WEB SERVICE ARCHITECTURE FOR MOBILE COMPUTING

Sangyoon Oh

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree
Doctor of Philosophy
in the Department of Computer Science,
Indiana University
August 2006

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Doctoral Committee

_____
Dr. Geoffrey C. Fox (Principal Advisor)

_____
Dr. Dennis Gannon

_____
Dr. Andrew Lumsdaine

_____
Dr. Sun Kim

August 1st, 2006

Acknowledgements

First and foremost, I am sincerely grateful to my advisor, Dr. Geoffrey C. Fox for his guidance and encouragement to this dissertation as well as my entire research career at Indiana University. With his keen insight and extensive experience, he helped me to focus on the important details. Working with him has been one of most rewarding experiences.

I am also very thankful to my other committee members. I would like to thank Dr. Andrew Lumsdaine for his valuable feedback and Dr. Sun Kim for his comments and suggestions. I am particularly thankful to Dr. Dennis Gannon for his constructive criticism on my research.

I have had the great pleasure of working with a wonderful set of people at Community Grids Lab. I owe a great deal of thanks to Mehmet Aktas for providing technical support for FTHPIS (Fault Tolerant High Performance Information Service) which I used in my prototype implementation and providing encouragement that kept me going through the hard times of the graduate student years. I am much indebted to Dr. David Bryan Carpenter for countless productive discussions on various aspects of my research. Discussions with him helped me to grow professionally. I am also thankful to Dr. Marlon Pierce for his continuous help, support and advice. I would like to thank Dr. Shrideep Pallickara for his help and advice on the topics of security and messaging.

Many others at Bloomington provided much advice and help to my research. In particular, George Fletcher was deeply involved in many aspects of this research. He provided countless hours of discussion and counsel during my graduate student years. I would like to thank Harshawardhan Gadgil for his early comment on this research. I am grateful to others, including Beytullah Yildiz and Kwangmin Choi, who filled my life in Bloomington with the quality times.

I would like to thank staffs of Computer Science department, particularly Sherry Kay who helped me with all the paper work and Lucy Battersby who provided me all necessary information I need to know.

Last but not the least, I am especially grateful to my family for their support throughout my long and winding road of graduate studies. My parents provided invaluable support throughout my graduate student years and they encouraged me every step of the way. This would not have been possible without their love and support. My sisters, brother-in-law, and nieces with their love and support made this work worthwhile.

Abstract

The conventional Web Service communication framework does not adequately meet the needs of mobile computing. It is physically constrained and requires an optimized messaging scheme to prevent performance degradations in not only mobile computing, but also conventional computing that is interacting with mobile applications. With our novel architecture called the Handheld Flexible Representation (or HHFR), mobile applications 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. Despite its important role in distributed computing, mobile computing hasn't reached its full potential because of the limited availability of high speed wireless connections, such as third generation cellular technology (3G) or Broadband Wireless access. In this dissertation, we show that the messaging scheme of our new architecture significantly speeds up the messaging performance in comparison to a conventional SOAP-based Web Service messaging scheme when the request and response messages are in the same syntax, i.e. the same service is used continuously.

Contents

List of Figures

List of Figures

Chapter 1.

Introduction and Motivation

This dissertation presents an architecture for optimizing Web Service performance in mobile computing. Our thesis is that the conventional Web Service communication framework does not adequately meet the needs of mobile computing. It is physically constrained and requires an optimized messaging scheme to prevent performance degradations in not only mobile computing, but also conventional computing that is interacting with mobile applications. We propose a novel architecture called the Handheld Flexible Representation (or HHFR), with which mobile applications can negotiate characteristics for message stream and representation, and exchange messages in an optimized streaming fashion. By distinguishing between message semantics and syntax with a high-performance channel, the architecture provides an overall system framework for Web Services applications in mobile computing environment. Despite its important role in distributed computing, mobile computing hasn't reached its full

potential because of the limited availability of high speed wireless connections, such as third generation cellular technology (3G) or Broadband Wireless access. In this dissertation, we show that the messaging scheme of our new architecture significantly speeds up the messaging performance in comparison to a conventional SOAP-based Web Service messaging scheme when the request and response messages are in the same syntax, i.e. the same service is used continuously. We expect our research to grow in significance as the mobile infrastructure improves.

1 Introduction: Web Services technology and Mobile Computing

Web Services technology profoundly affected distributed computing after it first emerged a few years ago. Like its predecessors, such as the Common Request Broker Architecture (CORBA) [100], Remote Method Invocation (RMI) [101] and Distributed Component Object Model (DCOM) [102], Web Services' primary goal is to inter-relate distributed functionalities. But unlike its predecessors, it achieves its goal in an elegant and technology-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 platforms, may be running in different operating systems, or may be written in different programming languages, can communicate through web Service interfaces.

The interoperability of Web Services mainly comes from its Extensible Markup Language (XML) based open standards. The Simple Object Access Protocol (SOAP) [84] is defined in XML. Since it is text-based and self-describing, the protocol can convey

information between services in heterogeneous computing environments without worrying about conversion problems. Naturally, there are many other Web Service specifications[1]. Two of them, which are based on XML, are Web Service Description Language (WSDL) [85] and Universal Description, Discovery and Integration (UDDI) [103]. WSDL defines a standard method of describing a Web Service and its capability, and UDDI defines XML-based-rules for publishing Web Service information. Because of its strong interoperability across diverse services in a distributed environment, Web Service-based Service Oriented Architecture (SOA) has become the backbone of Grid computing. The Open Grid Services Architecture (OGSA) [29] defines a standard Web Service environment for Grid computing.

Just as Web Services technology has become an industry standard way of connecting remote and heterogeneous resources, mobile devices have become a vital part of people's everyday life. People use mobile devices anytime and anywhere, (e.g. cellular phones, PDA devices with either a cellular or a wireless local area network (WLAN) connection based on the IEEE 802.11 specifications [104], and hand held game consoles). In this dissertation, we define mobile devices to be those that are not only small size computing devices, but are also equipped with a wireless connection so that they can participate in some type of distributed computing. The number of cellular service subscribers has increased rapidly in the last 5 years[2]. As cellular phones become important devices, their usage is not limited to voice communication. People can also check Email and access the Internet with a cellular phone. In fact, cellular phones are so useful that fewer people are

[1] We will use both specification and standard interchangeably throughout the dissertation. To be precise, a standard is a specification that has undergone a formal or de facto evaluation process.

[2] Cellular phone services are also gradually replacing landline phone services. According to a study which was conducted by the U.S. Bureau of the Census, 7% of all American households only have cellular phones and that rate rises to 20% for persons age 15 to 24 [107].

wearing a wrist watch these days because they check the time with their cellular phone [138]. New models of cellular phone devices are capable of playing digital audio and video files, and they are also equipped with adequate memory space (e.g., 1GB secure digital memory) to store hundreds of music or video files. The features of the PlayStation Portable (PSP) [108] include ability to play movies[3], and/or music, and it is not limited to gaming. PSP also supports IEEE802.11b [105] and IEEE802.11g [106] connections which allow the user to play a game with other players. Because of these improvements, Mark Weiser's Ubiquitous computing will become viable with the proliferation of mobile devices [50].

Recently, the capability of mobile devices and wireless connection technology has increased dramatically. The performance of the mobile device is significantly enhanced by faster processors, larger installed memory, and enhanced user display. Meanwhile, the connection to a network has become easier through a widely available packet-switched 2.5G or 2.75G network as well as through third generation networks[4], which are in an early developmental stage in US.

2 Motivation: Problems in integrating Web Services with Mobile Computing

Web Services technology recognizes mobile computing as an area to which it should expand. Through integration, Web Services enables pervasive accessibility by allowing for user mobility as it overcomes the physical location constraints of conventional

[3] Universal Media Disc (UMD) is used for playing movies, music, and games on PSP.
[4] There is another breakthrough in the metropolitan area network (MAN) technology, IEEE 802.16. Though we limit our presentation to the cellular technology, since it is the most dominant and popular wireless technology with billions of subscriber. Also the IEEE 802.16, including WiBro (Wireless Broadband), which is deployed in South Korea, is at too early a stage to discuss its popularity.

computing. However, 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.

However, despite the fact that the condition of mobile computing has so much in improved recent years, applying current Web Service communication models to mobile computing may result in unacceptable performance overheads. This potential problem comes from two factors. 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 compared to other distributed computing approaches such as HTTP (Representational State Transfer: REST [109]), RMI or DCOM. Second, the performance and quality gap between wireless and wired communication will not close quickly.

As discussed, there have recently been radical improvements in the performance of wireless cellular connections. However, there are important obstacles which prevent the performance of wireless connections to match that of wired systems. First, 3G technology, which provides a maximum bandwidth 300~500kbps for downloading and 56~90kbps for uploading, can not match the bandwidth capabilities of a wired connection, which provide 10Mbps~1Gbps for both downloading and uploading. Second,

3G has a limited deployment and currently does not reach most of the US non-Metropolitan areas [5] . Third, wireless connections using radio waves face more degradation factors like buildings and landmarks than wired connections. Such degradation results: a poor quality of service for wireless connections. Ultimately, both of these factors are decided by battery-life and the rate of power consumption: mobile devices depend on a battery to maintain operations. Since a faster processor and a faster connection usually require more power consumption than slower devices, they will shorten the device's use time, which will lead not only to usage difficulties but also consumer resistance. This slows the deployment of 3G technology.

The problems created by encoding/decoding and slow wireless connections are the following: First, the message size will increase when data is converted into a textual format, which contains not only data, but also many descriptive tags. The size increase can be as high as an order of magnitudes, if the document structure is especially redundant (e.g., in the case of arrays). Even in a conventional computing environment, it is always good to have a small message. But in mobile computing, it is absolutely required because of the narrow bandwidth connection.

Secondly, encoding data into a SOAP message requires a text-conversion, where the *in-memory* representation is converted into a textual format. The decoding process does the reverse work. If the data is non-textual, such as a floating point number, the conversion is very expensive in terms of performance overhead, which is especially significant for relatively low-powered mobile devices.

[5] Even in fully serviced areas, such as South Korea, limited numbers of cellular users are using 3G, because of its high cost to use.

Finally, even though the Web Service specification is not limited to any specific transport protocol in its architecture, the Hyper Text Transfer Protocol (HTTP) protocol is currently the most popular transport protocol among mobile Web Service implementations. However, particularly in mobile computing it does not perform adequately to be used in some application domains, which need high performance communications. Because it is based on a request/response paradigm, sending a request in HTTP is tied to receiving a response. When the communication channel has a high latency, this request/response paradigm produces performance degradation.

3 Thesis Summary

This dissertation proposes a novel architecture called the *Handheld Flexible Representation* (HHFR) [6] for efficient and optimized Web Service messaging performance in mobile computing. HHFR introduces a negotiation stage in order to set up a high performance communication channel. This distinguishes between message semantics and syntax to allow for a flexible representation of a message. In HHFR, a 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 or 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

[6] It would be abbreviated as HFR, but conventionally, Handheld is abbreviated as HH in mobile computing.

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, 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, and it can be implemented through various transport protocols such as the connection-oriented Transmission Control Protocol (TCP), the connectionless User Datagram Protocol (UDP), or HTTP with a persistent connection[7].

Secondly, HHFR uses a Data Format Description Language (DFDL)[8] [58]-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.

Thirdly, in the HHFR, a Context-store 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

[7] HTTP 1.1 specification defines the mechanism to send one or more (usually, between two and five) requests.
[8] It is a XML Schema based descriptive language proposed by the Global Grid Forum (GGF). In Chapter 2, we will discuss it in detail.

conventional SOAP message representation on demand. For the context-store we use the Fault Tolerant High Performance Information Service (FTHPIS) [110], which is WS-Context [95] compliant and was developed by the Community Grid Laboratory at Indiana University. Our architecture represents a novel use of this technology and presents interesting new performance measurements on FTHPIS.

In order to demonstrate the potential of HHFR, we focus on an application domain where two Web Service participant nodes exchange a series of messages. In this dissertation, we define this message series as a *stream*. For applications using a specific service, messages in the stream have the same structure and the same data type, if the client application links to the same service repeatedly. Applying our new approach to the stream yields many advantages in communication performance, such as the ability to use a flexible representation and to store meta-data in the Context-store. The overhead in setting up our approach is only incurred once per stream and amortized over the many messages potentially contained in a stream.

In this dissertation, we describe the architecture design and implementation of the HHFR-based Web Service communication platform which is compatible with conventional SOAP Web Services. The prototype of the architecture, which is implemented in Java, is presented in detail and is evaluated through two benchmark applications. We present performance results demonstrating that the HHFR achieves efficient Web Service communication and outperforms the conventional SOAP communication particularly with applications that exchange messages in a streaming fashion. Our presentation is particularly focused on applications in mobile computing, but the approach may be more general.

4 Thesis Contributions

This dissertation investigates significant research problems that emerge with the increasing need for a system level framework integrating Web Services technology and mobile computing with a broad, clean architecture rather than using various ad-hoc approaches. Those research problems include, but are not limited to:

- The verbosity of the XML-based SOAP message format causes performance degradation in Web Service message exchanges in mobile computing. This performance issue involves:
    - The size of messages which increase after a SOAP serialization.
    - Encoding and decoding includes the conversion between text to/from non-text format conversion
- A conventional Web Services communication channel does not adequately meet the need of mobile computing environments for the following reasons:
    - A high latency wireless connection slows down overall message exchanges.
    - The conventional Web Service transport, HTTP, ties sending a request message with receiving a response message.

These problems have been investigated by many researchers. As a result, there have been many systems proposed to solve them. However, to the best of our knowledge, none of these proposals or implementations tried to provide a system level solution rather than an ad-hoc solution for part of the problem.

The goal of this dissertation is to design a system-level architecture that can:

- Distinguish the semantics from the representation of Web Service message content.

- Describe a representation or syntax of the message in a XML-based description language.
- Support a high-performance communication channel.
- Provide a mechanism to negotiate the characteristics of a stream, which is a series of message exchanges.
- Provide an interface to an Information Service to store the meta-data of stream.

In this dissertation, we present our investigations into the problems of Web Services and our design and implementation of the Handheld Flexible Representation (HHFR), which meet goals stated above. Therefore this dissertation makes the following contributions in the area of mobile computing:

The Handheld Flexible Representation Architecture and the prototype implementation: the main contribution and focus of our dissertation is the HHFR architecture, which offers a system level comprehensive communication framework to Web Services applications in mobile computing environment. The architecture and its prototype implementation include the following contributions:

- Proposing and implementing an asynchronous messaging system through a high-performance communication channel to reduce the performance overhead in mobile communication. Because of high latency, the interval between the request and the response through wireless connections is essentially unnecessary overhead that wastes communication time. Asynchronous messaging combined with a high-performance channel reduce the performance gap by filling up “a logical pipeline” [3]

between two endpoints. We show that our approach can utilize the connection better than conventional methods. Once two endpoints agree on using second channel to exchange messages, they send and receive messages in streaming fashion.

- Defining the Simple_DFDL. The flexible representation can be achieved by separating the semantics of a message and its representation. We present a detailed discussion of the Simple_DFDL, which describes the data format for the HHFR prototype. We also present the structure of the Simple_DFDL, which is similar to that of DFDL: i.e., the Simple_DFDL language describes the data format, the Schema Processor builds the HHFR Data Model, and the stub converts data from and to a preferred representation.
- Proposing the Context-store to store the meta-data of a stream and implementing interfaces to the FTHPIS. We present our approach which saves the meta-data of a stream to reduce the size of the messages in the stream, and implements interfaces to the Information Service. Our empirical experimental results show significant performance savings by using this approach.

Detailed Performance Evaluations: we present a detailed performance evaluation of the HHFR through two benchmark applications. Each application covers different data domains: strings and floating point numbers. The results show that applications in the HHFR outperform the conventional SOAP-based Web Service communication through HTTP. We present the Context-store related performance results, which show bandwidth savings by storing meta-data to the Context-store and exchanging slimmed down messages. In addition, the scalability analysis shows that a server can support several

thousands of simultaneous streams in our approach to use Information Service as a Context-store.

List of recommendations for an ultimate solution: an additional contribution of this dissertation is our recommendation for an ultimate solution. We present a list of our recommendations uncovered through our studies and investigations.

