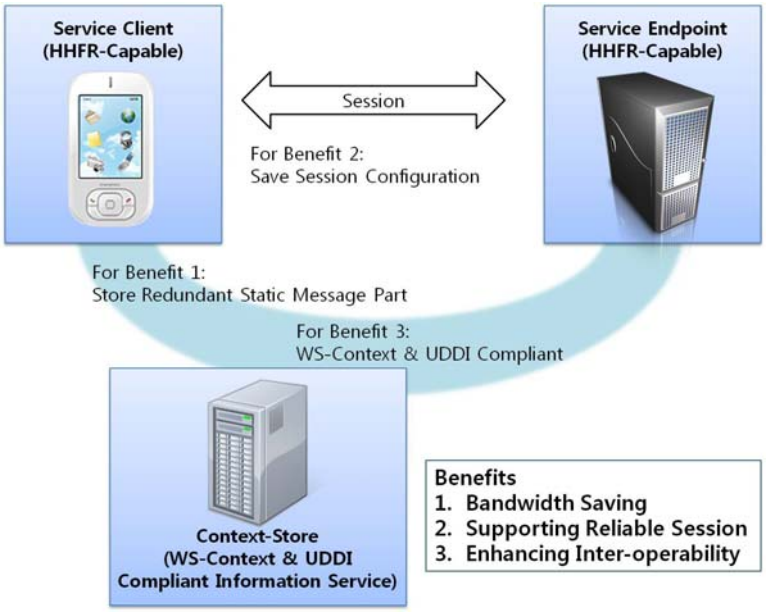



**Fig. 3**. An Overview and Benefits of Extended HHFR

### 4.2 Context-store in Mobile Web Service

We chose Java as a language platform for both mobile and conventional sides because it is portable across platforms, and the JME together with third party products, provides a rich set of libraries. The architecture itself is not limited to any specific language platform and can be applied to message communications between heterogeneous platforms, but we believe a single-language prototype such as Java can show the effectiveness of the architecture design in all respects but a few (e.g., the capability to float data conversion between different operating systems). We validate efficiency and scalability of proposed extension design through the empirical experiments and the results are shown in section 5.

The purpose of our scheme is to provide a way of using the information service from mobile applications to enjoy advantages noted above. On the other hand, integrating a Web Service based the information service (e.g., Context-store) with HHFR brings a dependency on SOAP-Java binding on the Apache Axis library. The Axis version of the JME (Java Micro Edition) environment or any other popular programming environment for small and embedded devices has not yet been developed and will not be in the near future because of a lack of related programming libraries, such as advanced XML parsers and utility libraries. So it is not feasible to use the existing Axis-based client interface (to an Information Service) without porting the code. Unfortunately, replacing JSE APIs with JME APIs is not possible, so we must find an alternative solution.

The solution to the first problem includes a direct serialization of SOAP request message and a parsing SOAP without Axis SOAP-Java binding. The same approach we used for the negotiation message is used here: we use the kSOAP[5] library [26] for this process. SOAP

[5] kSOAP is the product of an open source project, Enhydra, led by Stefan Haustein

serialization using kSOAP library needs an ad-hoc method to integrate with Information Service while SOAP parsing is straightforward. Because Axis SOAP-java binding is not available for the JME environment, we focus on generating SOAP messages by the WS-Context client based on Axis. The Axis-Java binding adds a hierarchically referenced element to the structure if the binding process meets a Java wrapper when it serializes SOAP message. As a result, Axis based SOAP binding code for WS-Context Service client generates multi-referenced XML. Unfortunately, kSOAP does not support such advanced binding APIs; rather, it provides more direct SOAP serialization APIs. For example, a piece of Java code below will result to generate a XML fragment in **Fig. 4-(b)**, which is highlighted as bold face.

```
SoapObject context = new SoapObject(NAME_SPACE, "ContextType");
SoapObject context_data = new SoapObject(NAME_SPACE, "ContextType");
SoapObject contextID = new SoapObject(NAME_SPACE, "string");
context.addProperty("context-identifier",  identifierKey);
context.addProperty("context-data", data);
```

**Fig. 4-(a)** shows the `getContext` SOAP request message of the conventional WS-Context client using Axis, and **Fig. 4-(b)** shows the flattened SOAP request message produced by the mobile WS-Context client using kSOAP.

