Efficiency Benefits in Mobile Web Service Architecture Using Context-store

Sangyoon Oh[1] *Member*, **Mehmet S. Aktas**[2] *Non-member*, **and Geoffrey C. Fox**[3] *Non-member*
[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 October 5, 2009; revised November 10, 2009; accepted November 20, 2009; published December 25, 2009

Abstract

Mobile computing became general computing environment as mobile devices getting powerful and wireless connectivity improves dramatically. As well, Ubiquitous Web Service becomes more viable because of the success in mobile computing. However, there are few huddles remains to integrate Web Services with mobile devices such as battery-life problem and messaging efficiency issue. We proposed and implemented a novel Web Service architecture, HandHeld Flexible Representation for mobile computing to optimized messaging representation by separating data content from the syntax. In this paper, we extend our HHFR to increase messaging efficiency and improve reliability by introducing an online Web Service Repository. In the new architecture, Static and/or redundant parts of Web Service message are stored and retrieved to/from the Context-store. Mobile Web Service participants will get the performance benefits as well as handling failures better on rather unstable mobile devices and services. We describe our architecture and evaluate our approach by testing the performance of the resulting system. The empirical results show that our framework outperforms conventional Web Services with mobile clients as well as capable to be extended to Internet scale with minimum overhead added.

Keywords: Mobile Web Service, XML, Online Web Service Repository, Handheld Flexible Representation (HHFR), Web Service Framework

This research was supported by the MKE(The Ministry of Knowledge Economy), Korea, under the ITRC(Information Technology Research Center) support program supervised by the NIPA(National IT Industry Promotion Agency" (NIPA-2009-C1090-0902-0003).

DOI: 10.3837/tiis.0000.00.000

1. Introduction

Web Service has emerged as a de-facto standard for Service Oriented Architecture in recent years [1][2]. It also profoundly affects overall distributed computing area. Like its predecessors, such as CORBA, RMI and DCOM, its primary goal is to inter-relate distributed functionalities. But, unlike its predecessors, it achieves its goal in elegance and neutral manner; it provides well-defined interfaces for distributed functionalities, which are independent of the hardware platform, the operating system, and the programming language. So distributed functionalities, or services, which may be running on different hardware platform, may be running in different operating systems, or may be written in different programming language, 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 interoperable capability.

While 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 a cellular phone, a smart phone (e.g. iPhone), and a handheld game console (e.g. Playstation Portable). Web Service technology recognizes mobile computing as an area where it should expand to [3]. Through integration, Web Services enables pervasive accessibility by acquiring mobility as it overcomes physical location constraint of the conventional computing. Meanwhile, mobile computing also requires a technology that connects mobile systems to a conventional distributed computing environment. Web Services may be the perfect candidate for such connection, since a strong interoperable capability is the key requirement of the technology. This will be important for its success when we consider the fact that the mobile computing environment is much heterogeneous in terms of hardware platform, operating system, and programming language. This, the integration of mobile computing with Web Services technology will give many advantages to both sides [3][4].

However, despite the fact that the condition of mobile computing has so much in improved recent years, there are fundamental differences between wireless and wired environment including intermittent connections, limited processing power, and battery-life problem on the wireless side. Thus, applying current Web Service communication models to mobile computing may result in unacceptable performance overheads. This potential problem comes mostly from XML's verbosity. The interoperability of Web Services mainly comes from its Extensible Markup Language (XML) based open standards. However, its self-descriptive characteristic causes two problems in the mobile computing environment. 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, which is the same for known participants. This causes the message size increase and 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. messaging scheme) to prevent performance degradations in not only mobile computing, but also conventional computing that is interacting with mobile applications. The optimized architecture (i.e. an ultimate solution) should provide following capabilities: 1) an optimized information representation to minimize the size of messages, 2) a streaming style message

exchange, which is clearly different from the request-response style of the current HHTP, and 3) an online Web Service Repository where participants are able to store static or redundant parts of the message.