The HHFR architecture can be used in many areas. For example, a Geographic Information System (GIS) client on a mobile device can benefit from our novel architecture. Assume that the client receives periodical GIS information, such as temperature and pressure, and that the gathered periodic information as a collection will generate a graph. Since conventional Web Service messaging supports only SOAP formats and a high latency and narrow bandwidth wireless connection is not sufficient for big data set, the client should have a framework, by which it can choose among various data format such as ASCII[9], Geography Markup Language (GML) [111], and binary and thereby utilize the wireless connection better.

5 Thesis Roadmap

The rest of this dissertation is organized as follows.

In Chapter 2, we present background on the HHFR architecture, presenting previous, and on-going efforts which address the performance limitations of SOAP based Web Services technology. In Chapter 3, we give an overview of mobile computing and the

[9] American Standard Code for Information Interchange

current status of mobile Web Services. This chapter explores the specific mobile computing problems, which are added to the existing problems of conventional distributed computing, and it also details the current Web Services strategies for mobile computing.

Chapter 4 describes our HHFR architecture in detail. The architecture proposed in Chapter 4 is an idealized solution and we discuss a prototype where practical considerations require some compromises in Chapter 5. The prototype is then subjected to several tests, which are analyzed to help clarify the key features of the full HHFR architecture.

Chapter 6 presents a detailed performance study of the two benchmark applications we used to demonstrate the performance saving aspects of the HHFR architecture design. In Chapter 7, we discuss the Context-store implementation of the architecture design and its performance evaluations, which are specifically design to examine the Context-store's validity. Finally, in Chapter 8 and Chapter 9, we outline several areas for future work and present the conclusions of our research.

Chapter 2.

Background and Related Work

In this chapter, we present background on the HHFR architecture, presenting previous, and on-going efforts, which address the performance limitations of SOAP based Web Services technology. Some of these efforts do not specifically target mobile computing environments, but they share similar research issues with mobile computing. As explained in the previous chapter, Web Services in a mobile computing environment face problems of performance-degradation similar to the conventional distributed computing environment. So a primary research issue in the mobile Web Service area is the attempt to provide an efficient message processing scheme while preserving XML's interoperability.

Related work on solving this problem can be categorized as either individual message optimization or as message stream optimization. An individual message optimization approach produces a simplified, efficient, and self-contained message, which is a

different format (or representation) to XML. The messages in the different representation can be converted to and from the XML format, which is called roundtripping. For example, Fast Infoset (FI) from Sun Microsystems [61] [62] and XBIS [13], [72] by Dennis Sosnoski fall into this category.

On the other hand, the message stream approach optimizes a whole sequence of related messages, which we define as *a stream*[10]. This approach includes a certain form of negotiation to define stream characteristics, and optimized message representation in the stream. Examples of this category include Fast schema from Sun Microsystems [60] [141] and our own HHFR architecture [44].

Within this chapter, we will describe other related works as well whose goal is attaching binary data to SOAP message. Examples of this include are Message Transmission Optimization Mechanism (MTOM) [86], XML-binary Optimized Packaging (XOP) [87] and Direct Internet Message Encapsulation (DIME) [63].

We classify the 6 approaches described in Table 2.1.

Table 2.1 Categorized XML Optimization Efforts

Individual Message Approach (Self-Contained Message)	Stream of Messages Approach (Non Self-Contained Message)
Fast Infoset of Sun Microsystems	ExtremeFastWS
XML Schema-Based Compression (XSBC)	Fast Web Service of Sun Microsystems
XML Infoset Encoding (XBIS)	Handheld Flexible Representation

[10] From the application level point of view, we use the term *a session* for consecutive messages between a service and a client.

1 Lineage of XML Binary Representation

The self-contained markup syntax of XML makes SOAP messages self-describing. Yet its syntax causes performance limitations (or degradation) in some computing domains. In this section, we describe previous work on Binary Representations for XML and SOAP.

1.1 XML As A Data Interchange Format

XML has become a popular data format for interchanging information because it is self-describing and easy to implement. Since each data field [11] of an interchanged message[12] is individually described by its markup, they are individually understandable and can be modified independently of other data field. An application uses an *engine*, which is called *parser* in many cases, to process XML data. The parser can be a separate module and maintained independently. Consequently, applications whose data interchange format is XML can be loosely coupled to each other regardless of what platform they are running and in what programming language they are built.

Yet there are problems with XML in mobile computing. One reason is that the data size is larger after a transformation from binary format to text-based XML. Also XML document requires non-trivial amounts of processing time to parse, transform, and validate text-based markup syntax. Despite the performance degradation that exists when using XML, application developers and users in a conventional distributed computing domain are not very concerned, because their machines are powerful and have

11 i.e. Information Items defined by XML Information Set (Infoset) specification [82]
12 It is either a single XML document or a collection of multiple XML document. For example, a SOAP message instance consists of multiple XML documents. Header parts may contain several XML, but a payload should have one XML document.

connections sufficiently fast. But mobile computing and real-time computing are not like conventional distributed computing in that regard. They are sensitive to the performance overhead. The processing overhead of XML and SOAP is amplified in the relatively low-powered and low-bandwidth mobile computing environment. In addition, real-time computing is much more sensitive to transmission latency due to the larger data and increased processing required.

As a result, there has been much discussion of, and effort put into, finding more efficient, but still XML-conformant representations.

1.2 XML Binary Characterization Working Group

Binary XML is defined by the XML community as "A format that does not conform to the XML specification yet maintains a well-defined, useful relationship with XML." [90] In other words, we can also define it as any format that has a filter[13] and its inverse to conventional XML. Binary XML is quite common because there are many areas that need the binary XML specification when the verbosity of XML causes performance degradation.

The report of the W3C Workshop [89] on Binary Interchange of XML Information Item Sets (Infoset)[14] [82] documents the increasing demands for binary XML. The report incorporates both the conclusions of the workshop, which met in September 2003, and several dozen position papers that were presented at the workshop [60] [64] [1]. The purpose of the workshop was to study methods of compressing XML documents and

[13] A filter converts formats between the conventional XML and Binary XML. It may or may not need to access information outside a format (i.e. a message)
[14] We have a detailed explanation and discussion in Chapter 4.

transmitting pre-parsed and schema specific objects. The requirements of binary XML Infoset as identified at the workshop are the following:

1) It must maintain universal interoperability.
2) It should provide a generalized solution that is not limited to a specific application domain.
3) Processing time including data binding time should be reduced from original XML documents.
4) There should be a negotiation – if it fails and the receiver cannot understand the binary, it should fall back to the XML/SOAP text format.

The discussion led W3C to form the XML Binary Characterization Working Group (XBC WG) [112] for further research. In March 2005, a series of XBC WG notes were released, providing a formal definition of binary XML [57], its use cases [91], measurement methodologies [93], and properties [92]. The XML Binary Characterization note specifies the W3C recommended property requirements that must by supported by binary XML. The properties are:

- Transport independence: the binary XML format should be independent from the transport service in that it should be error-free and deliver the messages in order, regardless of the message length.
- Human Language Neutral: the format also should not impose more restrictions on one human language than another.
- Royalty Free: the format should be free to create and use.

- Platform Neutrality: platform neutrality doesn't require the format to perform identically on all computer platforms and architectures; rather it requires that binary XML not be defined around any platform specific parameters.
- Content Type Management[15]: Content Type Management means that the format should define its own media types, encodings, or both. The XML stack consists of validation, transformation, querying, APIs, canonicalization, signatures, encryption, and rendering.
- Integratable into the XML Stack: the format should easily find its place within the existing body of XML-related technologies.

In section 5 of the XBC Characterization note, three additional properties are mentioned that must be supported: Direct Readable and Writable[16], Compactness, and Processing Efficiency. A scheme, which otherwise looks to be well composed, could violate one of the required properties, and so cannot be considered as a W3C endorsed binary XML. A popular example is GNU ZIP (GZIP) [76]. It preserves the byte-to-byte integrity of XML and provides a good compactness. Yet, its overall performance is poor when it is applied to a short document with non-redundant vocabulary since its processing time includes compression and decompression time as well as serialization and deserialization. Because of this long processing time, which violates the processing efficiency requirement, GZIP is not considered to be a W3C endorsed binary XML.

The imperative to speed up Web Services for small devices are well summarized in W3C's "XML Binary Characterization Use Cases [91]" document. This document

[15] The last two property requirements are necessary to ease integration between existing XML and web technologies and the new binary XML format.
[16] The format should be serialized in one logical step.

describes situations where XML possesses potential overheads and defines these situations where devices have limited memory, limited processing power, and a limited battery life. These are critical factors for Web Services for small devices, and they are even more critical when they are connected to narrow bandwidth and high latency networks. As a result, this computing domain would have considerable benefits from a small packet and streaming processing scheme.

2 XML Alternatives: Self Contained or Not

Some systems or proposals do not satisfy all the requirements of the W3C XML Binary Characterization, but they can still provide good insights into the performance problem of XML and good designs to tackle it. To that end, we will discuss proposals for optimized XML messaging including those inconsistent with the W3C requirement. As discussed, we categorize the research into the optimization of XML message processing into two groups: the individual message approaches and the stream of messages approaches.

2.1 Self Contained Approaches

Most proposals that follow the XML Binary Characterization of the W3C have a goal of producing a self-contained alternative to an individual XML message, optimized for faster processing and smaller packet size.

The basic optimization strategy is to replace a redundant vocabulary with indexes. Tables and indexing are key elements of the mechanism. This approach is similar to

Table 2.2 Encoding/Decoding Steps

Encoder	Decoder
1. Add a newly encountered string to table.	1. Add a newly encountered string to table.
2. Replace the string with index on the next occurrence.	2. On encountering an index, look up the corresponding string and substitute it.
3. Repeat 2 for all string occurrences until the end of the document	3. Repeat 2 for all index occurrences until the end of the document

document compression, but it focuses more on processing speed as opposed to reducing document size, which is the main concern of most compression schemes.

Message processing begins when an encoder transforms an XML document into an optimized format, i.e., a message in an optimized representation. In processing, an encoder traverses through an XML document. When it encounters a new string, the encoder adds the string to a table and gives it an index. If the string recurs, the encoder replaces the string with its index. The encoder continues replacing strings with indexes until it reaches the end of the document. Later a decoder transforms the optimized message back into an XML document. The decoder reverses encoding procedure, also using tables and indexes. Decoder adds a string to a table when it encounters the new one[17]. When it encounters an index, the decoder replaces the index with a corresponding string from the table. Positive integer values are used for indexing because this is a straightforward process and the size of positive integer value is small. Table 2.2 shows transforming steps by encoder and decoder. In addition to replacing strings, the

[17] If the new string is accompanied with an index, the index puts into the table together with the string. See our discussion about Fast Infoset.

compression may be applied to character information items to reduce their size, where an XML specific compression tool, such as XMill [31], can be used.

By preserving XML information items and properties, the individual message optimizing scheme produces a message format that is still self-contained yet can be converted back into the XML document format. This is the primary advantage of this approach; it produces an efficient and lightweight representation. But the conversion of the alternative representation back to the XML document has one significant limitation, the conversion of canonicalized information items. For example, there is no way to distinguish two different forms of empty elements in the inversion process, e.g. `<e1/>` and `<e1></e1>`. It is because both of those empty elements are permitted as *well-formed* XML elements by the XML specification and yet each of the two elements has the same canonicalized form. So any XML application does not distinguish between those two different representations. So an empty element could easily be converted into representation, different from the original. Again, the normalizing white spaces of start and end tags, the relative and lexicographic order of namespace and attribute axes, and any replaced entity references may not be converted in the original element. The XML Infoset data model is not bound to the specific representation. So the XML Infoset based XML application extracts XML Infoset information from the XML document that is well formed by XML specification, but it would keep them in the preferred canonicalization format, which may differ from the original. The reverse-conversion[18] could then produce a different representation from the original.

[18] The conversion back to the XML document format

The encoding and decoding schemes do not need to address the reliability issue. Since an individually optimized message is self-contained, the approach is independent of the transport protocol, and it will not be affected by the characteristics of that protocol. Therefore, each message delivery depends on the transport protocol's transmission control.

Fast Infoset

We will give a practical example of individual message optimization using the Fast Infoset specification of Sun Microsystems. Fast Infoset (FI) aims to provide an XML alternative in order to provide faster and more efficient Web Services in restricted computing environments. The specification was proposed at the W3C Workshop on Binary Interchange of XML information Item by Sun Microsystems and it was researched at the same company. Later they proposed it as a specification under Abstract Syntax Notation One (ASN.1) [78]. Fast Infoset uses an existing standard to achieve interoperability. The telecommunication industry standard, ASN.1 is a flexible notation that describes a data structure and type of message exchange by providing a set of formal and platform independent rules for describing data. The mapping XML Schema to ASN.1 is defined in the X.694 standard [99].

In the Fast Infoset specification, the serialized XML document is called the Fast Infoset Document. It contains information items and their properties as well as the hierarchical structure of the XML document. Examples of supported properties of information items are children, notations, character encoding schemes, versions, namespace names, localnames, prefixes, namespace attributes, and the normalized value of attribute information items. Fast Infoset introduces many features that improve

message processing performance; for instance, the primary feature is to replace redundant vocabularies. It uses predefined tables and references to an external vocabulary called an Initial Vocabulary. This idea exploits common entries in a SOAP envelope, many of which are already known before encoding in many cases. Using an initial vocabulary table allows the transformation process to avoid processing known entries dynamically. This means that processing time is significantly reduced.

FI has several features that contribute to faster processing and smaller message size when it produces a Fast Infoset document. Some of them are:

- Length-prefixing of content
- No end tag
- No escaping of character data
- Embedding of binary content

FI prefixes the context length for a decoder so that it can allocate resources accurately and possibly reject a content that exceeds the size limit. In addition to prefixed content length, FI removes the end tags from the document. The designers of FI claims that escape character checking is time consuming and they remove the step. FI allows binary data embedding in the document so that the conversions from and to base64 can be avoided.

Figure 2.1 shows an example document and its transformation according to the Fast Infoset specification. Figure 2.1(a) is the XML document. Figure 2.1(b) shows the corresponding representation with indexed strings and qualified names in a symbolic form. Figure 2.1(c) and 2.1(d) give a qualified name table and a generic string table, respectively.

A curly bracket is used for an identifying string, i.e. a new string and a square bracket is used for a replaced string. Thus, "**[1]**<>**{1}**two" of 2.1(b) means that the qualified name "tag" is indexed as 1 in the qualified name table and an index of 1 is used for generic string "two". The Fast Infoset document is encoded using ASN.1.

```
<root>
<tag>one</tag>
<tag>two</tag>
<anotherTag>one</anotherTag>
</root>
```

(a)

```
{0}<root>
{1}<tag>{0}one
[1]<>{1}two
{2}<anotherTag>[0]
```

(b)

	Local Name (Qualified Name)
0	root
1	tag
2	anotherTag

(c)

	Generic String
0	One
1	Two

(d)

Figure 2.1. A Simple Example of Fast Infoset Indexing

XBIS and Cross Format Schema Protocol XFSP

Developed by Dennis Sosnoski, XBIS also uses a generic scheme for replacing repetitive words (a define-and-replace scheme). XBIS is similar to XMill, which is a XML specific compressing tool and will be described in Section 3 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.

Cross Format Schema Protocol (XFSP) [1] 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 [113] model to parse the schema. Combined with XML Schema-based Compression (XSBC) [12], XFSP provides binary serialization and a parsing framework. Eric Serin and Don Brutzman of the Naval Postgraduate School designed and implemented XFSP, and are currently researching streaming X3D [23] documents in the XFSP framework. X3D is a XML-based open standard for three dimensional data.

2.2 Non Self-Contained Approaches

Another approach to improving Web Service performance uses a stream of messages. In this approach, messages are not self-described and are in an alternate representation.

The steps of this type of message processing are simpler than the steps in the individual message optimization approach, which needs to replace repetitive words (e.g. in this type of message processing does not require a data conversion to and from text format). But in order for it to work a customized encoder and decoder are required because the application processes a message that is not a general self-contained XML message, but rather it is an alternative format message. The encoder and the decoder must write and read the schema specific data equivalent to XML information items as it is defined in the XML Schema document. The application is not able to handle a message unless it follows the type and structure in the schema.