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Body>
    <ns1:getContents
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://wsctx_service.WSCTX.services.axis.cgl">
        <body href="#id0"/>
        </ns1:getContents>
    <multiRef id="id0" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xsi:type="ns2:GetContents"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"xmlns:ns2="http://wsctx_
    schema.WSCTX.services.axis.cgl">
       <correlation-id xsi:type="xsd:string" xsi:nil="true"/>
       <context href="#id1"/>
       </multiRef>
    <multiRef id="id1" soapenc:root="0"
    soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
    xsi:type="ns3:ContextType"
    xmlns:ns3="http://WSCTX.services.axis.cgl/wsctx_schema"
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
        <context-identifier xsi:type="xsd:string">
        context://hhms/Sangyoon </context-identifier>
        <context-data xsi:type="xsd:string" xsi:nil="true"/>
       </multiRef>
    </soapenv:Body>
</soapenv:Envelope>
```

**Fig. 4. (a)** getContext() SOAP request message created using Axis. Referenced elements are highlighted in boldface.

```
<v:Envelope xmlns:i=http://www.w3.org/1999/XMLSchema-instance
  xmlns:d="http://www.w3.org/1999/XMLSchema"
  xmlns:c="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:v="http://schemas.xmlsoap.org/soap/envelope/">
   <v:Header />
   <v:Body>
      <n0:getContents id="o0" c:root="1"
        xmlns:n0="http://wsctx_service.WSCTX.services.axis.cgl">
          <body i:type="n0:body">
             <context i:type="n0:ContextType">
                <context-identifier i:type="d:string">
                context://hhms/sangyoon</context-identifier>
             </context>
          </body>
      </n0:getContents>
                                        </v:Body>
</v:Envelope>
```

**Fig. 4. (b)** getContext() SOAP request message created using kSOAP for a mobile WS-Context client. Elements resulted from the piece of Java code are highlighted in boldface.

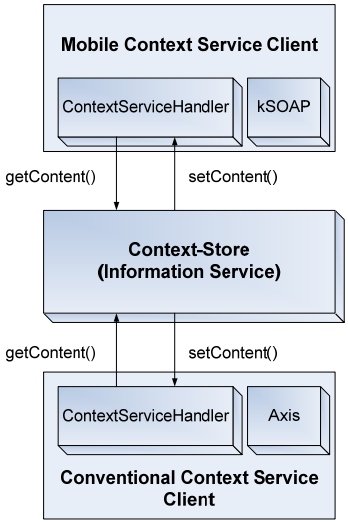



**Fig. 5.** Mobile and conventional context service clients

As depicted in **Fig. 5**, the two primary WS-Context related functionalities of Information Services are `getContent()` and `setContent()` methods, which provide access and store operations that are equivalent to the Axis based component of the conventional client. Method calls are not tied to any other operation in the HHFR session, so they can be called at anytime when the HHFR runtime or the HHFR client service needs to create, update or retrieve context in the Context-store (i.e., Information service). Thus, the following Java program 1) create `ContextServiceHandler` object with the Context Service URI and the service (implementation) version, 2) store given context of any type paired with a unique identifier, and 3) retrieve context. `ContextServiceHandler` object is a wrapper class and provides `getContent()` and `setContent()` methods.

`getContent()` and `setContent()` methods throws `java.lang.InterruptedException`, since the handler runs as a Thread. Running as a Thread can avoid a possible deadlock situation, which could occur when the network fails or due to an operational error.

```
ContextServiceHandler handler = new ContextServiceHandler(SERVICE_URL, 0);
try {
   boolean result = handler.setContext(identifier, givenContext);
   Object contextData = handler.getContext(identifier);
}
catch (java.lang.InterruptedException exception) {
   exception code…
}
```

There are few limitations that could be improved and extended. First, the ad-hoc method to generate a SOAP message is the biggest obstacle to automate client code generation. Compared to automatic Java binding generation of Axis, the method also imposes human error when the multi-referenced SOAP is converting into flattened structure.

## 5. Performance Evaluation

The goal of performance evaluation is to demonstrate the effect of using the Context-store in the HHFR architecture, how much overhead to access the Context-store, and the scalability of the approach. Preliminary evaluation of HHFR architecture focused on performance gains using preferred message representation (i.e., binarization of SOAP message) is presented in detail in Ref [7].