There have been many proposals that address possible performance degradation. Spectrums are wide from technical approaches like binarization of XML messages [5][6] to abstract user modeling approaches to maximize user experience (e.g. better application response time and application startup time) [4]. However, they 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. Our preliminary results are presented in Ref [7][8]. Mobile applications using HHFR can negotiate characteristics for message stream and representation, and exchange messages in an optimized streaming fashion. By distinguishing between message semantics and syntax, the architecture provides an overall system framework for Web Services applications in mobile computing environment. The HHFR architecture provides two required capabilities of the ultimate architecture: an optimized information representation and a streaming style message exchange. The empirical results show that our HHFR architecture outperforms conventional Web Services with mobile clients.

In this paper, we propose an extended version of HHFR to increase a messaging efficiency and improve our scheme's reliability by introducing an online Web Service Repository, which is a third capability of the ultimate solution. Expected benefits to the architectures are following. First, the primary and the most important benefit is the performance gain. If the mobile application exchanges consecutive messages with a Web Service, we may store static and redundant part of the message to the repository and reduce a message size. Second, by saving negotiation information, unchanging message parts, and session history to the more reliable repository, we can handle better failures on rather unstable mobile devices and services. Finally, the repository we adapt is UDDI and WS-Context specification [9] conformance. Thus we expect adapting makes our HHFR architecture expansion interoperable (i.e. collaborate) with other clients with UDDI and WS-Context compatible-composite applications.

We organize this paper as follows. In Section 2, we discuss related works that address obstacles of mobile Web Service. We overview our HHFR architecture design in Section 3 and illustrate the expended version of HHFR with online Web Service Repository expansion in detail in Section 4. In Section 5, we show our empirical results and we discuss and conclude in Section 6.

2. Related Works

In this section, we present background on our extended version of the HHFR architecture, presenting previously, and on-going efforts, which address obstacles of mobile Web Service and XML performance limitations. Web Service has been widely used to integrate distributed components. Its language and platform neutral characteristics are well fitted to the heterogeneous distributed computing environment. However, there has also been concerns about its communication overhead caused by verbose XML and its complex mechanism to register and utilize service descriptions. In this section, we overview research efforts that addresses performance issue of XML based Web Service technology.

Approaches to improve performance of mobile Web Services are categorized as either naïve compressing message or binarization of messages. Compressing message approaches utilize various compressing mechanism to provide smaller sized message to reduce bandwidth usage of constraint wireless communication channel. Tian et al. studied mobile Web Services environment and pointed performance concerns about the XML messaging efficiency [10]. The experiment shows that 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 message. Developed by Dennis Sosnoski, XBIS [17] 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 the XML document in the same order they appear in the text. Like other repetitive words replacement schemes, it defines each name as text only once, and then uses a handle value to refer back to the name when it is repeated. However these naïve approaches do not address fundamental problems of mobile Web Service environment as well as adding one more layer of processing (compression-decompression) to less-capable mobile CPU.

Thus another stream of researches for improving mobile Web Service environments is focusing on utilizing binarization mechanism of XML. Whether it is self-contained (maintaining self-descriptive characteristics of XML) or not, there has been a lot of studies and proposals [5][7][8][13]. 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 XML Document [5]. Cross Format Schema Protocol (XFSP) [14] is another project that serializes XML documents based on a schema. Initially, it was created to provide as 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) [15], XFSP provides binary serialization and a parsing framework. ExtremeLab of Indiana University presents their research about binary XML for scientific application (BXSA) [13] that has a new structure and layered format based on XBS [16].

We have presented series of related works, which is focused on optimizing message representations. However, there have not been many researches, which address mobile Web Service's performance issue from the system level perspective rather they focus on single problem at a time. Especially, study about integrating Web Service repository to improve mobile Web Service's performance is not done yet. In this paper, we propose a novel architecture based on our original message optimizing architecture, HHFR and detailed study about utilizing online Web Service repository to 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 a 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 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 prototype implementation and detailed empirical evaluation results are presented in [7][8].