Generation of the encoder and decoder could either happen dynamically or statically:

- Since an XML Schema document describes a structure and types, the application builds an encoder and decoder by parsing an XML Schema of the message. To use a different format for exchanging messages, the application simply parses a schema document in the new message format and generates the new encoder and decoder dynamically.
- When the schema of messages is fixed for the lifetime of the given application, the encoder and decoder in the application can be generated statically when it is compiled and deployed.

The stream of messages approach has obvious advantages and disadvantages. One of the advantages is the performance gained by avoiding a text conversion. Since the approach does not require text-based XML data format, the application is able to ship binary data directly without conversion. Also, the size of messages can be reduced by not requiring a descriptive markup for XML information items.

One of the disadvantages comes from the same reason: the messages are not self contained. The strength of XML messaging lies in the descriptiveness of the XML documents. In the processing of messages in the series of messages[19], processing one message does not affect the processing of other messages. Because it separates data (i.e., semantics of the message) from its XML syntax, the representation of one message must be similar to other messages in the stream. So this approach does not allow any changes in the structure and types of messages in the stream.

Despite the disadvantage presented here, some application domains needing high-performance communication should be able to sacrifice the self-describing characteristic of an XML document.

[19] We define the series of messages as a stream in the previous chapter.

Extreme! Lab's Multiprotocol Approach Recommendation (ExtremeFastWS)

Extreme! Lab at Indiana University researches the limits of SOAP performance for scientific computing where large data sets, such as arrays, are common [35] [36]. Their experiments show major improvements by using a) a schema-specific parser mechanism for arrays, b) a persistent connection, and c) a streaming of messages to prevent full-serializing objects from determining length. The most serious overhead when exchanging large scientific data sets is the conversion from in-memory floating point numbers to a textual format. This research suggests that it is more beneficial for scientific applications to use multiple communication protocols including a binary representation and fast protocols other than SOAP. The problems faced here with conventional Web Services are similar to the ones in mobile computing. Both need to overcome the performance limitations of SOAP.

Fast Web Services from Sun Microsystems

Fast Web Services (FWS) of Sun Microsystems is intended to provide a fast and efficient Web Services specification that is interoperable with existing technologies and minimizes the impact on application developers. It has been developed by the same group of people at Sun as Fast Infoset. The FWS implementation encodes XML information item data using ASN.1 encoding rules, like X.694. The difference between Fast Infoset and Fast Web Services is that Fast Infoset uses a self-contained message while Fast Web Services uses a schema specific binary data format.

Fast Web Services provides fast processing and low bandwidth usage by adopting the ASN.1 standard to XML schema. Additionally, since its data transformation is transparent to the application developer, the Web Service Definition Language (WSDL) and higher layer are unchanged in developing applications. But since the approach needs a tailored encoder and decoder – a stub and a skeleton for the schema specific data – it has a limited expandability.

Example: Data Format Description Language

The Data Format Description Language (DFDL) [58] from the Global Grid Forum (GGF) falls into the stream of messages category in a broad sense. DFDL defines both the structure and type of binary information using an XML Schema style language.

DFDL is a descriptive language that is proposed to describe a binary format file or stream for Grid computing. Like Extensible Scientific Interchange Language (XSIL) [59], it is XML-based and comes with an extensible Java data model. DFDL defines the structure and type of data. For example, it defines whether the data is big-endian or little-endian. It also defines complex data formats such as arrays. DFDL is designed to be processed through a DFDL parser.

The message format description in our HHFR architecture is based on DFDL. In this architecture, we define a simple XML-schema-based descriptive language and develop a language parser using the XML Pull Parser (XPP) [7]. Our prototype implementation will show the design advantages of our architecture. We will discuss DFDL in more detail in Chapter 5.

2.3 Processing Headers

There are already over a dozen specifications that define SOAP header elements providing message routing, transactional semantics, message security, etc. SOAP 1.2 requires all top-level information item in the header to be namespace qualified. So the SOAP header becomes a consistent place to put messaging metadata that is guaranteed not to conflict with other metadata in the same SOAP message.

Despite the fact that there is no specific guideline in the SOAP specification about what information should be placed in the SOAP headers, the headers usually contain information that assists Web Services to communicate with each other in a secure and robust way. Examples include information for message exchange, security information, routing instructions, etc.

The handlers in the Axis [27] architecture which process the SOAP messages demonstrate how the system processes the headers. The handler is a module, which acts as a message interceptor, as it processes a part of a SOAP message. A target Web Service of the given SOAP message is considered to be one of the handlers, which is located at the end of the handler chain. A simple example of processing a header with handlers involves a WS-Addressing header, which could be encrypted. An addressing handler examines an address header in the SOAP message and then dispatches an appropriate service. If the header is encrypted, a security handler in the chain should be activated.

Compared to the SOAP body, which is consumed by the services, the header itself is processed by the system, i.e. all handlers other than *the Target Service* in the Axis. This different processing level makes it difficult to apply the same alternative representation strategy to the headers while keeping them compatible to general Web Service

specifications. The conventional Web Service framework, e.g., Axis or the .NET framework of Microsoft [25], cannot understand the new alternative representation which differs from the SOAP format. For example, a ReliableMessaging handler in the system, which is defined by the WS-Reliable Message specification [96], cannot process a header in the alternative representation, e.g. the representation defined by the WS-Reliablility [114], because it does not recognize it as a SOAP header.

The WS-Security [97] case is more complicated. The WS-Security specification allows Web Service participants to encrypt individual elements rather than the whole SOAP message, and this makes the specification more flexible and efficient. At the same time, encrypting individual elements makes the Security handler hard to implement in an alternative representation other than SOAP because the representation doesn't have the element syntax of XML specification.

To process the message in the alternative representation, the system, which is a Web Service container or a SOAP engine that understands it, would convert back to a SOAP message format and process the headers in a conventional method. The drawback of this method is the conversion overhead. Since the goal of the alternative representation includes high-performance processing of the message, converting back to the SOAP message format creates a huge redundant overhead to message processing. One way around this redundancy is to place a handler that understands the new alternative representation in the early phase of the handler-chain like a transport handler in the AXIS architecture and processes the header.

The object model for Axis2 [28], called the Axis Object Model (AXIOM) has an interesting approach to this problem. Internally, Axis2 uses an XML Infoset based data

model – AXIOM. The AXIOM allows the Axis2 SOAP engine more freedom to process alternative representations flexibly since *In-Pipe* takes the incoming SOAP message and maps it into AXIOM object, which is XML Infoset based and doesn't stick to the XML syntax.

3 Compressing XML Documents

The XML document optimizations described in the previous section could be considered compression schemes for XML documents. However, they are not 'true compression schemes' because the optimizations reduce the processing overhead as well as reduce the size of the document.

A self-contained and human-readable XML document may often be huge because of its text-encoding and descriptive tags. Due to this fact, extensive research has been done on compressing XML documents. The Results of this research can be applied to Web Services to improve performance in this setting.

gzip [76] is a data compression program that is based on the DEFLATE [77] algorithm, which is a combination of LZ77 [55] and Huffman coding [55]. The gzip file format uses a variable length code table for encoding source symbols just like other DEFLATE algorithm-based data compression formats – e.g., PKZIP [16], PNG [75] and ZIP. The gzip is a widely-used generic text transformation standard used in much early research on the restricted environments. The adoption of gzip in Web Services to reduce the size of SOAP messages is straightforward, but it doesn't add processing time. This is true because a message must be compressed before sending it and decompressed after receiving it. Because the redundancy checking in the compression algorithm consumes

many processor cycles, it doesn't save time in most mobile computations. So while generic text-based compression works well where powerful machines communicate over a slow connection, like a modem connection, it doesn't give much advantage in mobile computing environment with limited processing power and memory space.

Compared with generic compression, which yields poor performance on small messages, XML-specific compression e.g., XMill [31] and Millau [32] may do much better. XMill is a XML specific compressor based on a grouping strategy. It rearranges document text and groups text items together based on their semantics to achieve a better compression ratio. Start tags are assigned to integer values and end tags are replaced by a '/'. Dictionary encoding is used to assign the integers. After the rearrangement, text items in a group will have many similarities. Later, a conventional compression algorithm, such as gzip, is applied to a specific memory window, which includes multiple text groups. This algorithm then exploits the similarities between text items in a group.

Even though the XML specific compression achieves a better ratio between original size and compressed size (the experiment in Ref [31] shows it performs twice or better) and reduces the document size, the additional layer required compressing and decompressing add a significant overhead to the overall processing. Since the compression method saves bandwidth but does not reduce processing time, it can not serve as a full solution to the performance bottleneck in mobile Web Services.

4 Binary Attachments

We will now describe efforts to attach binary data to Web Service messages. The primary motives of these efforts include data integrity and reducing processing

overheads. Yet it is also important to consider the idea behind efficient binary data processing.

4.1 Binary Data as a MIME Attachment: MTOM

The W3C XML Protocol Working Group has released the draft of Message Transmission Optimization Mechanism (MTOM) [86], XML-binary Optimized Packaging (XOP) [87], and Resource Representation SOAP Header Block (RRSHB) [88]. Together these specifications target two data domains – multimedia data and data that includes digital signatures. Increasingly, multimedia data is exchanged using SOAP. Audio, graphic, and video files already have standardized formats, like JPEG, GIF, and MP3. They may be very large files, but they are efficiently encoded with specific algorithms. Encoding these multimedia files in SOAP would consume many extra processor cycles, which would be intolerable in some application domains. So the attachment of binary data is an important issue if Web Services technology is to be used pervasively in the multimedia data domain.

XOP is an alternative serialization to the W3C recommended XML [81]. XOP is a MIME-based package, allows binary data to be included along with an XML document, and avoids text-conversion overhead, though it still preserves the XML markup structure. MTOM describes how XOP is layered onto the SOAP and the HTTP transport.

The last specification, RRSHB, describes the semantics and serialization of a SOAP header block for carrying resource representations in SOAP messages. The `representation` element is an information item that describes the type of Web Resource, for instance image files, by including the `resource` attribute in any URI. Its

```
--MIME_Boundary
Content-ID: <mymainpart@crf.canon.fr>
Content-Type: application/xop+xml;charset=UTF-8;type="application/soap+xml"
Content-Transfer-Encoding: binary

<soap:Envelope                          xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xmlmime="http://www.w3.org/2004/06/xmlmime"
xmlns:xop="http://www.w3.org/2004/08/xop/include">
        <soap:Header></soap:Header>
        <soap:Body><ns1:EchoTest xmlns:ns1="http://example.org/mtom/data">
                <ns1:Data>
                        <xop:Include href="cid:thismessage:/frog.jpg">
                        </xop:Include>
                </ns1:Data></ns1:EchoTest>
        </soap:Body>
</soap:Envelope>

--MIME_Boundary
Content-ID: <thismessage:/frog.jpg>
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
……
binary data
……

--MIME_Boundary--
```

(a) MTOM/XOP Message

```
--MIME_Boundary
Content-ID: <mymainpart@crf.canon.fr>
Content-Type: application/xop+xml;charset=UTF-8;type="application/soap+xml"
Content-Transfer-Encoding: binary

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
xmlns:xmlmime="http://www.w3.org/2004/06/xmlmime"
xmlns:xop="http://www.w3.org/2004/08/xop/include">
        <soap:Header></soap:Header>
        <soap:Body><ns1:EchoTest  xmlns:ns1="http://example.org/mtom/data">
                <ns1:Data>
                        ……
                        Base64 encoding of binary data
                        ……
                </ns1:Data></ns1:EchoTest>
        </soap:Body>
</soap:Envelope>
```

(b) Conventional SOAP Message

Figure 2.2. A Simple Example of XOP-SOAP Message

mandatory child element, `data`, can have any number of character information items. The resource may be cached in a remote location, and the specification describes how the SOAP message recipient accesses data that exists not in the SOAP message, but rather as a cached representation of external resources. Figure 2.2 shows a simple example of the MTOM/XOP message.

4.2 Wrapping Binary Data: DIME

Direct Internet Message Encapsulation (DIME) [63] is another approach to attaching binary data to a SOAP message. It has been developed and is supported by Microsoft. A DIME message header is pre-defined in binary (see Figure 2.3 for the fields in the header). A DIME parser needs to parse every message before receiving it and after sending it. It is a very different from attaching binary data to MTOM/XOP, which preserves SOAP encoding and appends MIME parts to the message for binary data. Conversely, DIME wraps a SOAP message with binary data by a predefined rule. An example of a DIME message from an article at Microsoft Developer Network Magazine is shown in Figure 2.4.

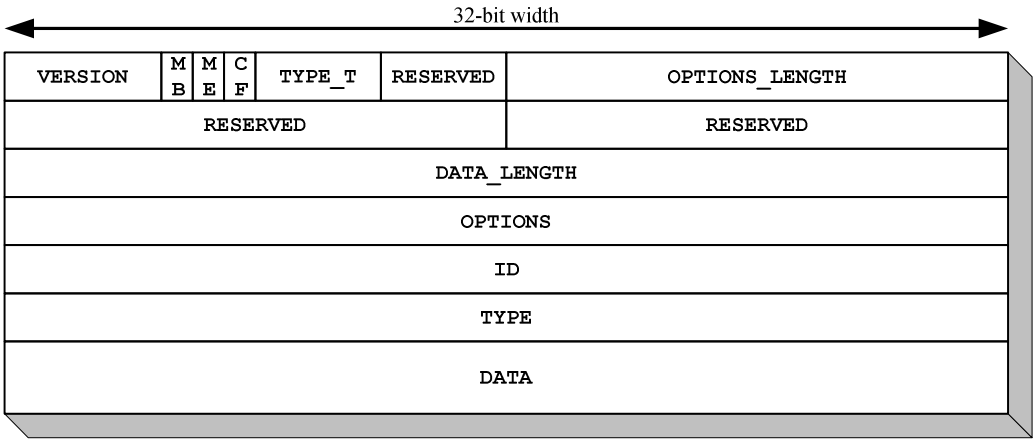

Figure 2.3. DIME Fields

```
00001 1 0 0 0010 0000000000000000000
0000000000000000 0000000000101000
00000000000000000000000110110101
http://schemas.xmlsoap.org/soap/envelope
<soap-env:Envelope
  xmlns:soap-env="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:msg="http://example.com/DimeExample/Messages/"
  xmlns:ref= "http://schemas.xmlsoap.org/ws/2002/04/reference/">
  <soap-env:Body>
    <msg:GetMediaFile>
      <msg:fileName>myMediaFile.mpg
      </msg:fileName>
      <msg:file ref:location=
          "uuid:F2DA3C9C-74D3-4A46-B925-B150D62D9483" />
    </msg:GetMediaFile>
  </soap-env:Body>
</soap-env:Envelope>
-----------------------------------------------------------------------
00001 0 0 1 0001 0000000000000000000
0000000000101001 0000000000001010
00000000000101011010101011100000
uuid:F2DA3C9C-74D3-4A46-B925-B150D62D9483
video/mpeg
<<First 1.42 MB of binary data for myMediaFile.mpg>>
-----------------------------------------------------------------------
00001 0 1 0 0000 0000000000000000000
0000000000000000 0000000000000000
00000000000010000110110001000000
<<Remaining 552 KB of binary data for myMediaFile.mpg>>
```

Figure 2.4. A Simple Example of DIME Message

4.3 Comparing binary XML with sending binary data over SOAP

Sending binary data with SOAP message, as described in the MTOM/XOP and the DIME approaches, allows sending binary data (so called *opaque data*) between Web Services without text-conversion. These methods can reduce much of the processing overhead that is caused by a direct text-encoding like base64. Especially, MTOM/XOP has become a popular option to send opaque data efficiently because it is simple to use and it is strongly supported by IBM [20], Microsoft [21] and BEA [22].

However if the desired data is not in a fixed format, the above approaches cannot be applied directly. For example, arrays in a binary format or any user defined structure need additional mechanisms so that all participants agree on the structures of data that are exchanged. Accordingly, the HHFR covers the larger data domains rather than the MTOM/XOP and the DIME. The HHFR cover, the more general user defined data structure by negotiating it.

5 Summary

Conventional XML processing imposes a performance overhead, and mobile Web Services are particularly sensitive to this overhead. There have been many attempts to develop a binary representation of XML, and many attempts to build efficient mobile Web Services. In this lineage, XMill provides an efficient way to compress XML documents, while Fast Infoset from Sun Microsystems combines efficient XML processing with mobile Web Services.