**Table 1**. Summary of evaluation measurements

| | Measurement | Protocol | Comment |
|---|---|---|---|
| 1 | Context-store access time | SOAP | A time to access a Context-store from a mobile client |
| 2 | Round Trip Time to exchange a message | HHFR | Bandwidth gain from using a Context-store |
| 3 | Scalability | SOAP | The scalability of our approach which is analyzed from the processing time of the Context-store service. |

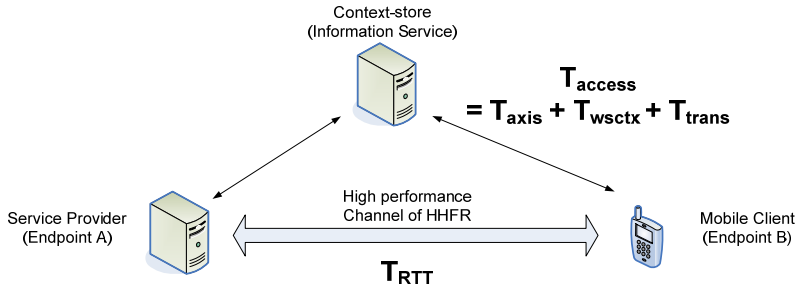



**Fig. 6**. System parameters

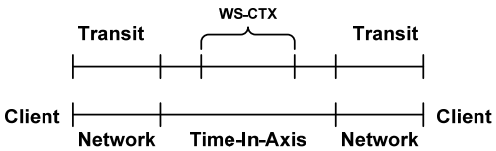



**Fig. 7**. System parameters with time frame

### 5.1 Evaluation Model and Test Environment

For the evaluation, we focus on measuring and analyzing three values: 1) time to finish a Context-store access from a mobile client (i.e., a time between a SOAP request and a SOAP response), 2) bandwidth (time) gain from using a Context-store, and 3) the scalability of our approach to use a Context-store. Thus, we design the evaluation measurements to have three aspects. First, we measure the time to access the Context-store from a mobile client. Second, we measure the Round Trip Times to show the performance effect of using the Context-store to store redundant and/or unchanging parts of the SOAP message. Third, we analyze the scalability of our approach by measuring the time it takes to process a WS-Context SOAP message on the service side. The second measurement is distinguished from the other two because it uses a high performance channel of HHFR to exchange messages, and the first and the third experiments use a conventional SOAP message for measurements. **Table 1** shows a summary of our evaluation measurements.

For the evaluations, we assume the following system parameters.

- $T_{access}$: time to finish accession to a Context-store (i.e., save a context or retrieve a context to/from the Context-store) from a mobile client
- $T_{RTT}$: Round Trip Time to exchange message through a HHFR channel
- N: the maximum number of stream supported by one server
- $T_{wsctx}$: time consumed to process setContext operation
- $T_{axis\text{-}overhead}$: time consumed to process Axis data-binding and HTTP request/response process
- $T_{time\text{-}in\text{-}server}$: time consumed in the Axis server
- $T_{trans}$: time consumed to transmit a message over the network
- $T_{stream}$: the length of a stream in seconds

We measure $T_{access}$ in the first experiment and measure $T_{RTT}$ in the second. In the third experiment, we measure $T_{wsctx}$ and $T_{time\text{-}in\text{-}server}$ and assume $T_{stream}$ to analyze the scalability of our model. **Fig. 6** and **Fig. 7** show parameters on the illustrated system model.

$$T_{access} = T_{wsctx} + T_{axis-overhead} + T_{trans} \tag{1}$$

In our test model, we set that there are three Context-store accesses per session, i.e., two accesses are made from each Web Service participant nodes at the beginning of the session, and one access is made to report the end of the session to the Context-store. Let us consider N simultaneous streams are happening during the time period of $T_{stream}$. Thus, we can formulize the calculation of the scalability of our approach to use the Context-store that is the maximum number of supported simultaneous streams as follows:

$$\frac{3N}{T_{stream}} \approx \frac{1}{T_{time-in-server}} \tag{2}$$

$$N \approx \frac{T_{stream}}{3 \times T_{time-in-server}} \tag{3}$$

It should be noted that there are three major parameters on which our evaluation analysis depends. First, $T_{access}$ is governed by $T_{trans}$ that can vary from the wireless (i.e., cellular) technologies. Second, the $T_{axis\text{-}overhead}$ at the Web Service container is the dominant factor in message processing. In this evaluation, we used Axis 2 version 1.4 with an Axis data binding to measure the message processing overhead (i.e., $T_{time\text{-}in\text{-}server}$). Finally, the stream length (i.e., how long an application usage session lasts) is also an important parameter to analyze scalability. In our analysis, we assume the stream length as ten minutes (i.e., 600 seconds).

Three Context-store accesses per stream spread over the stream length; thus, the longer the stream length, the more simultaneous streams can be supported.

Evaluations conducted over 14.4kbps wireless cellular networks and the mobile applications are running on Smart phones that equipped with a 144MHz ARM processor. **Table 2** shows the summary of the configuration.