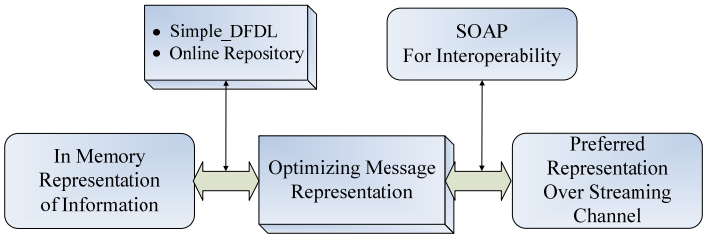

Fig. 1. An Abstract Overview Diagram of HHFR

3.1 Motivation and Design Overview

In many cases, mobile applications are interacting with a Web Service in session style. That is, they exchange messages continuously and messages are in 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 exchanges messages with the same message format and the only context values are changing. A mobile client in a Video/Audio conferencing using Unified Communication environment [18] are going to receive and send voice and image data in 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 interact and exchange messages in a session, most of messages are in the same representation (i.e. structure and format) except a starting message and an ending message. Messages in the middle are in the same structure but changing values in them as new information is produced. Also, messages in the session may include some information, which is repeating during the session.

For the domain of application that uses session style, we propose a novel mobile Web Service architecture HandHeld Flexible Representation, which utilizes the characteristic of this repeating structure and information. In mobile computing environment, where mobile clients and services are co-exists, usage scenario of HHFR is as follows: Web Service participant initiates a stream, which is a series of message exchanges 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, 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 the HHFR uses this negotiation to establish a message stream.

There are three key design points of the HHFR architecture, which make the message exchanges in HHFR efficient. Firstly, HHFR uses a Data Format Description Language (DFDL) [19]-style data description language, named the Simple_DFDL, to represent a message structure and type. The 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 readers can find detailed information about Simple_DFDL in [7].

Secondly, in the HHFR, applications exchange messages in a streaming style. The HHFR sets up a message stream between the participants based on the characteristics negotiated. The message exchange is then freed from "waiting for response" by adapting an asynchronous messaging style. Thirdly, in the 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) negotiated characteristics of a stream. By storing the message fragment or meta-data of the stream as context, the application can exchange slimmed down messages that contain only the vital part of the message content without losing the formal ability to be able to produce the conventional SOAP message representation on demand.

3.2 Replacement of XML Syntax with Optimized Representation: Simple_DFDL

By separating message semantics from syntax, the 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 overall performance of 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) [20]. A binary message representation also helps the HHFR architecture to avoid textual conversion. The architecture simplifies the conventional encoding/decoding stage[1], 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 [21].

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. They 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, and message syntax and message, which is generated from the given Simple_DFDL document.

Simple_DFDL, which is a simple restricted XML Schema Definition (XSD), is our method of defining the XML syntax of the 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 following:

- 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.