In some application domains, the need for processing speed is more important than preserving the self-contained nature of individual SOAP messages. Approaches based on streams of messages, like ExtremeFastWS of Extreme! Lab or Fast Web Services of Sun Microsystems [115], may perform better such domains.

Chapter 3.

Background on Mobile Devices

The mobile computing environment adds fundamentally new problems (e.g. intermittent connection, address migration, and low power) to the existing problems of conventional distributed computing. In this chapter, we overview mobile computing and the current status of mobile Web Services.

1 Mobile Computing Environment

Forman and Zahorjan [48] identified the characteristics of a mobile computing environment. Although efficient power consumption is not a particularly important issue in conventional distributed computing, in mobile computing, it is considered critical because minimizing power consumption can improve mobile devices portability by lengthening the life of a charge. The more processor cycles and the more wireless communication the device uses, the more power it consumes. This implies physical

constraints on mobile computing including low bandwidth communication and limited processing memory. Therefore, the programming library is typically not as capable as in conventional computing.

1.1 Limited Programming Library

Generally, mobile devices are equipped with small memories. Smart-phones typically have less than 64MB of non-volatile memory. A tiny footprint is still the most important requirement for the programming library and an application program on a mobile device, even though Secure Digital (SD) flash cards could add a gigabyte or more memory space.

Because of this code size issue and the limited instructions of native operating systems in mobile devices, like PalmOS [17], Symbian [18], and WindowsCE [19], the wireless programming environment is not as rich as a typical wired environment. The Java Platform suffers just these from the same limitations. Java for small and wireless devices – Java 2 Platform, Micro Edition (J2ME) [8] – provides far fewer library packages than the Standard Edition (J2SE) [9].

As depicted in Figure 3.1, the J2ME organization involves configurations and profiles. A configuration defines a basic, lowest common runtime environment. It includes a virtual machine and a core library. There are currently two configurations. The Connected Limited Device Configuration (CLDC) supports low-end and resource-constrained devices, like cellular phones, pagers and low-end PDAs. The other configuration, the Connected Device Configuration (CDC), supports high-end connected devices like high-end PDAs, set-top boxes and car navigation systems. CLDC requires a

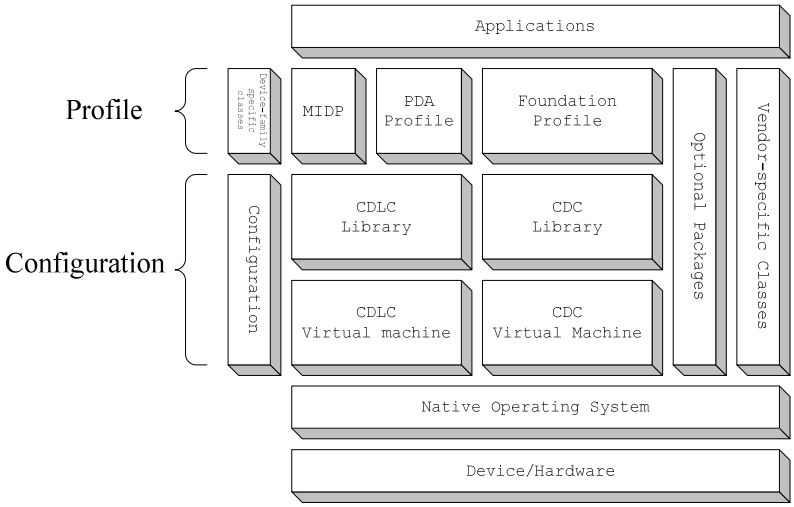

Figure 3.1. J2ME Components and Organization

total of 192KB and includes the tiny K Virtual Machine (KVM) instead of a conventional Java virtual machine. The core library of CLDC is a subset of the standard core Java language package. It introduces a `javax.microedition.io` package to support network access. In the package, a large part of J2SE capabilities is removed from `java.io` and `java.net` packages. Above here CDC requires 4.5MB memory and includes many more core J2SE classes. In summary, CDC includes more libraries than CLDC, but requires more memory space than CLDC.

The other important feature of J2ME is the profile. It provides classes for specific uses like maintaining and updating persistent data in a local device or providing user and networking interfaces. There are five profile standards in the J2ME. Among them, the Mobile Information Device Profile (MIDP) is the most popular and most deployed profile based on the CLDC. Combined with the CLDC, MIDP supports the smallest and most constrained devices currently available. The combination provides a smaller package than other profile and configuration combinations. For example, it only supports a limited Graphical User Interface (GUI) with `javax.microedition.lcdui`. More advanced

packages like `javax.swing` and `java.awt` are not supported. In addition, other features, like math, input/output, and Java Native Interface (JNI), are not supported with this combination.

Earlier, we describe mobile programming libraries focused on Java technology. Similarly, other languages for mobile platforms, like C# [10] and Visual Basic [11], are also limited compared to conventional desktop computing environment.

1.2 Network Connection

By definition, mobile devices connect with remote resources over a wireless network, which allows for mobility. Wireless connections give mobility, but the connection yields lower quality of service than wired connection. Naturally, wireless connections using radio waves face more degradation factors than wired connections. Geographical obstacles, such as intervening buildings and landmarks, block the radio signal and introduce noise, resulting in low bandwidth, high error rates and frequent disconnections.

Moreover, the performance and quality gap between wireless and wired communication environment will not close quickly. Consider, for example, the deployment of the third generation of cellular technology (3G). In order to help resolve the bandwidth problem, the International Telecommunications Union (ITU) defines the global standard for 3G technology in its standard IMT-2000, which is expected to provide faster connections (300~500kbps for downloading and 56~90kbps for uploading) for voice data and non-voice data such as video telephony. IMT-2000 is supposed to be a single standard, but in practice IMT-2000 has been split into several radio interfaces including UMTS (Universal Mobile Telephone System) [116] and CDMA-2000 [117].

UMTS is time-code based and the successor of GSM. Service providers of UMTS and W-CDMA, which is the base-technology of the UMTS, include FOMA (Freedom of Mobile Multimedia Access) offered by Japanese mobile phone operator NTT DoCoMo and by Vodafone in the UK, Germany, the Netherlands and Sweden. In early 2006, Cingular has deployed UMTS networks in several metropolitan areas in the US, such as San Francisco and Seattle[20]. CDMA-2000 is the successor of CDMA (Code Division Multiple Access). Many CDMA mobile service providers in US, Japan, and Korea (outside GSM zone) adopt EV-DO (Evolution-Data Optimized)[21] as the 3G technology, which is a CDMA-2000 based wireless radio broadband data standard. In the US, Verizon Wireless and Sprint deployed EV-DO service in major cities as of early 2006.

This is a big improvement in the connection speed available through wireless devices, especially if we compare it to the current 2.5G cellular technology like GPRS (General Packet Radio Service) [118]) or 2.75G services like EDGE (Enhanced Data Rates for GSM Evolution) [119] with up to 56kbps connection. But the bandwidth and quality gap between wired and wireless communication remains large even with the fully serviced 3G connection. And in most cases, the cellular bandwidth that comes with a pay-per-packet policy is much more expensive than what is available through a wired platform device.

[20] The US has not yet provided radio spectrum for UMTS, so services should share with 1G and 2G networks. This makes US operator spend more effort for developing networks.
[21] Abbreviated as EV-DO or 1xEV-DO and often EVDO

2 Current Web Services supports in mobile computing

In this section we overview the current status of mobile Web Services, which requires low bandwidth and efficient message processing compared to conventional and wired distributed computing.

2.1 Overview

Lately, Web Services on mobile devices are becoming more feasible because of technological progress in both mobile devices and Web Service. On the other hand, the performance of the mobile device is greatly improved by faster processor, larger equipped memory, and enhanced user display, and connection to the network has become easier through a widely available packet-switched 2.5G or 2.75G networks as well as through the third generation networks, which are in an early developmental stage in US. Meanwhile, the Web Service technology is getting popular as a tool to connect heterogeneous applications. Flexibility and universality, originating from the use of XML based SOAP and the ubiquitous HTTP, help integrate disparate resources over various computing environments. We expect a synergy between these two sides, creating faster and better connections. But, as described in the previous section, mobile computing is still bound by hardware and network limitations despite innovations.

These constraints – both physical and programming limitations – make it hard to adapt conventional Web Service frameworks like Apache Axis and Microsoft .NET to the mobile environment. We propose a novel Web Service communication framework providing efficient message exchange to overcome these constraints. Our discussion is

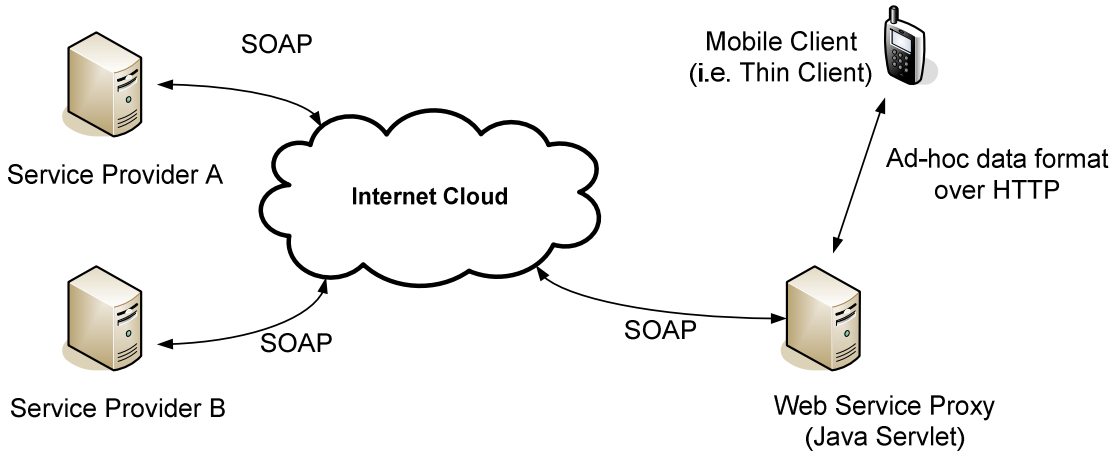

Figure 3.2. Computation Offloading Scheme Using Proxy for Mobile Web Service

particularly focused on mobile computing environment in this dissertation, but the approach may be more general.

2.2 Offloading Computation Using the Java Servlet API

An early model of the mobile Web Services was based on a computation off-loading scheme, called the wireless portal network architecture [66] as shown in Figure 3.2. Since it is simple to implement as a proxy to mobile applications, many experiments are done using the Java Servlet API [24]. A servlet is a Java object which uses the functionality provided by a Java platform that receives HTTP requests and generates responses. Servlets are the Java counterpart to dynamic web contents technologies, such as Common Gateway Interface (CGI) and Active Server Pages (ASP). Yet, as we have mentioned, despite extensive experimentation, there still exists an API gap between wired computing and mobile computing. By using a servlet as a gateway (or a proxy), mobile application developers can build a thin Web Service client on a mobile device. The thin client makes a request with only core data. The gateway converts the data to a valid SOAP request and

delivers it to the ultimate receiver. The problem with mobile devices is that they can only run thin client applications that are not Web Service capable, which means that offloaded functionalities reside on the server where the servlet is running. This ad hoc scheme is easy to implement, so there are many projects which had followed the scheme. But this approach loses an important Web Service characteristic – interoperability. Because they are tailored to the proxy interface, ad-hoc client applications are not interoperable with other conventional Web Service participants.

2.3 kSOAP and J2ME Web Services

Recently mobile specific SOAP APIs have become available, allowing one to design a traditional Web Service architecture with mobile devices. There are two major products in the community: the first is the kSOAP [14], and the other is a specification defined by the Java Community: J2ME Web Services (JSR172) [73].

Because it provides a tiny footprint (about 40kb) and an easy-to-use SOAP library there are many mobile Web Service implementations using the kSOAP library, which is the product of an open source project, Enhydra[22], led by Stefan Haustein. kSOAP is an XML-RPC based SOAP implementation. Its functionality is provided by the `HttpConnection` class and its `call()` method. Unlike traditional APIs, kSOAP and kXML together offer an efficient Web Services programming environment for the mobile device. In our own HHFR design, we use kSOAP and kXML to build a negotiation message and to parse an XML schema.

[22] The project also provides an XML pull parser with a tiny foot-print, called kXML for the MIDP environment [15].

The slimmed-down library of J2ME left out essential packages for Web Services, like the Java API for XML Parsing (JAXP). The Java Specification Request 172 (JSR 172) covers the two components that are needed to implement Web services on mobile devices, but which are missing in the standard J2ME library: remote service invocation and XML parsing. JSR 172 supports SAX (Simple API for XML) 2.0 based JAXP 1.2. It also supports major Web Services specifications, including SOAP 1.1, WSDL 1.1, XML 1.1, and XML Schema. It doesn't include UDDI (Universal Description, Discovery, and Integration) because the specification focuses on how to consume remote services, not how to provide them. In other words the Web Service API on J2ME is a client-oriented API with a JAX-RPC style runtime environment.

Both APIs provide us enough features to implement Web services on the mobile devices, but the mobile environment still does not function equivalently to the conventional computing environment. The APIs of kSOAP and JSR 172 still inherit the performance limitations of conventional Web Services. So the use of the libraries is limited to lightweight applications.

Chapter 4.

Handheld Flexible Representation Architecture.

In this chapter we present a new software architecture designed to optimize and expand 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 overview of the proposed HHFR architecture design appears in Figure 4.1. Here we will propose the full and idealized HHFR architecture, although in later chapters we discuss a prototype where practical considerations require some compromises. We will present an analysis in chapter 8 of the differences between the full architecture and our prototype. Despite these

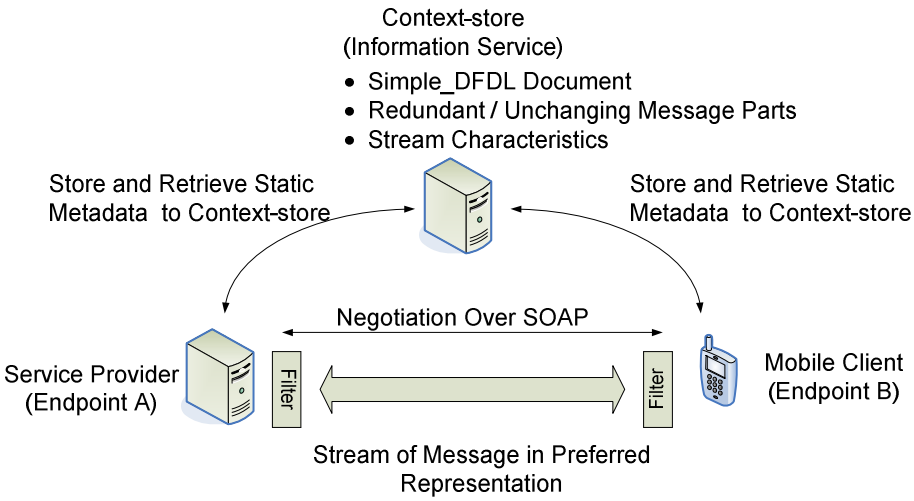

Figure. 4.1. Illustrated Overview of HHFR Architecture

differences, we will also argue that the prototype is able to test and analyze the key features of the full HHFR architecture.

1 Design Overview

Before presenting the HHFR architecture in detail, we must enumerate the main aspects of the new architecture design:

Replacement of XML Syntax with Optimized Representation: HHFR provides a communication option to exchange SOAP messages in a optimized representation by separating the SOAP message semantics from the XML syntax of the message. For example, a fragment of a SOAP message, `<year>2006</year>`, can be separated into its XML syntax and its value, `2006`. A conventional SOAP message is represented in XML-based syntax as structured and typed data. This representation causes the performance bottleneck discussed in Chapter 2, which is magnified in a mobile environment. The

representation, distinguishing between the data structure and types and the SOAP Body payload[23], is negotiated in the beginning of a message exchange. In this stage, a restricted XML Schema, Simple_DFDL, is used to characterize the representation of the message semantics, i.e. SOAP Body payloads. A binary representation is one of the multiple representations supported by the architecture. We define an optimized representation as a representation that is preferred in a given environment with criteria. For instances, a preferred representation to applications with a bandwidth criterion is an optimized representation that requires minimal bandwidth to transmit. Similarly, a preferred representation to applications with a CPU cycle criterion is an optimized representation that requires minimal amount of processing time.