**Table 2**. Summary of the Machine Configuration

| | **Service Client** | **Service Provider** |
|---|---|---|
| Processor | Intel Xeon (2.4GHz) | ARM (144MHz) |
| RAM | 2GB | 32MB |
| Bandwidth | 100Mbps | 14.4kbps |
| Java | Java 2 SE | CLDC 1.1 and MIDP 2.0 |
| SOAP Engine | Axis 1.2 | kSOAP 1.1 |

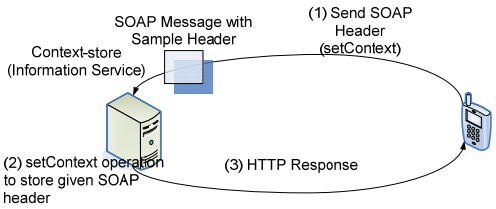



**Fig. 8**. Set up for measuring Context-store accessing overhead

```
<?xml version="1.0" encoding="UTF-8" ?>
<S:Envelope xmlns:S="http://www.w3.org/2001/12/soap-envelope"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:wsu="http://schemas.xmlsoap.org/ws/2002/07/utility"
  xmlns:wsp="http://schemas.xmlsoap.org/ws/2002/12/policy"
  xmlns:wsrm="http://schemas.xmlsoap.org/ws/2003/03/rm"
  xmlns:wsa="http://schemas.xmlsoap.org/ws/2003/03/addressing">
    <S:Header>
     <wsa:MessageID>http://Business456.com/guid/daa7d0b2-c8e0-476e-a9a4-d164154e38de
</wsa:MessageID>
       <wsa:To>http://fabrikam123.com/serviceB/123</wsa:To>
       <wsa:ReplyTo>
          <wsa:Address>http://Business456.com/serviceA/789
          </wsa:Address>
       </wsa:ReplyTo>
       <wsrm:Sequence>
          <wsu:Identifier>http://Business456.com/RM/ABC
          </wsu:Identifier>
          <wsrm:MessageNumber>2</wsrm:MessageNumber>
       </wsrm:Sequence>
    </S:Header>
    <S:Body />
</S:Envelope>
```

**Fig. 9**. WS-RM message example for Context-store access measurement

## 5.2 Experiment 1: Context-store Access Time

In this section, we present the time measurements to access the Context-store. To measure the time, we used the `setContext()` operation[6] of the information service. We measured the

[6] We choose the `setContext` operation as an example. Similar performance evaluation can be made for the

Round Trip Times of the Context-store accessing transactions. A mobile client sends a sample SOAP message with Web Service Reliable Messaging (WS-RM), and the Information Service responds back. The experiment setup is illustrated in **Fig. 8**. The size of the headers used in the test, which is shown in **Fig. 9**, is 847 bytes, and the entire SOAP message size is 1.58KB.

The measurement results were collected with the same configurations as the previous experiment through 200 iterations. **Table 3** shows the average values of the collected data.

**Table 3**. Summary of the measured Context-store accessing overhead

| | Set 1 (sec) | Set 2 (sec) | Set 3 (sec) | Set 4 (sec) | Set 5 (sec) | Ave. of Sets |
|---|---|---|---|---|---|---|
| Ave±error (sec) | 4.194±0.083 | 4.197±0.093 | 4.177±0.123 | 4.028±0.066 | 4.036±0.097 | 4.127±0.042 |
| StdDev (sec) | 0.457 | 0.511 | 0.676 | 0.363 | 0.530 | 0.516 |

### 5.3 Experiment 2: Evaluation of the performance measurement of the full SOAP message and optimized SOAP message with the use of Context-store

To demonstrate the effectiveness of using the Context-store, we measured the Round Trip Times ($T_{RTT}$) of both the full SOAP message and optimized message with the use of Context-store. As discussed in section 3, the applications in HHFR store unchanged and/or redundant parts of the SOAP message to the Context-store. By saving this metadata, the size of the message can be reduced and the performance of the messaging can also be increased. Before presenting the performance evaluation, we present a practical usage example of the Context-store, i.e., storing a message with WS-Addressing headers. We then present the measurement methodology and results.

A sample SOAP Header example: Our choice for a sample SOAP header comes from the WS-Addressing Specification [27]. The WS-Addressing Specification defines transport neutral mechanisms to address Web Services and messages. It defines two constructs that convey information between Web Service endpoints (e.g., reference-able entity, processor or resource). The two constructs are 1) endpoint references at which Web Service messages can be targeted and 2) message information headers; endpoint references convey information that identifies a Web Service endpoint (or individual message in some cases). For individual message addressing, the specification defines a family of information headers that allows the uniform addressing of messages. Message information headers, which are the second construct, convey end-to-end message characteristics, such as message identity, origin and destination of the message.