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

- The Simple_DFDL do 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 the 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>
```

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. But 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 a message format A, the service may process received format B message if the conversion process overhead is higher than the threshold that is defined in the HHFR design specification.

In a conventional Web Service environment, XML is the representation format, which provides interoperability to the heterogeneous participating nodes. Yet in some constrained computing environment, processing an XML format message becomes a performance bottleneck because of its verboseness. The preferred representation concept of the HHFR architecture can provide an optimized representation for 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. Secondly, to set up a fast and reliable means of communication, the architecture should negotiate the characteristics of the stream.

A stream of messages shares the same representation, meaning, thus 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 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[2], which 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.

Compared to the conventional Web Service communication method, the negotiation stage is an additional overhead[3] in the HHFR architecture, and this will discourage use of the scheme in a short message stream, which has few exchanged messages. However, for larger message streams with many of redundant messages, the HHFR architecture's negotiation overheads are negligible.

4. Web Service Repository: Context-store in HHFR

In this section, we detail our extended version of HHFR with Context-store design and implementation. We describe the motivation of a Context-store design and design characteristics. In the following section, we provide our empirical experiment result using implemented service to validate the design efficiency of the Context-store and it effect to the overall architecture.

4.1 Motivation and Design Characteristics

As we discussed, a Context-store is an essential components 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, such as 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. The overview of HHFR extension architecture is depicted in **Fig. 2**.

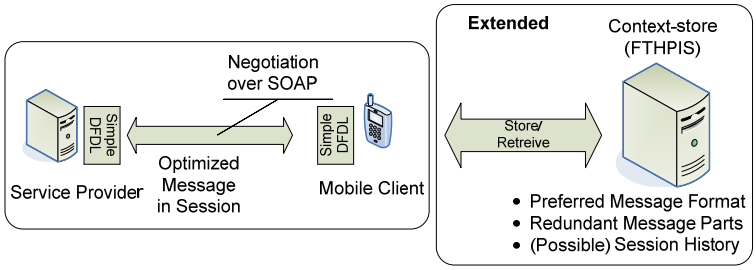

Fig. 2. An Overview of Extended HHFR

[2] If the negotiation can be continued until the two participants reach a single agreed upon point.
[3] Others are a Repository (i.e. Context-store) accessing overhead and Simple_DFDL designing overhead.

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 a 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 a 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 [9]. 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, participating nodes can archive 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.

Community Grids Lab (CGL) developed the information service [8], which is based on the WS-Context specification defined by OASIS as a part of Fault Tolerant High Performance Information System Project[4]. The WS-Context implementation supports static, semi-dynamic data, and dynamic data. Combined with the extended UDDI service, forms a hybrid information service that is applied to the Geographical Information System (GIS) service [22] at CGL, the Global MultiMedia Collaboration System (GlobalMMCS) [23], or the Sensor Grid.

As noted, for the mobile Web Service applications, information service (i.e. Context-store) works as an online Web Service repository and it gives two major advantages. First, it allows applications to reduce a size of message in efficient way. As discussed earlier, static data from the messages in a session can be stored and retrieved. However, the Context-store, which is WS-Context conformance, supports the store/retrieve capability in simple and clear fashion. The different types of data can be stored and retrieved in a single data format through the service interface. Second, it provides a persistent storage service for faulty wireless environment.

4.2 Context-store in Mobile Web Service

In this section, we describe our implementation of Context-store for the mobile Web Service applications. 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 the 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 the HHFR brings a dependency on SOAP-Java binding on the Apache Axis library. The Axis version for the JME (Java Micro Edition) environment or any other popular programming environment for small and embedded devices is not developed yet and won't 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

[4] For a more detailed description of the project, visit http://www.opengrids.org/wscontext

porting the code. Unfortunately, replacing JSE APIs with JME APIs isn't 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 [24] for those processes. SOAP 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 to generate SOAP messages that generated 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 doesn't 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. 3(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. 3(a) shows the `getContext` SOAP request message of the conventional WS-Context client using Axis and **Fig. 3(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>
```

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

Fig. 3(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 :type="d:string">
                context://hhms/sangyoon</context-identifier>
             </context>
          </body>
      </n0:getContents>
   </v:Body>
</v:Envelope>
```

Fig. 3(b). getContext() SOAP request message created using kSOAP for the mobile WS-Context client. Elements resulted from the piece of Java code is highlighted in boldface.

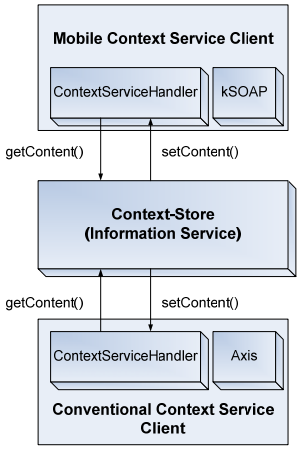

Fig. 4. Mobile and conventional context service clients

As depicted in **Fig. 4**, the two primary WS-Context related functionalities of Information Services are `getContent()` and `setContent()` methods, which provide access and store operations that is 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 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 an unique identifier, and 3) retrieve context. `ContextServiceHandler` object is a wrapper class and provides `getContent()` and `setContent()` methods.