Focus on Message Streams: HHFR works best for Web Services, where two participating nodes exchange a series of messages, which we define as a stream. For applications using a specific service, messages in the stream have the same structure and the same data type for information items, when application use the same service repeatedly. Most of the message headers are unchanged in the stream. Therefore, the structure and type of SOAP message contents and unchanging SOAP headers may be transmitted only once, and rest of the messages in the stream have only payloads.

Context-Store as a Repository: In the HHFR architecture, a Context-store module holds the message stream's static information including a) the unchanging or redundant SOAP message parts, b) the Simple_DFDL file as a data representation, and c) other stream characteristics, which are negotiated at the beginning of the stream. By saving

[23] The term, XML syntax, are used to mean a data structure and types.

unchanging or redundant SOAP headers and a data representation in the Context-store and sending optimized binary format messages, the architecture reduces the size of messages.

Interoperable Web Service Architecture: In contrast to other ad-hoc solutions to the SOAP performance problem, this architecture will not change the overall interoperability of existing Web Service standards. We do not need non-Web-Service data representations and a non-Web-Service transport mechanism. Our approach provides seamless integration with current Web Service applications by using conventional SOAP messages to set up an optimized representation and a transport. Whenever responding (or receiving) node claims it is not compatible with the HHFR, the participants can fall back to the conventional SOAP communication.

2 SOAP Infoset Based Data Model and Separation of Representation

We present here our HHFR data model based on the XML Infoset specification, and explain how we separate the message semantics from its XML syntax. The essential idea of the HHFR architecture is to provide an optimized data representation with a streaming style message exchange between two participating nodes. We design the architecture to provide a representation optimized to different criterion according to the given communication characteristics while not sacrificing SOAP compatibility. The optimized representation includes, but not limited to a binary and a conventional SOAP representation.

2.1 XML and SOAP Infoset

SOAP is an XML based language. The latest specification, SOAP 1.2 [84], is defined using the XML Infoset. XML is a self-contained and structured language, which is well suited for interchanging data in distributed computing. It is a syntax for building structured contents. The syntax is described in the XML specification using the Backus-Naur form [EBNF]. This strict specification keeps the XML parser and application development simple and error-free. Since the original specification was put forward, an increasing need has emerged for an abstract data model of XML corresponding to the logical document structure.

The XML Infoset specification states:

> "[This] Specification defines an abstract data set called the XML Information Set (Infoset). Its purpose is to provide a consistent set of definitions for use in other specifications that need to refer to the information in a well-formed XML document."

The XML Infoset specification was introduced to facilitate the definition of other languages that are based on the XML data model. An immediate benefit from the specification affects application design and developments which manipulate a data model through XML APIs. The model defined by the XML Infoset is not tied up with any specific XML API, like the Document Object Model (DOM), the Simple API for XML (SAX), or the XML Pull Parser (XPP). Thus, the application development is free to

define a data model as far as it follows the XML Infoset specification. One of the possibilities allowed by the XML Infoset specification is for the data model to have parsers that read a binary form of XML.

In our architecture, we define the data model based on SOAP Infoset. In this way, the HHFR architecture is able to separate SOAP message semantics from their syntax, i.e. the representations, without losing any content or properties of the message.

2.2 Binary Representations of SOAP Message

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). Even a very simple message with a single text element can have its size cut in half [37]. In a conventional computing environment, it is always good to have a reduced amount of data to exchange. However, in mobile computing, this is necessary because of the narrow bandwidth connection. Although visionary statements claim that mobile computing will eventually have connections as fast as wired computing, this is not much help right now or in the near future. The faster the connection a mobile device uses, the more battery power it consumes. This is a critical factor in mobile computing. As a result of this limitation, reduction of message size is a significant issue in the message architecture.

A binary message representation also helps the HHFR architecture to avoid textual conversion. The architecture simplifies the conventional encoding/decoding stage[24], 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 [36].

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 which is discussed in Chapter 5 to distinguish message semantics, i.e., information, and message syntax.

As we discussed in Chapter 2, there are two different approaches to achieving a binary representation of XML documents. In our architecture, we use a message stream approach in order to optimize and simplify each message. A special stage called a negotiation stage is introduced to negotiate characteristics of the message stream including the representation of messages. The details of the negotiation stage will be described later in this chapter.

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

2.3 Simple_DFDL

To define the XML syntax that we separate, the architecture makes use of the XML Schema Definition (XSD). This recommendation of the World Wide Web Consortium (W3C) formally describes how the elements in an XML documents should be constructed. Originally it was intended to be used in the checking the validity of XML documents; however it is also an abstract representation of an XML document's structure and its characteristics, including elements and data type. There is, however, an issue to be considered in using the XML Schema to define the structure and types of a SOAP message stream. As described previously, the XML Schema replaces all types of SOAP messages well. But the XML Schema itself is an XML document, and therefore, there are limitations in its ability to represent the structure of the SOAP message, since the XML Schema definition contains options and references. Therefore there can be many different structure for instances of the XML Schema definition.

We constrain the XML Schema definition to achieve a single structure by parsing the XML Schema document itself (i.e. the Simple_DFDL document is a sample XML instance of a message which will be used for message exchange). There are several restrictions to the Simple_DFDL definition compared to the XML Schema definition. Some of them are presented here to present the idealized design and the details of the Simple_DFDL definition are discussed in Chapter 6. Some of these restrictions include:

- A Simple_DFDL document should be a single XML Schema document rather than multiple documents.
- 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 do not support facets like `minInclusive` and `maxInclusive` to restrict the valid values.

These restrictions make an instance of a given Simple_DFDL definition, i.e., a Simple_DFDL document, have only one structure. Parsing a valid XML document to the given Simple_DFDL definition produces a single structure, and therefore, the HHFR architecture can use a Simple_DFDL document as a representation of both structures and types.

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. We discuss the implementation of HHFR and its representation conversion in detail in Chapter 5.

3 Negotiation of Characteristics

In this section, we discuss the necessity and role of the negotiation stage. 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.

3.1 Supporting Alternative Representation of SOAP Message

A stream of messages shares the same representation, meaning these messages share identical structure and type of XML fragments, i.e., SOAP Body parts. As we discussed in the previous section, the separated structure and types can be represented as a restricted XML Schema document, i.e., a Simple_DFDL document. To establish an optimized representation stream, Simple_DFDL documents for both request and response messages should be exchanged at the beginning of the stream. 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. In the HHFR architecture, both the representation and headers are archived in the Context-store.

3.2 Negotiating Characteristics of Stream

Because most connections between mobile devices have narrow bandwidth and high latency, the most popular transport protocol in Web Service technology, i.e. HTTP, can be expensive to use in mobile computing. This problem has been studied by researchers

[36], who suggest using chunk overlaying and a pipelined-send through HTTP 1.1 connection to improve performance. Those operations require a 'Persistent connection' or a 'Keep-Alive' feature. Unfortunately, these are not always available for network protocol implementation on all mobile devices and all cellular networks.

For the above reasons, the HHFR architecture provides high performance communication channel options, including TCP and UDP, as well as a default HTTP transport. The high performance communication channel options offer an asynchronous (or also called non-blocking) messaging scheme in place of the HTTP synchronous request-response mechanism. The options in HHFR look similar to previous ad-hoc solutions to Web Service performance issues as discussed in Chapter 2. Once the negotiation is successful, a high performance communication channel option can be used.

Reliability issues are addressed in the negotiation stage as well. For example, UDP transport is a simple high-performance datagram Internet protocol [120], which doesn't provide either the reliability or an ordering guarantee like TCP does. This means that Datagrams may be missing or arrive out-of-order. Thus, the architecture design implements the Web Service Reliable Messaging (WS-RM) specification on UDP transport. As an example, Figure 4.2 depicts a possible message exchange between WS-RM

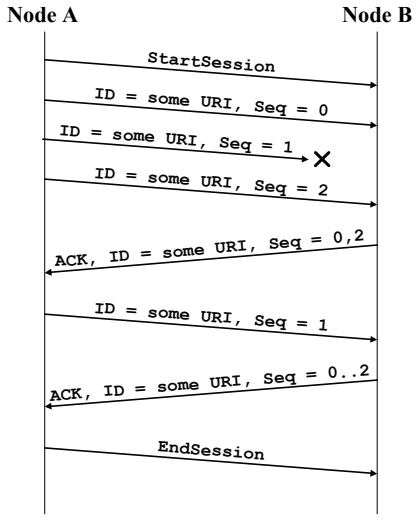

Figure. 4.2. Possible Message Exchange Between Two WS-RM Endpoints

nodes through UDP or TCP transport. A detailed implementation of the high-performance communication transport is presented in Chapter 5.

3.3 Negotiation Stage

The issues discussed above are negotiated in an initial Negotiation Stage. The negotiation stage uses a single (or multiple, if necessary) conventional SOAP message[25], 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. Table 4.3 contains examples of element information items in the negotiation schema.

Table 4.3 Summary of Element Information Item in HHFR Negotiation Schema

Element Information Item	Element Information Item Syntax
a fast connection	`<StreamURI>Some_URI<StreamURI/>`
a reliability scheme	`<RM_Scheme>Some_Reliable_Scheme<RM_Scheme>`
a HHFR capability	`<isHHFRCapable>true</isHHFRCapable>`

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

A HHFR-capable node responds with a true value in an `isHHFRCapable` element. If the element in the response message contains a negative value, or a fault message with error information is received, the sender must fall back to a regular SOAP protocol message exchange. If the element contains a positive value, a negotiation handler starts processing the negotiation message to set up a message stream. A successful end of the negotiation stage leads to a message streaming stage according to the negotiated characteristics.

To help readers understand better our negotiation stage, we can compare our negotiation stage with the one in Web Service Secure Conversation (WS-SecureConversation) specification [94], which defines the architecture to exchange multiple messages securely; a secure context in the architecture refers to an established authentication state. To establish the secure context, a secure context token, which contains a secret key and other security properties, should be created and propagated among participants. The participants in a sequence of message exchanges negotiate on the contents of the security context token. The negotiation structure is similar to the one used in HHFR. The initiating participant sends a `<wst:RequestSecurityToken>` request to the other participant, and a `<wst:RequestSecurityTokenResponse>` is returned. If it is successful, a response from the final participant contains `<wst:RequestedSecurityToken>` and `<wst:RequestedProofToken>` pointing to the secret key for the context.

The number of participants in these different architectures is different however. The WS-SecureConversation specification is designed based on the assumption that these are multiple participants. The HHFR architecture design is a general framework for mobile

Web Services, but since it is focusing on the message transmission between only two participants, the negotiation stage in HHFR does not define a propagation of the negotiation request. The core object to negotiate is different as well. WS-SecureConversation negotiates a shared secret and the HHFR negotiates a schema file of the exchanging data representation.

Compared to the conventional Web Service communication method, the negotiation stage is an additional overhead[26] 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. In Chapter 6, we present and analyze performance data for a prototype implementation of the HHFR architecture.

4 Message Handling

This section overviews message processing in the HHFR architecture. The outermost XML element of a SOAP message is the SOAP envelope element. It is composed of a body (payload) that contains program data and optional headers. The headers contain additional information, such as parsing instructions, security information, routing information, and reliability information. The architecture handles the static information of messages (unchanging headers) and the dynamic information (payload and changing headers) of the stream differently.

[26] Others are a Context-store accessing overhead and Simple_DFDL designing overhead.

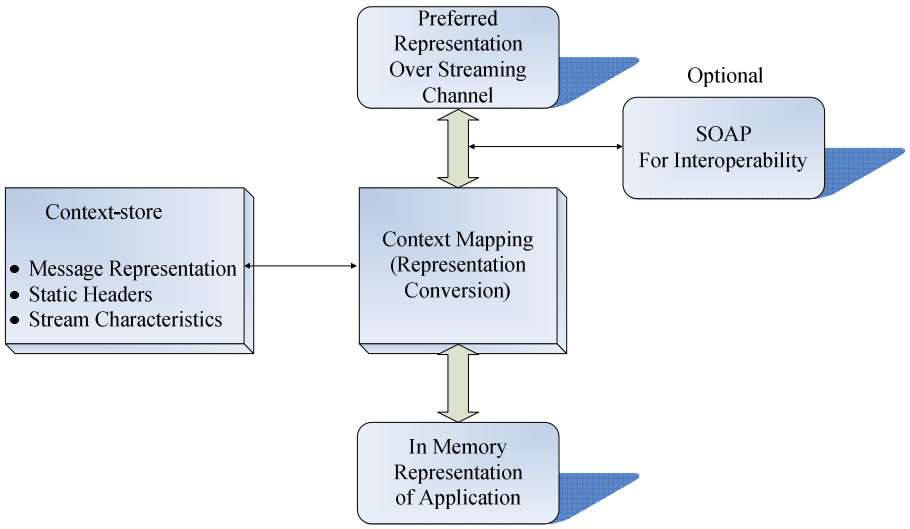

Figure. 4.3. Relationship of Different Forms of SOAP Messages and Their Defining Context

4.1 Handler for SOAP Header Processing

As discussed, in HHFR, the static unchanged headers of a SOAP message in the stream are stored to the Context-store at the negotiation stage. Headers unique to an individual message are processed by the appropriate handlers and are transmitted as an additional part of the optimized representation of the SOAP message. As discussed in Chapter 2, there are two methods to process headers in appropriate handlers. The HHFR architecture design uses a handler modification method where a modified handler understands a HHFR message packet which includes headers. In the method, a message packet contains header information as well as the body contents. A good example is a WS-RM header that marks sequence numbers or ACKs. In this example, the reliable message handler (RM-Handler), which could be either a part of the message handler or an independent header processor, understands the structure of the message and retrieves a sequence number or ACK by probing the message to find the information.

4.2 Message Transformation Process

As discussed in Chapter 2, a message which goes through an individual message transformation approach holds its logical representation. On the other hand, a message stream approach requires an internal Data Structure Object that holds a representation of messages in the stream to be able to process messages. The object is used to extract information from an exchanged message. The architecture builds a data structure object by processing a Simple_DFDL object during a negotiation. Figure 4.3 shows such an abstract message transformation process.

We originally intended to use Data Format Description Language (DFDL) to define a binary format representation, and to utilize the DFDL library to read and write binary format data. But the schedule of the DFDL implementation does not match ours. Instead of using DFDL, we use a simple restricted schema definition for the prototype implementation, i.e., the Simple_DFDL definition discussed in the previous section. For array processing, we added a Built-in simple type, `array`, to support any array type elements in a representation to the Simple_DFDL definition. A syntax example is shown below. The detailed definition of the Simple_DFDL is presented in Chapter 5.

```
<element name= "MyArray" type="array"
               primitives="float" value="90">
```

After the architecture initiates a stream that uses a binary representation, a stream-reader reads and a stream-writer writes the in-memory view data to a binary stream for a high-performance communication connection. A reader does a sequence of typed-reading, (e.g., `readFloat()`) to get the information items of a SOAP message from the network stream according to the internal data structure, while a writer does a sequence of typed-

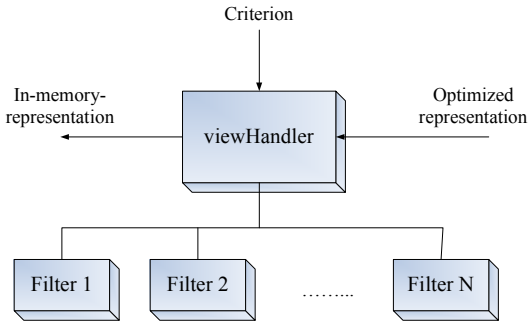

Figure. 4.4. viewhandler: Selecting Filters for Optimized Representation

writing, (e.g., `writeInt()`). Alternatively, in a case where the conventional SOAP message style representation is being used, a SOAP message is simply exchanged.

A transformation from the optimized representation object (message) to an XML document in the HHFR architecture should be invertible. As explained in Chapter 2, the inversion does not cover the normalized information items in general. Because individual messages in the HHFR architecture are not self-contained, a filter which inverts the optimized representation to conventional XML needs information outside the messages, i.e. internal data structure which is built from a Simple_DFDL document. By parsing a Simple_DFDL document, the architecture builds an internal data structure that contains the names of element information items, attributes, and child properties. 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.