An application using HHFR framework stores unchanging and/or redundant SOAP parts into the Context-store (Information Service) and retrieves them when they are needed. So, we can store many of the WS-Addressing header parts to improve message communication performance. To support this idea, we present a practical example of this usage. Two Web Service endpoints, i.e., A (a service provider) and B (a mobile Web Service client), start a series of Web Service transactions. Endpoint B requires WS-Addressing headers only if it needs to send a reply or address an individual message. Thus, those headers that are unchanged for the rest of the stream can be archived in the Context-store. Among the elements of WS-Addressing header parts, <messageID> must not be archived because it is unique for each message. In this example, we also assume that there is no message referencing so that we can avoid leaving referencing items. The example scenario is depicted in **Fig. 10**, and **Fig. 11**

---

`getContext` operation.

shows a sample SOAP header used. Except <messageID> that is highlighted as bold face, most of the WS-Addressing headers can be removed from the message.

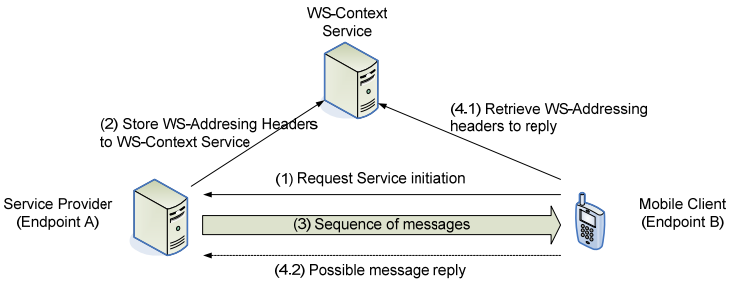



**Fig. 10**. Scenario for WS-Addressing example

```
<S:Envelope xmlns:S="http://www.w3.org/2003/05/soap-envelope"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
   <S:Header>
    <wsa:MessageID>
      uuid:6B29FC40-CA47-1067-B31D-00DD010662DA
    </wsa:MessageID>
    <wsa:ReplyTo>
      <wsa:Address>http://business456.example/client1</wsa:Address>
    </wsa:ReplyTo>
    <wsa:To>http://fabrikam123.example/Purchasing</wsa:To>
    <wsa:Action>http://fabrikam123.example/SubmitPO</wsa:Action>
   </S:Header>
   <S:Body/>
</S:Envelope>
```

**Fig. 11**. Sample SOAP message for the WS-Addressing example

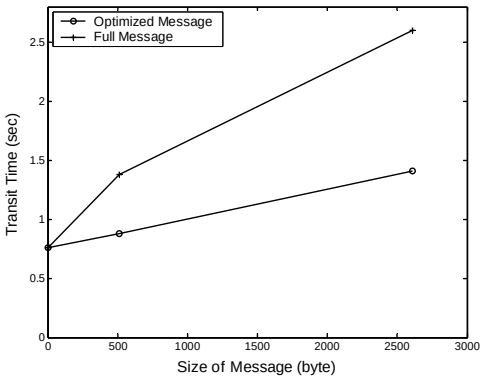



**Fig. 12**. Round trip time of optimized message exchange through the HHFR high performance channel compared with full message exchange

**Fig. 12** shows the Round Trip Time ($T_{RTT}$) of message exchange using the HHFR communication with the use of Context-store compared with the Round Trip Time without the the use of Context-store (i.e., with full header message transactions). Times are collected for 50 repetitions of three different sizes (2byte to 2.61KB). Two practical examples of Web Service headers are used in this measurement; the Round Trip Time for a large message are collected using a sample message with WS-Security headers, a sample message with

WS-Addressing headers is used for a medium size, and a SOAP message that has a body element with no data and no header is used for an empty message. It should be noted to clarify the testing environment that this experiment ran over a HHFR message stream. It is because this experiment is done to show the effectiveness of the HHFR framework that uses the Context-store. The results of the given examples show that we save 83% of message size on average and 41% of transit time on average by using our design. These results are shown in **Table 4**.