```
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…
}
```

`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 be occurred when network failed or operational error.

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. Compare to automatic Java binding generation of Axis, the method also imposes the human error involvement when the multi-referenced SOAP is converting into flattened structure.

5. Performance Evaluation.

The goal of this 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.

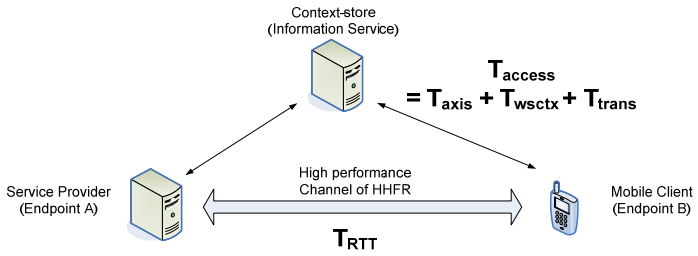

Fig. 5. System parameters

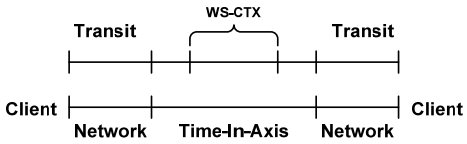

Fig. 6. System parameters with time frame

5.1 Evaluation Model and Test Environment

In the evaluation, we focus to measure and analyze three values. The three values are: a time to finish a Context-store access from a mobile client (i.e. a time between a SOAP request and a SOAP response), bandwidth (time) gain from using a Context-store, and the scalability of our approach to use a Context-store. Thus, we design the evaluation measurements to have three aspects: First, we measure a 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. This measurement is distinguished from other two because it uses a high performance channel of HHFR to exchange message and the first and the third experiments use a conventional SOAP message for measurements. Third, we analyze a scalability of our approach by measuring a time to take to process a WS-Context SOAP message on the service side. **Table 1** shows a summary of our measurements.

For 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 Axis server
- T_{trans}: time consumed to transmit message over network
- T_{stream}: length of 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. 5** and **Fig. 6** 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 access 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 Context-store. Let's consider N simultaneous streams happening during the time period of T_{stream}. Thus we can formulize the calculation of the scalability of our approach to use Context-store, which is the maximum number of supported simultaneous streams as following:

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

$$N \approx \frac{T_{stream}}{3 \times T_{time-in-server}} \quad (3)$$

It should be noted that there are three major parameters on which our evaluation analysis depends on. First, T_{access} is governed by T_{trans} that can vary from 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 message processing overhead (i.e. $T_{time\text{-}in\text{-}server}$). Finally, the stream length (i.e. how long an application usage session last) is also an important parameter to analyze the 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 stream length is, the more simultaneous streams can be supported.

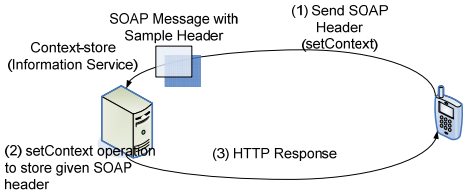

Fig. 7. Set up for measuring Context-store accessing overhead

```
<?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. 8. 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 a Context-store. To measure the time, we used the `setContext()` operation[6] of the information service. We measured Round Trip Times of the Context-store accessing transactions. A mobile client sends a sample SOAP

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

message with Web Service Reliable Messaging (WS-RM) and the Information Service responds back. The experiment setup is illustrated in **Fig. 7**. The size of the headers used in the test, which is shown in **Fig. 8**, 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 2** shows the average values of the collected data.

Table 2. 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 performance measurement of the full SOAP message and optimized SOAP message with Context-store usage

To demonstrate the effectiveness of using the Context-store, we measured the Round Trip Times (T_{RTT}) of both the full SOAP message and the optimized message with Context-store usage. As discussed in section 3, 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. Then we present the measurement methodology and results.

A sample SOAP Header example: Our choice for a sample SOAP header comes from the WS-Addressing Specification [25]. The WS-Addressing Specification defines transport neutral mechanisms to address Web Services and messages[7]. 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 store unchanging and/or redundant SOAP parts to the Context-store (Information Service) and retrieve 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 a 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. 9**, and **Fig. 10** shows

[7] The definition is from the WS-Addressing Specification [136]