4.3 View[27] Selection Handler

As shown in Figure 4.4, a viewHandler in the HHFR architecture selects an optimized view (i.e. optimized representation) which is another name for the data representation. In our architecture design, each view is optimized according to the characteristics that are negotiated during the negotiation stage and the principles that are predefined by an architecture specification. As discussed earlier in this chapter, 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 viewHandler responds to the negotiation requestor with views (i.e., representation) supported by the viewHandler. Suppose the viewhandler 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.

Presentation formatting: Transforming representation, in the prototype is similar to "formatting representations for an end-to-end communication" [3]. Both provide a suitable format for transmitting messages. The primary differences between them are that transforming representation of the HHFR prototype provides multiple representations to optimize a communication format, and it reduces data manipulation overhead. The presentation formatting is the transformation of communication data from the in-memory representation of the application to a form able to be transmitted over-the-wire. Mostly, the representation format is predefined in an appropriate and efficient way between two

[27] We use this term to refer to representation throughout this chapter.

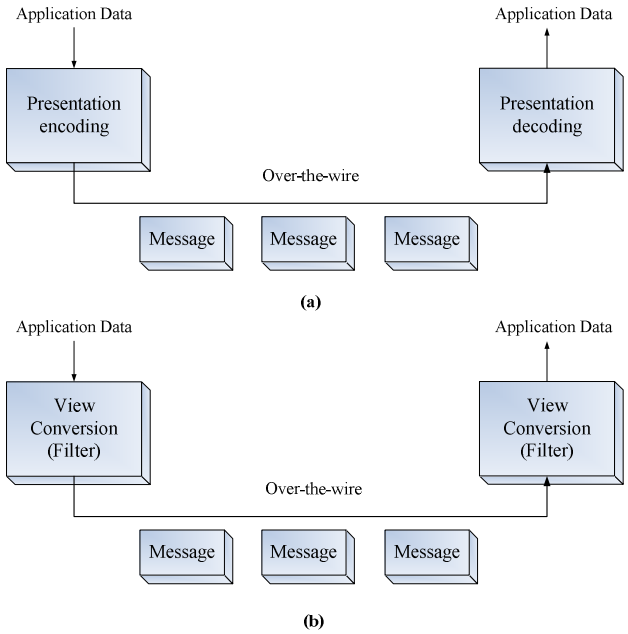

Figure. 4.5. Representation Formatting (a) and Transforming Representation (b)

participating remote entities. A representation formatting is shown in Figure 7.5 (a) [3] and (b) shows a transforming representation process.

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 view conversion of the HHFR prototype has several options in multiple representations, and can provide an optimized representation for the application and the given network characteristics.

The goal of the representation transforming and the presentation formatting are similar and the steps of both involve data conversion, packing a structure, and serializing (e.g. the message encoding process in the Remote Procedure Call (RPC) stub). One main

difference, however, is the distinct step in the representation transforming in order to choose an optimized representation according to the communication condition.

5 Context Store

One of the essential components of the architecture is the Context-store. In the previous sections, we discussed the unchanged information in the conventional SOAP message, such as namespace and encoding style information. The Context-store archives the static context information from a SOAP negotiation message, such as unchanging or redundant SOAP headers, a Simple_DFDL document as a message representation, and the characteristics of the stream. 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 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.

The context of the stream contains shared information among nodes and the HHFR specification itself. This is where the WS-Context specification [95] 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) [121]. The HHFR architecture design defines information in the context-store with a URI. In fact, we derived the HHFR scheme itself as *URI-S*. The current representation

of the message in the stream is indicated *URI-R* and the choice of transport protocol is *URI-T*. We use the Fault Tolerant High Performance Information Service (FTHPIS), which is WS-Context compliant and was developed by the Community Grid Laboratory at Indiana University. We will present details of the implementation and integration in Chapter 7.

Chapter 5.

The Prototype Implementation of the HHFR Architecture

To demonstrate the effectiveness of the HHFR architecture, we have implemented a prototype mobile Web Service framework based on this architecture. The prototype implementation makes the HHFR design concrete by supporting three key HHFR architecture design points: the DFDL-style data description language, message streaming, and using the Context-store to store redundant or unchanging static data. Combined together into a single complete system, the HHFR prototype (our research framework), the DFDL-style data description language and message streaming provide an efficient and flexible mobile Web Service communication method.

A normal stream of the runtime system is as follows: 1) an HHFR-capable endpoint sends a negotiation request to the intended endpoint. The negotiation request is a

conventional SOAP message that includes characteristics[28] of the following stream. 2) In the negotiation message, a service client endpoint (a negotiation initiator) sends an input data description written in the Simple_DFDL, which we describe later in this chapter, and a service endpoint (a negotiation responder) sends an output data description. 3) The two endpoints use a second transport channel for message exchange where they stream messages. Messages in the stream are in the form of the negotiated representation. 4) The redundant or unchanging message parts (static metadata) are stored into a dynamic metadata repository, the Context-store. If the service endpoint responds with a false value for HHFR capability or sends an error message (i.e., it is not HHFR-capable), the negotiation initiator must fall back to conventional SOAP messaging.

In this section, we discuss the implementation details of the HHFR prototype. First, we overview the prototype implementation. Second, we discuss our negotiation scheme. Then we discuss the implementation of each of the three key design points in turn with the exception of the Context-store which is discussed in Chapter 7.

1 Prototype Implementation Overview

The prototype implementation is a pure Java runtime system designed according to the HHFR architecture design. We chose Java as a language platform for both mobile and conventional sides because it is portable across platforms, and the Java 2 Platform, Micro Edition, 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-

[28] The characteristics that could be negotiated: a reliable messaging scheme, security related issues, and other Web Service Specifications like WS-Addressing and WS-Context. Obviously, the most important and essential negotiation item is the HHFR-capability for both the request and response message.

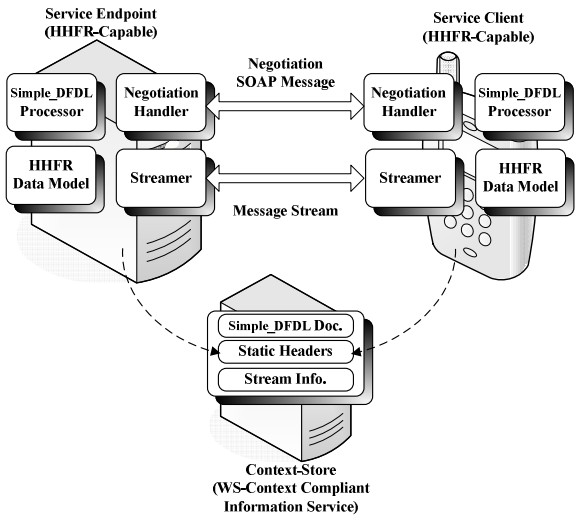

Figure. 5.1. Simple Overview of Prototype Implementation

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).

1.1 Implementing three key design issues

As depicted in Figure 5.1, the HHFR prototype, our research framework provides an optimized communication framework for mobile Web Services by implementing three key design issues[29]. The HHFR prototype implements a) the *streamer*, which is "the interpret style stubs" [3], to encode and decode the HHFR on-the-wire data format, b) a negotiation stage, c) high performance communication transport options, and d) the Context-store. Except the Context-store implementation, which is presented in Chapter 7 in detail, we discuss the implementation details of the prototype here, including the Simple_DFDL and the DFDL-style data description language.

[29] As we discussed, they are DFDL-style data description language, message streaming, and using the Context-store.

The Simple_DFDL is a data description language that is a small subset of the XML Schema Definition with a few additions. It is used to describe a data structure and type i.e. a representation[30]. The prototype provides the option of using multiple representations by separating information item data from the XML document syntax as we described in Chapter 4. An application may be capable of using several different data representation formats, one of which may be preferred over the others. In many cases, the HHFR participating-endpoints prefer a binary format representation because of efficiency. A fast transport channel and streamer enable the participants to exchange messages in a conversational fashion (in a stream).

The prototype is Java based and involves two runtime systems: a Java 2 Platform, Standard Edition (J2SE) runtime, which runs in an AXIS container, and a Java 2 Platform, Micro Edition (J2ME) runtime, which runs as a MIDlet. Because we recognize the limited memory available, the prototype implementation for mobile devices occupies only a 100KB, which includes the kSOAP and kXML libraries.

1.2 Utilizing existing efforts: Apache Axis, kSOAP, and Information Service of CGL (Fault Tolerant High Performance Information Service, FTHPIS)

We chose to use existing technologies to address and implement issues that are extraneous of our research interests such as a Web Service container, SOAP/XML parsers for mobile environments, and a metadata repository Web Service.

Web Service containers are widely implemented to perform a primary function (i.e. provide the SOAP server and additional functions like support for Web Service deployment). Both Axis from the Apache Software Foundation (ASF) and

[30] We use this term to refer to both data structure and data type throughout this research.

Microsoft .NET are well developed and popular implementations, but we chose Apache Axis as the SOAP container for the prototype implementation, because of it is an open-source and Java based framework[31]. The kSOAP[32] is a tiny foot print SOAP/XML parser for applications in J2ME environments and it provides the library used to process negotiation requests/responses in SOAP, to interact with the Context-store, and to parse the Simple_DFDL to get a representation format.

Details of the WS-Context compliant Information Service (FTHPIS) usage in the HHFR prototype is described in Chapter 7.

2 Negotiation Scheme

A normal HHFR message stream starts with a negotiation stage during which two endpoints exchange negotiation SOAP messages. By design, a negotiation stage is essential to establish agreed upon characteristics for the following stream. During this stage, a service endpoint returns the characteristics, suggested by the negotiation initiator, which have been selected and confirmed by the service endpoint. In the prototype implementation, the stage simply starts when the initiator sends a SOAP request to an intended service endpoint and ends when the initiator receives a response from the service.

An example representation of the negotiation items follows. The `hhfr:negotiationType` type is used as a negotiation request SOAP message to describe negotiating characteristics. And the `hhfr:negotiationReturnType` type is used for a

[31] A C++ implementation is also available.
[32] An overview and description of kSOAP can be found in chapter 3.

negotiation response SOAP message. The following XML fragment is an example of the content of an `hhfr:negotiationReturnType` type.

```
<hhfr:negotiationReturn>
     <hhfr:isHHFRCapable>true</hhfr:isHHFRCapable>
     <hhfr:schema>……</hhfr:schema>
     <hhfr:streamURL>anyURL</hhfr:streamURL>
</hhfr:negotiationReturn>
```

The following describes the elements listed in the above schema example:

`/hhfr:negotiationReturn`

This represents an element of the type `/hhfr:negotiationReturn`. This example uses the `<hhfr:negotiationReturn>` element.

`/hhfr:negotiationReturn/hhfr:isHHFRCapable`

This REQUIRED element of the type `xs:boolean` contains the [Endpoint[33]-HHFRCapability] property of `negotiationReturn`.

`/hhfr:negotiationReturn/hhfr:schema`

This REQUIRED element of the type `xs:string` contains the [Simple_DFDL] property of `negotiationReturn`.

[33] The term 'Endpoint' in the description represents a negotiation SOAP message responder.

`/hhfr:negotiationReturn/hhfr:streamURL`

This REQUIRED element of the type `xs:anyURL` specifies the [high performance channel address] property of `negotiationReturn`.

In addition to setting up the stream characteristics, the negotiation stage distinguishes whether the service endpoint is HHFR-capable or not. Since the negotiation stage is performed using the conventional SOAP protocol, this interoperable method enables the service endpoint (the negotiation responder) to reject a HHFR stream and uses a conventional SOAP based Web Service communication. In this case, `/hhfr:negotiationReturn/hhfr:isHHFRCapable` is false Boolean value in the negotiation SOAP response. In our HHFR programming model, the HHFR-capability is provided as a function return. This means `negotiation()` method, by which the negotiation initiator starts the negotiation stage, gets Boolean value returns.

```
boolean a;
try {
    hhfrproto = new cgl.hhms.hhfr.HHFRHandler(SCHEMA_URL);
}
catch (cgl.hhms.hhfr.HHFRException hhfre) {
    hhfre.printStackTrace();
}
hhfrproto.setWSUrl(s);
a = hhfrproto.negotiation();
```

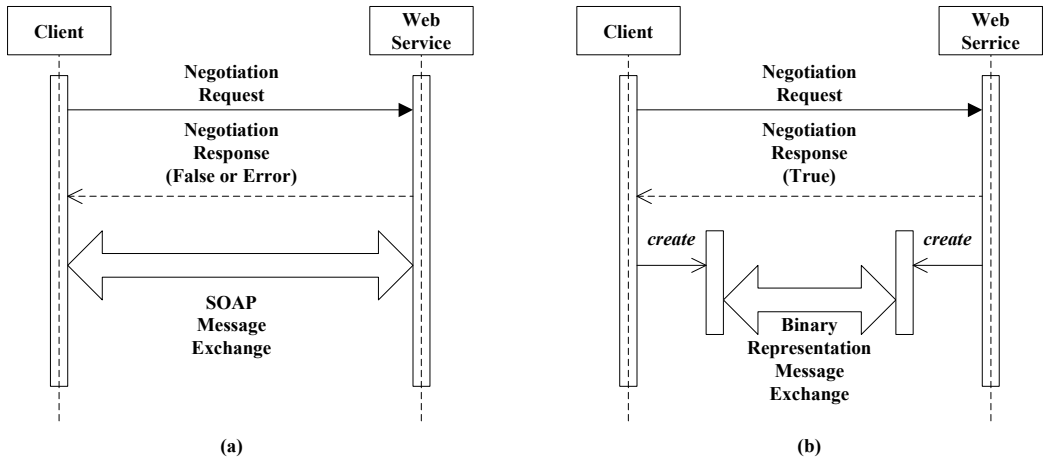

Figure. 5.2. Modified Sequence Diagram of Negotiation Stage (a) is the case of non-HHFR capable node response and (b) is the case of HHFR capable node.

After creating the `HHFRHandler` object and setting the URL of the service endpoint, the service client calls the negotiation method of `HHFRHandler`. According to the return value, a client either initiates an HHFR scheme with a high performance communication channel if it is `true`, or a client must fall back to using conventional SOAP based Web Service communication if it is `false`. The client (the SOAP initiator) must fall back if it receives a SOAP fault, which means the responding service doesn't have the proper (exported) method in it and doesn't understand the negotiation SOAP message. The negotiation stage is depicted in Figure 5.2.

3 The Simple_DFDL: The DFDL-style Data Description Language

In this section, we overview the data description language, the Simple_DFDL that is used to describe the structure and the type of input and output data. We define the purpose of the Simple_DFDL as the definition of the XML Schema Definition (XSD) [122]: "[It] defines a class of XML documents." When the language is combined with

Streamer, which converts exchange data in the preferred over-the-wire representation format from and to the internal HHFR Data Model, they achieve the basic goal of the Data Format Description Language (DFDL). In the following sentence, we overview the DFDL, define the Simple_DFDL, and compare them with each other.

3.1 DFDL Overview

The purpose of the DFDL, which is briefly described in Chapter 2, is to be processable through standardized parsers that read a DFDL description, along with files(s) or stream(s) of raw data to produce structured output. DFDL is proposed and designed for data exchanging between different formats in the grid that is the collaboration of distributed resources to solve large scale problems. Since the distributed resources are heterogeneous in many cases, it is important to provide for a data exchange between resources that use different data format.

The specification consists of the architecture, the data model, and language syntax of DFDL. Language syntax of DFDL corresponds to our Simple_DFDL. The HHFR prototype implements other parts of the DFDL architecture i.e. the architecture and the data model. Their implementations are detailed in the following section.

The architecture defines the parser, which processes a description to produce structured output[34]. It consists of three primary layers: the lower layer (Mappings), the central layer (abstract Data Model), and the upper layer (API). The primary layers are depicted in Figure 5.3. The architecture may be implemented as a library for applications,

[34] DFDL focuses converting raw data to structured output. Our Simple_DFDL provides bi-directional conversion.