**Table 4**. Summary of the round trip time

| Message Size | Without Context-store | | With Context-store | |
|---|---|---|---|---|
| | Ave.±error | StdDev | Ave.±error | StdDev |
| Minimum: 2byte (sec) | 1.54±0.039 | 0.217 | 1.54±0.039 | 0.217 |
| Medium: 513byte (sec) | 2.76±0.034 | 0.187 | 1.75±0.040 | 0.217 |
| Large: 2.61KB (sec) | 5.20±0.158 | 0.867 | 2.81±0.098 | 0.538 |

## 5.4 Experiment 3: Scalability of the Context-store

In addition to these two tests (i.e., the test that measures the accession overhead and the test that measures the effectiveness of the Context-store in the HHFR), we also analyze the scalability of our approach to use the Context-store with multiple message streams.

We measured the time needed to finish a Context-store request transaction (i.e., $T_{time\text{-}in\text{-}server}$) and the time to process `setContext()`operation (i.e., $T_{wsctx}$) with various sizes of contexts. The results are shown in **Table 5**. Since $T_{wsctx}$ is less than one millisecond, **Fig. 13** shows only the measurement of $T_{time\text{-}in\text{-}server}$. Both $T_{time\text{-}in\text{-}server}$ and $T_{wsctx}$ increase linearly as the size of context increases.

**Table 5**. Summary of the average time to process Context-store message with Axis 1.2

| Size of Context (bytes) | $T_{time\text{-}in\text{-}server}$ (msec) | | $T_{wsctx}$ (msec) | |
|---|---|---|---|---|
| | Ave±error | StdDev | Ave±error | StdDev |
| 1220 | 35±0.43 | 4 | 0.501±0.005 | 0.048 |
| 1320 | 63±0.54 | 5 | 0.498±0.005 | 0.044 |
| 1520 | 115±0.92 | 9 | 0.531±0.013 | 0.120 |
| 1720 | 174±1.64 | 16 | 0.508±0.005 | 0.050 |
| 1920 | 227±1.35 | 13 | 0.528±0.012 | 0.118 |
| 2120 | 293±2.01 | 19 | 0.517±0.012 | 0.118 |

With our measurements, we demonstrate how to calculate the maximum number of simultaneous stream supported by our approach to use the Context-store. First, we assume the stream length as 10 minutes, i.e., $T_{stream}$ = 600 seconds. Then, provided with formula 3 and $T_{time\text{-}in\text{-}server}$ time for a 1220 byte-size context, we can calculate the maximum number of simultaneous streams as follows:

$$N \approx \frac{600}{3 \times 0.035} \tag{4}$$

$$N \approx 5700 \tag{5}$$

Thus, if we have 100byte-size context, one server can support a maximum of 1600 streams. However, it should be noted that this illustration is based on the assumption that a web Server can handle this many simultaneous connections.

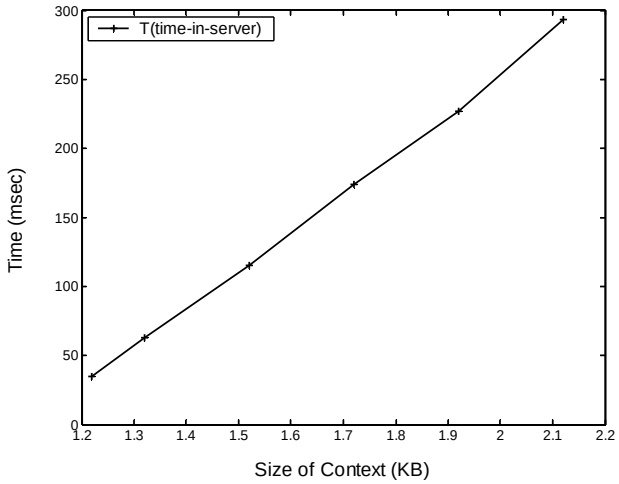



**Fig. 13**. Time to finish Context-store request message processing (i.e.,$T_{time\text{-}in\text{-}axis}$)

### 5.5 Discussions: Findings from the Experiments

As noted in 5.1, we design the evaluation to show the effectiveness of using Context-store in HHFR architecture. This evaluation does not include the performance comparison between with Context-store and without Context-store and there is a reason for it. Since the HHFR design assumes the redundant and static parts of the messages is removed from the exchanging messages and stored to the Context-store, it is obvious that there is less communication overhead when a client and a service exchange messages with some parts removed (i.e. using the Context-store in the HHFR). Thus, the issue here should be rather how long the session should be to get the direct benefit from the Context-store use in HHFR than whether we have performance gains or not. We conducted this evaluation in Ref [7] and for the given environment; we calculated the break-even point that is point below which the conventional Web Service communication framework outperforms the HHFR. If there will be more than four messages in a row in the given application session, the HHFR will perform better in mobile Web Service [7].