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

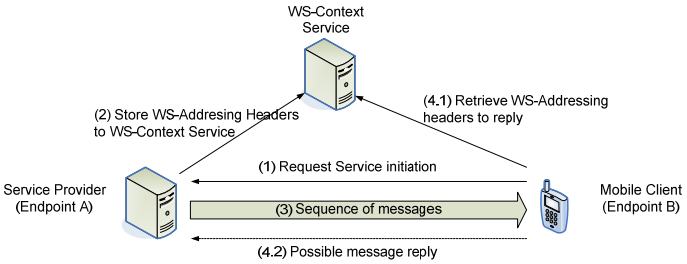

Fig. 9. Scenario for WS-Addressing example

```
<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. 10. Sample SOAP message for WS-Addressing example

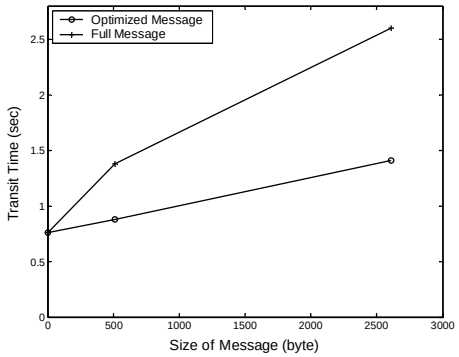

Fig. 11. Round trip time of optimized message exchange through the HHFR high performance channel compared with full message exchange

Fig. 11 shows the Round Trip Time (T_{RTT}) of message exchange using the HHFR communication with the Context-store usage compared with the Round Trip Time without the Context-store usage (i.e. with full header message transactions). Times are collected 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 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 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 3**.

Table 3. 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 a time to finish a Context-store request transaction[8] (i.e. $T_{time\text{-}in\text{-}server}$) and a time to process `setContext()`operation (i.e. T_{wsctx}) with various size of contexts. They are shown in the **Table 4**. Since T_{wsctx} is less than one millisecond, **Fig. 12** 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 4. 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 the 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 following:

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

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

Thus, if we have 100byte-size context, one server can support maximum 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.

[8] Resolution of the Java timer used in this experiment is one millisecond.

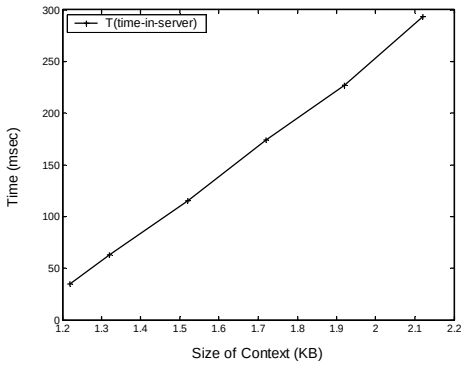

Fig. 12. Time to finish Context-store request message processing (i.e.,$T_{time\text{-}in\text{-}axis}$)

6. Discussion and Conclusion

In this paper, we have presented the extension of our HHFR architecture, which integrates online Web Service Repository with existing message optimization framework. We argue that the right way to address reducing the message size in the mobile computing is to save redundant or unchanging SOAP message parts (metadata about messages) to the Context-store and retrieve when it is needed. Our approach can increase efficiency of message exchange, since messages share many SOAP parts in the stream.

We design the Context-store based on the WS-Context Specification [26] and evaluated performance gains, which comes from the Context-store usage by comparing the transit time of a minimized message after storing redundant or unchanging SOAP parts of the message and the original. The example message is chosen from WS-Addressing Specification. The empirical result shows that the size is reduced to average 17% (83% gains) and transit time is saved in 41%. The Context-store accessing adds maximum 100msec overhead, which is 2.5% of adding to the RTT of the same size message delivery, to RTT of the dummy Web Service.

In addition to the messaging efficiency, we also argue that the addition of Context-store to the HHFR architecture can handle failures better on rather intermittent wireless computing environment. Through saving negotiation information and unchanging message parts 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, reliability of the overall system will be increased very much.

References

[1] Mike P. Papazoglou, "Service -Oriented Computing: Concepts, Characteristics and Directions," In Proceedings on Fourth International Conference on Web Information Systems Engineering (WISE'03), pp.3, Roma, Italy, 2003.

[2] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, Frank Leymann, "Service-Oriented Computing: State of the Art and Research Challenges," IEEE Computer Magazine, Vol. 40, No. 11, pp. 38-45, Oct. 2007.

[3] Satish Narayana Srirama, Matthias Jarke, Wolfgang Prinz, “Mobile Web Service Provisioning,” In Proceedings on AICT/ICIW, pp. 120, 2006.

[4] Hao-Hua Chu, Chuang-wen You, Chao-ming Teng, "Challenges: Wireless Web Services," ICPADS 2004, pp. 657-664.

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