However, we conduct evaluations which are specialized for this version of HHFR with the Context-store too. The second evaluation we conduct is to measure Round Trip Times (RTTs) for both conventional Web Service messaging with full XML content body and optimized HHFR messaging with the Context-store. As it is shown in **Fig. 12**, the transit time between a service client and a service provider is saved by 41%. We analyze the result and find the RTT saving is mostly coming from the size reduction which is 83% gain for our test cases. Thus we conclude the biggest performance benefit of the Context-store to the HHFR is to reduce the size of the messages after all by letting them include only essential parts of the SOAP messages.

Required characteristics to verify the effectiveness of the Context-store also includes minimal communication overheads to access the Context-store and scalability of the Context-store. To measure the communication overhead accessing the Context-store in regular cellular networks, we have measured the response time of the `setContext()` operation of the Context-store. The evaluation is conducted over 14.4kbps cellular networks and the average access time is about 4 seconds, however $T_{time\text{-}in\text{-}server}$ is about 100msec and pure network overhead is 3.9 sec. Thus, we expect smaller access time to the Context-store, if we have better connections like 3G cellular network (144kbps, ten times faster than current

evaluation environment). Finally we conduct scalability test for the Context-store implementation. As we derived formula (3) from formula (1) and (2), we get necessary parameters like $T_{time\text{-}in\text{-}server}$ and $T_{stream}$ from the evaluation tests. Using the numbers from **Table 4**, we can derive one Context-store server can support up to 1600 streams with the given machine configuration.

## 6. Conclusion

In this paper, we have identified an important factor to improve communication performance of mobile Web Service is addressing the messaging scheme at system level. There have been many researches which try to optimize a message representation; however they may not interoperable with the current Web Service framework. We design HHFR architecture to address the problem from the system level view and make the architecture compatible with the current architecture by supporting conventional Web Service messaging as an option.

We have presented the extension of our HHFR architecture that integrates the online Web Service Repository with the existing message optimization framework. We argue that the right way to address message size reduction in mobile computing is by saving redundant or unchanging SOAP message parts (metadata about messages) into the Context-store for retrieval as needed: (a) bandwidth saving. Other benefits we can have from the use of Context-store are (b) supporting reliable recovery from the failure and (c) enhancing interoperability. Through saving negotiation information, unchanging message parts and session state context to more stable repository, mobile Web Service participants retrieve and restore any given session state if it has lost session connections. If we add a periodical logging feature to the architecture and mandate participants to save session history, then the reliability of the overall system will be highly increased. Also, WS-Context and UDDI compliant Context-store will make the system a more interoperable environment. To the Web Services and their clients that are capable of understanding WS-Context or UDDI specification, HHFR architecture with the Context-store provides not only message optimizing capabilities but also a context repository (i.e. an online Web Service database).

However, there is a limitation on the proposed architecture as well. As mentioned in the paper, the HHFR architecture assumes that the mobile clients interact with Web Services in a session style. That is, they exchange messages continuously and messages are similar or the same. Thus, if a client and a service exchange message that are different from each other, HHFR architecture adds more overheads (i.e. negotiation stage and context-store access) than gives the optimization benefit to the message exchange.

The evaluation results and the analysis show that we can expect notable performance gains with the Context-store in comparison to without the Context-store system. Also, we show the access time to the Context-store is in affordable range. Since the access time is heavily dependent to the wireless connection speed, the new access time with faster connections like 3G cellular networks will be the negligible cost to the overall cost.

## References

[1] M. P. Papazoglou and D. Georgakopoulos, “Service-oriented computing,” *Communications of the ACM*, vol. 46, no. 10, pp. 25-28, Oct. 2003.

[2] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-oriented computing: state of the art and research challenges,” *IEEE Computer Magazine*, vol. 40, no. 11, pp. 38-45, Oct. 2007.

[3] S. N. Srirama, M. Jarke, and W. Prinz, "Mobile web service provisioning," in *Proc. of on AICT/ICIW*, pp. 120, 2006.

[4] H. Chu, C. You, and C. Teng, "Challenges: wireless web services," in *Proc. of on ICPADS 2004*, pp. 657-664.

[5] P. Sandoz and S. Pericas-Geertsen, "Fast infoset @ Java.net," in *Proc. of XTech 2005*.

[6] R. Carroll, D. Virdee, and Q. Wen, "Developments in BinX, the binary XML description language," in *Proc. of the UK e-Science All hands Meeting 2004*, Nottingham UK, Sept. 2004.

[7] S. Oh and G. Fox, "Optimizing web service messaging performance in mobile computing," *Future Generation Computer System*, vol. 23, no. 4, pp. 623-632, 2007.