[6] R. Carroll, D. Virdee, and Q. Wen, "Developments in BinX, the Binary XML description language," In Proceedings of the UK e-Science All hands Meeting 2004, Nottingham UK, September 2004.

[7] Sangyoon Oh, Geoffrey Fox, "Optimizing Web Service messaging performance in mobile computing," Future Generation Computer System Vol. 23, No 4, pp. 623-632, 2007.

[8] Mehmet Aktas, Geoffrey Fox, M.arlon Pierce, Sangyoon Oh, "XML Metadata Service," Concurrency and Computation: Practice and Experience Vol. 20, No 7, pp. 801-823 2008.

[9] B. Bunting, M. Chapman, O. Hurley, M. Little, J. Mischinkinky, E. Newcomer, J. Webber, and K. Swenson, "Web Services Context (WS-Context),"

[10] Min Tian, Thiemo Voigt, Tomasz Naumowicz, Hartmut Ritter, Jochen 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 Proceedings 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 Proceedings on the 9th International World Wide Web Conference WWW2000, Amsterdam Netherland, May 2000.

[13] Wei Lu, Kenneth Chiu, Dennis Gannon, "Building a Generic SOAP Framework over Binary XML," In Proceedings on HPDC 2006, pp. 195-204.

[14] 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, March 2003.

[15] E. Serin and D. Brutzman, "XML Schema-Based Compression (XSBC)", http://www.movesinstitute.org/xmsf/projects/XSBC/03Mar_Serin.pdf, Access Date: 10/06/2009

[16] Kenneth Chiu, Tharaka Devadithya, Wei Lu, Aleksander Slominski, "A Binary XML for Scientific Applications," In Proceedings on e-Science 2005, pp. 336-343.

[17] D. Sosnoski, "Improve XML Transport performance Part 1 and 2," IBM developersWork Article, June 2004. http://www-128.ibm.com/developerworks/xml/library/x-trans1.html.

[18] Department of Defense, "Unified Capabilities Requirements 2008 (UCR 2008)," December 2008.

[19] M. Beckerle, and M. Westhead, "GGF DFDL Primer", http://www.gridforum.org/Meetings/GGF11/Documents/DFDL_Primer_v2.pdf, Access Date: 10/06/2009

[20] M. Govindaraju, A. Slominski, K. Chiu, P. Liu, R. V. Engelen, and M. J. Lewis, "Toward Characterizing the Performance of SOAP Toolkits," In Proceedings of 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, November 2004.

[21] K. Chiu, M. Govindaraju, and R. Bramley, "Investigating the limits of SOAP performance for scientific computing," In Proceedings of 11th IEEE International

Symposium on High Performance Distributed Computing HPDC-11. Edinburgh UK. July 2002.

[22] Galip Aydin, Ahmet Sayar, Harshawardhan Gadgil, Mehmet S. Aktas, Geoffrey Fox, Sunghoon Ko, Hasan Bulut, Marlon E. Pierce, "Building and applying geographical information system Grids," Concurrency and Computation: Practice and Experience, Vol. 20, No. 14, pp. 1653-1695, 2008.

[23] Wenjun Wu, Hasan Bulut, Ahmet Uyar, Geoffrey Fox, "Adapting H.323 Terminals in a Service-Oriented Collaboration System," IEEE Computing Magazine, Vol. 9, No. 4, pp. 43-50, July-August, 2005.

[24] kSOAP, http://ksoap2.sourceforge.net/, Access Date: 10/06/2009.

[25] D. Box et al., "Web Service Addressing (WS-Addressing), August 2004, http://www.w3.org/Submission/ws-addressing/, Access Date: 10/06/2009

[26] Community Grids Lab, Fault Tolerant High Performance Information System (FTHPIS), http://www.opengrids.org/extendeduddi/index.html, Access Date: 10/06/2009

Sangyoon Oh (82-10-3129-7022 syoh@ajou.ac.kr)

Oh received M.S. in Computer Science from Syracuse University, U.S.A. and Ph.D. in Computer Science Department from Indiana University at Bloomington, U.S.A. He is an assistant professor of Div. of Information and Computer Engineering at Ajou University, South Korea. He previously held a research position at 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 (+90-212-6423434 mehmet.aktas@bte.mam.gov.tr)

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. During this time period, Aktas has worked for a number of prestigious research institutions ranging from NASA Jet Propulsion Laboratory to Los Alamos National Laboratory. Before joining the Indiana University, 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 into systems, data and Web science.

Geoffrey Charles Fox (1-812-219-4643 gcf@indiana.edu)

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