[8] M. Little, E. Newcomer, and G. Pavlik, "Web services context specification (WS-Context) Version 1.0," Apr. 2007, http://docs.oasis-open.org/ws-caf/ws-context/v1.0/wsctx.pdf

[9] M. Adacal and A. B. Bener, "Mobile web services: a new agent-based framework," *IEEE Internet Computing*, vol. 10, no. 3 pp. 58-65, May-June, 2006.

[10] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, and J. H. Schiller, "Performance considerations for mobile web services," *Computer Communications*, vol. 27, no. 11, pp. 1097-1105, 2004.

[11] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," in *Proc. of ACM SIGMOD 2000*, Dallas, TX, USA, May 2000.

[12] M. Girardot and N. Sundaresan, "Millau: an encoding format for efficient representation and exchange of XML over the web," in *Proc. of on the 9th International World Wide Web Conference WWW2000*, Amsterdam Netherland, May 2000.

[13] D. Sosnoski, "Improve XML transport performance part 1 and 2," *IBM developersWork Article*, June 2004.

[14] M. Aktas, G. Fox, M. Pierce, and S. Oh, "XML metadata service," *Concurrency and Computation: Practice and Experience*, vol. 20, no. 7, pp. 801-823, 2008.

[15] W. Lu, K. Chiu, and D. Gannon, "Building a generic SOAP framework over binary XML," in *Proc. of on HPDC 2006*, pp. 195-204.

[16] E. Serin, "Design and test of the cross-format schema protocol (XFSP) for networked virtual environments," M.S. thesis, Naval Postgraduate School, Monterey, CA, USA, Mar. 2003.

[17] E. Serin and D. Brutzman, "XML schema-based compression (XSBC)", http://xmsf.sourceforge.net/xsbc.html

[18] K. Chiu, T. Devadithya, W. Lu, and A. Slominski, "A binary XML for scientific applications," in *Proc. of on e-Science 2005*, pp. 336-343.

[19] T. Bellwood et al., "UDDI Version 3.0.2," UDDI specification technical committee, http://www.uddi.org/pubs/uddi_v3.htm

[20] R. Chumbley et al., "WS-interoperability: basic profile version 1.2," Mar. 2010, http://ws-i.org/profiles/BasicProfile-1.2-WGD.html

[21] R. Levenshteyn and I. Fikouras, "Mobile services interworking for IMS and XML web services," *IEEE Communications Magazine*, vol. 44, no. 9, pp. 80-87, Sept. 2006

[22] Department of Defense, "Unified capabilities requirements 2008 (UCR 2008)," Dec. 2008.

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

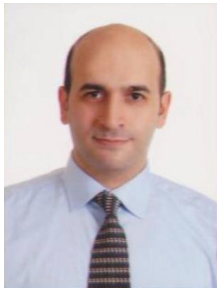
[24] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. V. Engelen, and M. J. Lewis, "Toward characterizing the performance of SOAP toolkits," in *Proc. of 5th IEEE/ACM International Workshop on Grid Computing*, Pittsburgh, Nov. 2004.

[25] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the limits of SOAP performance for scientific computing," in *Proc. of 11th IEEE International Symposium on High Performance Distributed Computing HPDC-11*. Edinburgh UK. Jul. 2002.

[26] kSOAP, http://ksoap2.sourceforge.net/

[27] D. Box et al., "Web service addressing (WS-Addressing)," Aug. 2004, http://www.w3.org/Submission/ws-addressing/

**Sangyoon Oh** received Ph.D. in Computer Science Department from Indiana University at Bloomington, U.S.A. He is an assistant professor of School of Information and Computer Engineering at Ajou University, South Korea. Before joining Ajou University, he worked for SK Telecom, South Korea. His main research interest is in the design and development of web based large scale software systems and he has published papers in the area of mobile software system, collaboration system, Web Service technology, Grid systems, and Service Oriented Architecture (SOA).



**Mehmet Aktas** received his Ph.D. degree in Computer Science from Indiana University in 2007. During his graduate studies, he worked as a researcher in Community Grids Laboratory of Indiana University in various research projects for six years. Before joining Indiana University, Dr. Aktas attended Syracuse University, where he received his M.S. degree in Computer Science and taught undergraduate-level computer science courses. He is currently working as a project manager in the Information Technologies Institute of Tubitak - Marmara Research Center. He is also part-time faculty member in the Computer Engineering Departments of Marmara University and Istanbul Technical University, where he teaches graduate-level computer science courses. His research interests span systems, data and Web science.

**Geoffrey Charles 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.