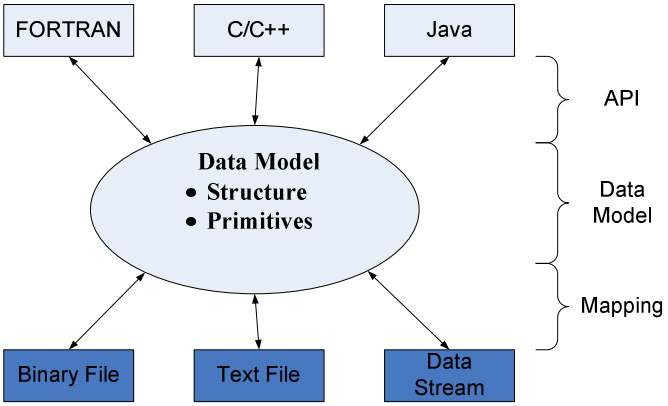

Figure. 5.3. Abstracted Overview of DFDL Architecture.

a standalone tool, or a Web Service. As depicted, the output could be realized through various programming interfaces, such as FORTRAN, Java, C/C++, or XML.

The Data Model layer defines the data structure independent of its physical representation. It supports most types defined by the XSD, including primitive types such as float, double, boolean, and all types of integers, strings, compound structures like arrays and vectors, and user-defined compositions. The Mapping layer defines the mapping between the concrete representation and the information content. Examples of the Mapping description include Endianess (big-endian or little-endian) and the length of integer (4-byte or 8-byte). The API describes how information is instantiated and accessed from the programming language.

The language requirements of DFDL are described in the DFDL Primer document and they are also the primary requirements for our Simple_DFDL. These requirements require the language to

- be able to describe the conceptual structure of the data as a sequence of primitive and composite types.

- support the semantic labeling of the data. This is primarily to support the eventual interpretation of the data by data consumers.
- describe multiple layers of conceptual data structures such as vectors or strings.

The syntax of DFDL is based on XSD, using which the DFDL user describes the abstract data model. Currently, DFDL syntax supports only a subset of the available XSD capabilities, and this also true for our Simple_DFDL. One of the XSD's capabilities used by DFDL is the annotation. XSD specifies its extension by "appinfo" annotation and DFDL uses the annotation to put Mapping information in it.

We use the following example from the DFDL overview report [123] produced by the "XML Virtual Garden" from IBM alphaWorks, to show how DFDL describes data format. In this example, we consider we have an array with a used-defined composition (`struct` or `class`).

```
struct { char c; short s; int i; long l; float f; double d; }
array[2] = {'\1', 1, 1, 1, 1.0, 1.0, '\2', 2, 2, 2, 2.0, 2.0};
```

The DFDL description for the `struct` is:

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:data="http://dataformat.org/">
 <xs:annotation>
  <xs:appinfo source="http://dataformat.org/">
   <data:defaults>
    <data:format data:encoding="bytes" data:byteOrder="littleEndian"/>
   </data:defaults>
  </xs:appinfo>
 </xs:annotation>
```

```
 <xs:element name="sextet">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="group" minOccurs="0" maxOccurs="unbounded">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="byte" type="xs:byte"/>
       <xs:element name="short" type="xs:short"/>
       <xs:element name="int" type="xs:int"/>
       <xs:element name="long" type="xs:long"/>
       <xs:element name="float" type="xs:float"/>
       <xs:element name="double" type="xs:double"/>
      </xs:sequence>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

This description will interpret the byte sequence as the XML data

```
<sextet>

<group><byte>1</byte><short>1</short><int>1</int><long>1</long><float>1
.0</float><double>1.0</double></group>

<group><byte>2</byte><short>2</short><int>2</int><long>2</long><float>2
.0</float><double>2.0</double></group>

</sextet>
```

3.2 Simple_DFDL

The Simple_DFDL is the description language we use to describe the data format for the HHFR prototype; it is a small subset of the XSD with few additions. The architecture of the Simple_DFDL is similar to that of DFDL: The Simple_DFDL describes data format, the Schema Processor (DSParser) builds the HHFR Data Model, and the Streamer converts data from and to the preferred representation format for the data.

3.2.1 Structures and types of Simple_DFDL

Like the XML Schema, HHFR consists of type definitions (Simple type definitions, Complex type definitions, and Model group definitions) and element declarations (Element declarations). The Simple_DFDL focuses on defining the primary component group of Schema components[35], the three components listed above, and Attribute declarations. We describe these components only in terms of the basics that are shared with XSD and in terms of the HHFR specific additions. Detailed information can be found in the XSD Specification.

A simple type definition is a set of constraints and information about the values it encodes. The Simple_DFDL defines a limited number of simple types built-in to the XML Schema, yet we believe there are enough to show the effectiveness of our experimental framework. Table 5.1 lists the built-in simple types of the Simple_DFDL. The current version of the Simple_DFDL doesn't support user-defined `sympleType`. This means it doesn't allow the user to derive a new simple type by restricting existing built-in simple types.

Table 5.4 Simple types built in to the Simple_DFDL

Simple Type	Examples
string	"googling", "headache", "illusion"
int	-2147483648, …-1, 0 , 1, … 2147483648
byte	-128, … -1, 0, 1, … 127
float	-INF, -1E4, -0, 0, 12.78E-2, 12, INF, NaN[36] (32 bit, IEEE-754 1985 floating-point standard)
boolean	true, false, 1, 0

[35] Schema components are the building blocks that comprise the abstract data model of the schema. There are 13 kinds of components in the XML Schema Specification [124]
[36] NaN is "not a number"

The complex type definition is a set of attribute declarations and content types. These are applicable to the [attributes] and [children] of an element information item respectively. It is an element containing other elements in a hierarchical fashion. The complex type element only can have mixed content, and can not have a simple or empty content. This enables us to simplify the complex type definition of the Simple_DFDL. So we declare a complex type element without a mixed attribute. The following is an example of the complex type used in the Simple_DFDL.

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

The array type definition is a Simple_DFDL-specific definition, which is a set of constraints and information about the values in a sequence. It is a limited version of XSD's sequence and defines data values in sequence, not in particle[37] elements. The current version of the Simple_DFDL defines an array type in ad-hoc way. The Schema definition requires for two element information items to be paired together in order to declare an array type. Thus if there isn't a pair, it is not a valid Simple_DFDL definition. This should be generalized and made to conform to other type definitions in the next

[37] '*A particle is a term in the grammar for element content, consisting of either an element declaration, a wildcard or a model group, together with occurrence constrains*' – XML Schema Part 1: Structures [124].

HHFR version by supporting simple content complex type, which enables a simple type to carry the attribute declaration.

3.2.2 Declaration of Simple_DFDL

The declarations of the Simple_DFDL definition also follow XSD Specifications. An element declaration is an association of the element name and its type definition. A declaration of element A as a float type is desired, it would be coded as such:

```
<xs:element name="A" type="float">.
```

As we have described it, an array type needs a pair of information items to declare. The first element declaration gives the parser the name of the array and the size. The second element declaration gives the type of array content. 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>
```

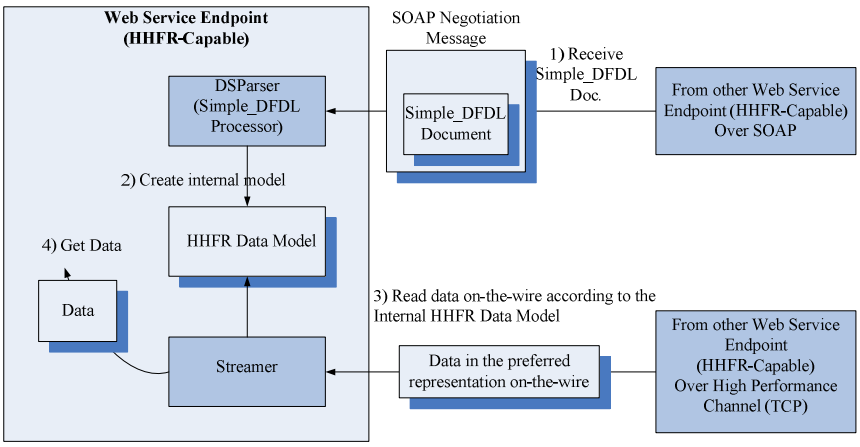

Figure. 5.4. Simple_DFDL Modules in HHFR prototype and Their Interactions.

3.2.3 Data processing using Simple_DFDL

As depicted in Figure 5.4, Simple_DFDL processing is an essential step that enables flexible representation message exchange in the HHFR Prototype. The Simple_DFDL Processor (`DSParser`) gets an Simple_DFDL instance (either a file or a stream), which is contained in the negotiation request and response SOAP message depicted in step 1) of Figure 5.4, as an input and produces an internal HHFR Data Model as output as depicted in step 2). The relation between the Simple_DFDL and the HHFR Data Model is similar to the relation between an XML document and its Java DOM Object.

After two steps, the HHFR runtime is ready to start a high performance communication option, which is discussed in the following section, and to process input data through `streamer`. The `streamer` is an "interpret-style stub" object, which is a popular design style in many data marshalling implementations. Compared to the more efficient "compiled-style stub" [3], which is popular in many client and server RPC implementations, the "interpret-style stub" is more flexible in allowing the dynamic representation of input data. The stub doesn't need to be re-compiled for different data

representations. The stub reads and writes message packets, a message unit in a preferred representation, through switch statements. In the prototype, the binary representation that is a sequence of bytes is the default representation format for the message packet. The simplified message writing is depicted in Figure 5.5.

3.3 Comparisons Between Simple_DFDL and DFDL

Both Simple_DFDL and DFDL try to achieve the same goal i.e. a way of describing interchanged data. They also share a similar architecture: a description language, a specific parser, and a language API. In this section, we compare them to show how they are related and what limitations the Simple_DFDL has. And we conclude based on the comparisons.

The first and biggest difference is a low level mapping. The lower layer of DFDL maps information content from and to a physical representation. The mapping enables DFDL to convert various data formats if it is described. For example, DFDL can map bytes in concrete representation no matter what format it is in (whether it is big-endian or little-endian). The annotation component of XSD, which gives information for both humans and programs, is used to embed the mapping layer information. For example, the following DFDL describe a sequence of an integer and a float.

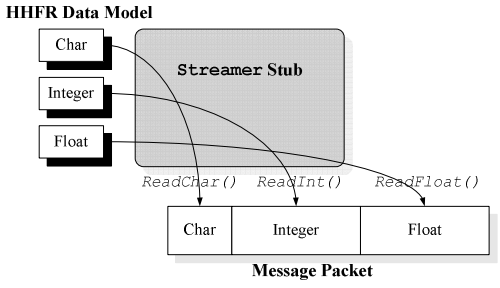

Figure. 5.5 Simplified Message Writing Process Using Streamer Stub

```
<xs:complexType name="example1">
        <xs:annotation>
            <xs:appinfo>
                <binaryProperties>
                    <byteOrder>bigEndian</byteOrder>
                </binaryProperties>
            </xs:appinfo>
        </xs:annotation>
        <xs:sequence>
            <xs:element name="x" type="dfdl:binaryInt"/>
            <xs:element name="y" type="dfdl:binaryFloat"/>
        </xs:sequence>
</xs:complexType>
```

Our Simple_DFDL doesn't define low level mapping. The Schema design is not limited to a single low-level mapping format, but is not implemented for this version. One of reasons for limited low-level mapping is the limited I/O operation supported by the J2ME language. The `DSParser` should have an annotations process capability or any equivalent mechanism to define the property of elements in the future release and this will support a variety of low-level mappings. But, the main goal of our research has been at the level of designing an overall architecture to separate data content from the syntax (representation) rather than implementing a specific language and platform interface to the raw data representation. So we believe Java Object mapping is enough to confirm the feasibility of our framework.

The conversion focus is also different. DFDL focuses on converting legacy non-XML data in order to use them in a grid environment. So it is likely to be used in one-way conversions only. Rather we focus on bi-directional stream conversion. The difference require us to design and implements more efficient approach whereas DFDL provided a more general and flexible approach.

The Simple_DFDL definition does not prevent the use of pointers and references in the schema document. Currently, however, we have not implemented this feature because supporting the reference in a complex type is a non-trivial implementation and it is not our focus of research. We summarize the comparisons between DFDL and HHFR in Table 5.2.

Table 5.2 Summarized Comparisons between DFDL and Simple_DFDL

	DFDL	Simple_DFDL
Low-level mapping	Supports various types of low-level mapping through extending annotations.	Supports only Java Object mapping. One of the reasons is the limited J2ME I/O supports.
Conversion Direction	Focuses on non-XML data to XML conversion.	Focuses on bi-directional stream conversion.
Reference and Pointer	Supported.	Not limited, but they are not supported in current version.
Array Support	Not included.	Supported in Ad-Hoc method.

4 Data Streaming

Data streaming is the key feature of our HHFR Prototype design and it enables the system to achieve efficient message exchanges in mobile Web Service environments. By

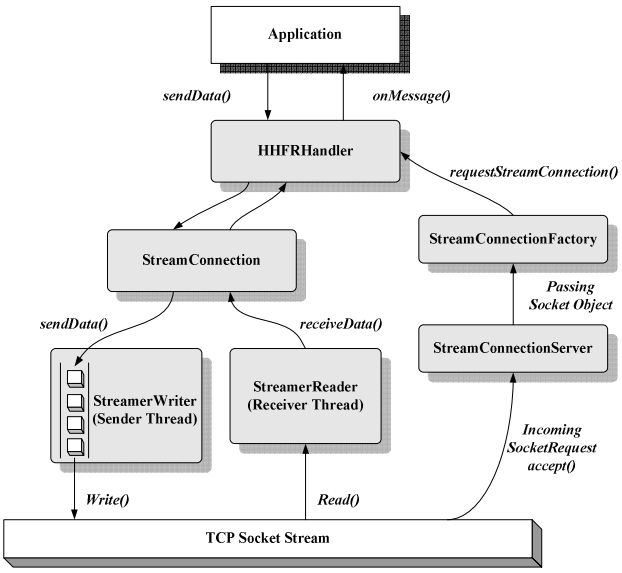

Figure. 5.6. High Performance Communication Channel Layer Diagram of TCP Receptor

streaming them, message exchanges overcome the wireless network problems of high latency and slow connection. Especially in flexible representation, we shorten the message transit time and reduce the bandwidth usage.

Data streaming has a rich research background which we discussed in the background chapter and our architecture design chapter. So we describe only our module structure and queuing approach here.

4.1 High Performance Communication Channel

The high performance communication channel of the HHFR Prototype provides an alternative to the default HTTP communication method that is asynchronous and optimized. As described, the negotiation response from the service must contain the

endpoint address (IP and port number). The second communication channel[38] is initiated by the service client.

The high performance communication channel layer for the TCP receptor is shown in Figure 5.6. On the service provider, `StreamConnectionFactory` waits for an incoming connection on a server socket and creates a `StreamConnection` that holds all streaming related classes such as a `StreamReader`, `StreamWriter`, and `Streamer`. The data that an application attempts to send is queued in a `StreamWriter`. The path that data takes includes `HHFRHandler` and `StreamConnection`. The received data follows the opposite path and is delivered to the `onMessage)` method.

4.2 Queuing in the Sender Thread

A sender thread (`StreamWriter`) uses a queue to decouple the message processing performance from the network performance. This applies to both J2SE and J2ME runtime systems, though the J2ME runtime gets more benefit because mobile wireless connections often have a narrow bandwidth.

Consider an example: without a queue, the mobile application is connected to a slow connection, such as a 2.5G cellular connection (expected downloading speed: 56 kbit/s) or wireless modem (expected downloading speed: around 14.4Kbps). Transmitting one big message will block a sender thread and this may degrade performance. Even a faster connection in a 3G network or conventional wired network could suffer the same problem – a 3G capable device is usually equipped with a faster processor and a larger

[38] The first communication in this context is a conventional SOAP communication.

memory space, but the performance difference between the message production and the transmission remains similar.

The introduction of a queue in the sender thread adds more asynchronous capability to the message transport. It decouples message processing and transmission. A sender thread receives a message packet from the message- producing thread and puts it in the queue. The sender thread then dequeues the next available message packet and writes to the network stream. The sender thread waits through a `wait()` method if there is no available message packet. This process works especially well for narrow bandwidth mobile connections where one big message packet might clog the transmission as in the example above. The extra buffering required in this method is a tradeoff because it consumes memory. However, between these two scarce resources, bandwidth and memory, the former is a bigger constraint. The physical memory limitation is currently less of a problem than bandwidth limitation because of the introduction of large capacity flash memories. These are, in fact, widely used because they create a huge advantage with little complexity.

Chapter 6.

Prototype Evaluation and Discussion

In this chapter, we present detailed performance benchmark results and observations of two benchmark applications using the HHFR implementation. The main purpose of benchmarking is to verify the expected performance saving potential of the HHFR architecture design. This chapter includes a complete description of the benchmarking approach and presents the observed measurements.

1 Benchmark Applications

We investigated the HHFR architecture's novel approach to increase the performance of mobile Web Service communications by using the alternative messaging paradigm rather than the conventional SOAP-based communication. This gives a Web Service application the two major advantages we discussed in previous chapters. First, the application can avoid data conversion to and from text format and uses an in-memory representation format to send and receive. Second, the application sends messages in a

stream fashion. Even though the SOAP specification doesn't require using any specific transport protocol, the currently available implementations for mobile Web Services use HTTP because of its pervasiveness in the World Wide Web environment and its overall popularity[39]. But in comparison to wired connections, HTTP produces higher latency over wireless connections[40]. In this situation, streaming enables faster and asynchronous message shipping. Therefore, we expect increased message delivery performance since the request message shipping is not tied to response message shipping (e.g. in a two-way conversation when the application sends a series of messages). This means the device can send the next request message without waiting for a response message from the previous request. Ultimately, this message streaming utilizes a big logical pipe[41] between an application and a service. Finally, the use of the WS-Context service as a meta-data repository, where the application and HHFR architecture can store redundant or unchanging data, allows an application to save bandwidth by reducing the message size.

The two benchmark applications we use in this chapter illustrate the first two advantages: data-conversion saving and latency-saving by streaming messages. While there are usage cases of more realistic applications such as an image capture application[42] [44], we chose these two benchmark applications because they simply and clearly show the performance savings of the HHFR architecture. We will discuss savings and performance gains from the third advantage in the next chapter.

---

[39] HTTP is the most popular protocol in World Wide Web. It is not limited by the existence of a firewall and its two-way conversation gives reliable data-transmission to applications and servers.
[40] We focus on cellular networks rather than including wireless LAN networks.
[41] The product of delay and bandwidth (delay × bandwidth) [3]
[42] A simple experiment we conducted connecting mobile devices with Global MMCS [125]. Using embedded camera in Nokia 3650 [126] and Treo 600 [127], the image capture application on the mobile devices sends captured images to a servlet over HTTP connections. The servlet application converts images into video stream and feeds it into GlobalMMCS session

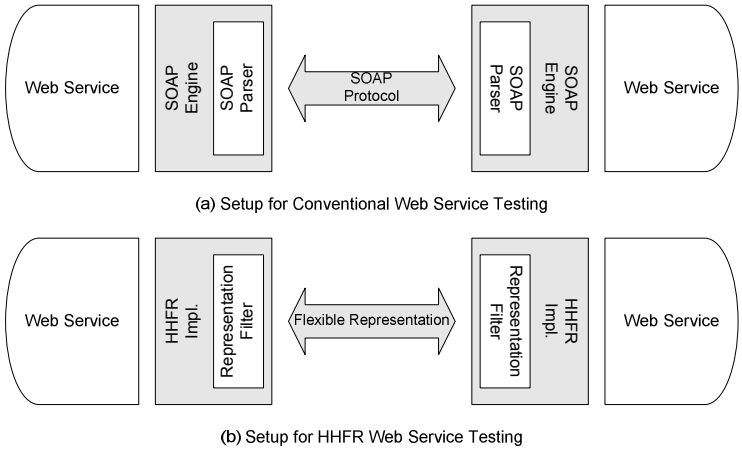

Figure. 6.1 Abstract Comparison Between the Conventional Web Service and Web Service Using the HHFR in the Benchmark Testing

As depicted in Figure 6.1 (a), the conventional Web Service services and their client applications utilize a SOAP Parser and exchange messages in the SOAP format. Figure 6.1(b) shows a symmetrical abstract view of a Web Service application using the HHFR. The HHFR implementation replaces the SOAP Parser role and provides capability to use flexible representations, and the messages in the communication are then presented in the preferred representation, which was decided upon by both parties in the negotiation stage.

The two benchmark applications we've tested represent processing in different data domains. The objective of the string concatenation service is benchmarking the prototype for a pure-text data domain; the objective of the floating point number addition is to benchmarking the prototype for a float data domain. For the float data domain, the service process of the conventional Web Services framework includes a float-to-text conversion that consumes many processor cycles. Our benchmark tests focuses on the performance effect of the HHFR runtime system. We tested various situations by changing the parameters of the performance model, which we describe in the following section. For instance, the number of messages in a stream and the size of the message

array are two parameters we varied during the tests. We compared the results against the results of the conventional SOAP-based benchmark applications with the same parameters.

1.1 String Concatenation Service

The first application is a string concatenation service. In the benchmark test, the service client creates a message that contains an array of words, with each word being eight characters long. Throughout the tests, we change the size of the array (the number of words in the array). When the service receives the message, it retrieves all the words from the array and concatenates them. It then creates a message with a single concatenated string and sends it back to the client. The following fragment of Java program shows how to create a test array of four words[43]:

```
private String[] voca =
            {"googling", "headache", "illusion", "nomadize"};
Vector v = new Vector();
for (int i = 0; i < size; i++) {
   v.addElement(voca[i]);
}
...Send Vector v to HHFRHandler object...
```

The test scenario for the application is the same for both the HHFR framework and the conventional SOAP-based application, except step 2 and step 3 of the HHFR framework test scenario do not have to be tied in order. Since the HHFR framework uses

[43] To help the reader understand the procedure, we use a Vector object as a data container. An actual benchmark test application uses a `cgl.hhms.Queue` object, which is a simple custom data container.

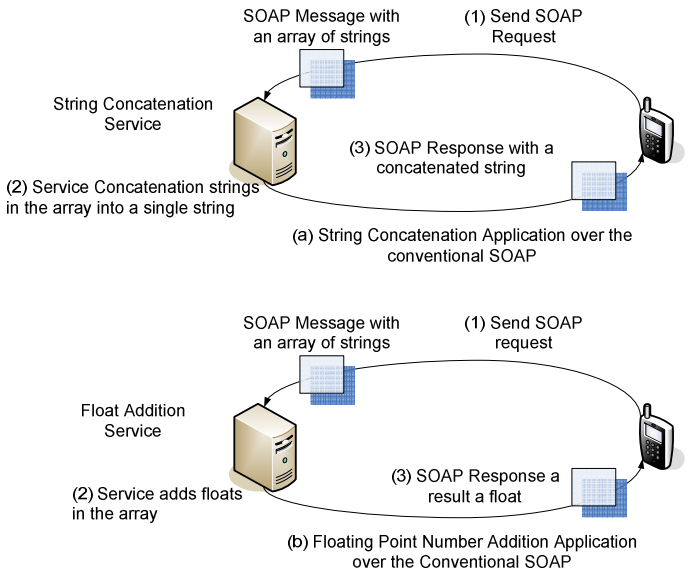

Figure. 6.2 Overviews of Interactions Between Web Services and Clients in a SOAP Test Scenario

asynchronous message streaming, the client isn't required to receive a response before sending the next request. Figure 6.2 shows an overview of the interactions between ordinary Web Service clients and services in our test scenario. It is summarized as follows:

1. A service client prepares a message with a given array size.
2. It sends one message to a service provider.
3. The service provider processes the message and returns a result in a message to the client.
4. Repeat step 1-3 for each message.

The test is done on a stream basis. We define a stream as a series of messages with the same array size. A tester who operates a mobile device inputs a number of messages

and the array size of the message. For example, since the service returns a concatenated string to each message, if the given stream consists of five messages, there are five requests and five responses. For any given stream, the total time to transact five requests and responses is measured. We also use an identical measurement scheme for the benchmark test of the SOAP-based application. A measured Round Trip Time (RTT) includes communication set-up delay, propagation delay, and concatenation processing time. It also includes array and message generation time. For testing the HHFR architecture we also measure a negotiation delay.

1.2 Floating Point Number Adding Service

The second application is a floating point number addition service. In the benchmark test, the service client creates a message that contains an array of randomly generated floating point numbers. Like the string test, we change two parameters during testing the size of the array and the number of messages in a stream. The service adds all the floating point numbers in the array of the message and returns a summation to the client. The following fragment Java shows how a floating point number array in tests was created for the tests.

```
for(int i = 0; i < numberOfMessage; k++) {
  Vector v = new Vector();
  Random rand = new Random();

  for (int i = 0; i < sizeOfMessage; i++) {
    float f = rand.nextFloat();
    v.add(new Float(f));
```

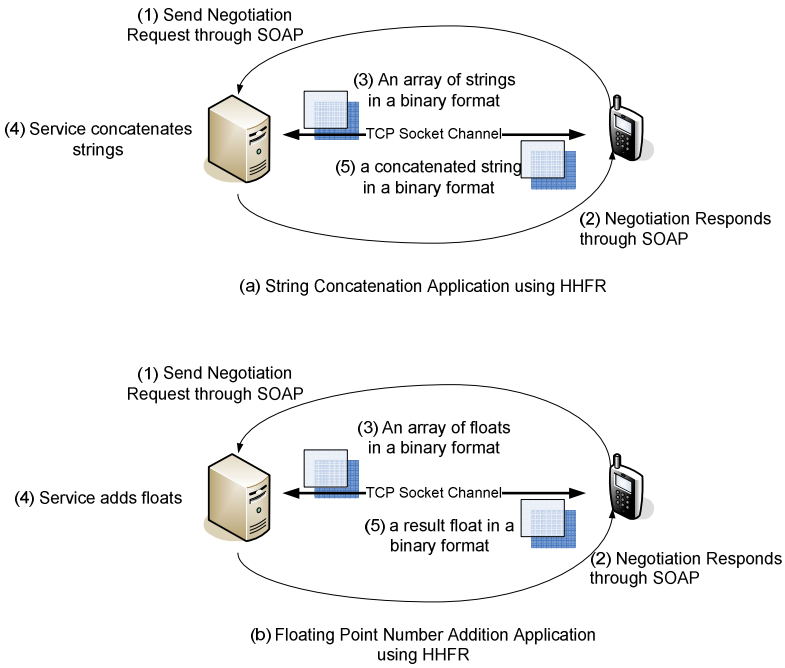

Figure. 6.3 Overviews of Corresponding Interactions Between HHFR Participants in a HHFR Test Scenario

```
    }
    ... Send Vector v to HHFRHandler Object ...
}
```

The test scenario for the application is the same for both the HHFR framework and the conventional SOAP-based application, except step 2 and step 3 of the HHFR framework test scenario do not have to be tied in order. The Figure 6.3 depicts an overview of interactions between HHFR benchmark participants in respective applications. The summary of the floating point number addition scenario is the same as for the string concatenation service, which is summarized as follows:

1. A service client prepares a message with a given array size.
2. It sends one message to a service provider.

3. The service provider processes the message and returns result message to the client.
4. Repeat step 1-3 for each message.

2 Performance Cost Analyze Modeling

In this section, we present the system models and examine the analytic cost models for both a) applications that use the HHFR prototype implementation, and b) applications which use the conventional SOAP-based Web Service communication.

To evaluate the cost models for different systems, we assume the following basic system parameter to analyze the cost.

- t_1 : time per message in a HHFR performance model
- t_2 : time per message in a conventional SOAP performance model
- O_a : overhead for accessing the Context-store Service
- O_b : overhead for negotiation
- O_c : overhead for designing the Simple_DFDL document.
- C_{hhfr} : total time for finishing stream of the HHFR
- C_{soap} : total time for finishing stream of the conventional SOAP framework

In this analysis, we assume that the cost per message (t_1 and t_2) is time. O_a, an overhead for accessing the Context-store, is a time from the start of sending a request SOAP message to the end of receiving a response SOAP message. Likewise, O_b, an overhead for a negotiation, is a time for finishing a negotiation stage, which is from the start of sending a negotiation request to the end of receiving a negotiation response. O_c is

an overhead for designing Simple_DFDL document, which can be generated automatically. For example, once an API gets a WSDL document from a given Web Service as input, the library generates a Simple_DFDL document.

With our system parameter assumptions, we analyze the cost of a HHFR performance model and a SOAP-based Web Service model. We consider the total cost on time for finishing a message exchange. In the general case, assume we have *n* messages in a stream. The cost of message exchanges in HHFR model consists of message exchange cost (nt_1) and overheads ($O_a + O_b + O_c$).

$$C_{hhfr} = nt_1 + O_a + O_b + O_c \quad (1)$$

The cost of message exchanges in the conventional SOAP model consists of the message exchange cost (nt_2). Since the SOAP model we consider in this analysis uses a request/response-based message exchange, such as a HTTP communication model, a time for each message exchange is independent of the others, and there is no common overhead for a given message exchanges.

$$C_{soap} = nt_2 \quad (2)$$

Among the system parameters, we assume cost parameters, t_1 and t_2, and a schema designing overhead, O_c depend on message size to the size of message[44]. The Context-store accessing overhead, O_a, can also be dependant on the size of message. It is likely that we will have a larger overhead when we have a bigger message, since the fragment of the message which is unchanging or redundant tends to be large when the message is

[44] Dependant or response variables

large. In this analysis, O_c, the overhead for designing the Simple_DFDL document is defined and counted as one of three overheads. But it is given a zero value for the current framework because the process of designing a Simple_DFDL is not automated, but rather implemented in an ad-hoc method.

3 Performance Evaluation Configuration

We performed the tests on two machines. Table 6.1 contains the summary of their system configurations.

Table 6.5 Summary of Machine Configurations

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

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

Service-provider applications, the HHFR framework, and the Axis Web Service container run on a Linux machine (gridfarm8). Service-client applications run on a Treo600 Smart-phone [127] equipped with CLDC 1.1 and MIDP 2.0.

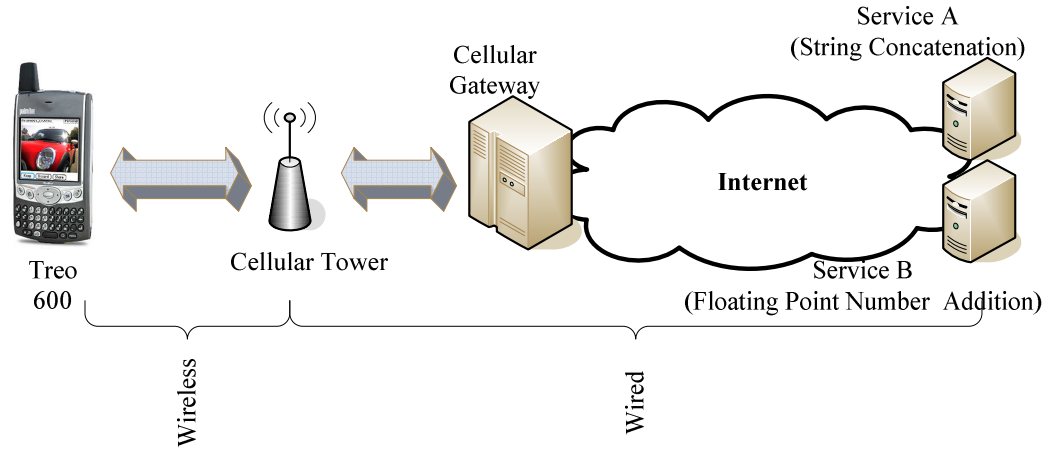

Figure. 6.4. Abstract Overview of the Connection Setup Between Treo600 and Service Machine

3.1 Connection Setup

Connections between a mobile device, such as the Treo600, and its service providers are logical process-to-process channels, which are partly wireless and partly wired, as depicted in Figure 6.4. The service bearer of the Treo600 is Sprint and its wireless service type is the CDMA-based PCS Vision[45]. The wireless part of the channel goes from the mobile client[46] to a radio-tower[47] (or a base-station). The rest of the channel, including the connection between the gateway machine of the service-bearer and the host machine running our benchmark applications, is wired. The connection is made as follows: the mobile station connects to the closest radio tower wirelessly. If the mobile station moves to out of the area of the cell, a Mobile Switching Center (MSC) performs a handover mechanism so that the mobile station moves seamlessly from the original cell to an adjacent one [4]. Once the base-station connects to the device, it makes a logical connection to the service provider, which is in our case is the string concatenation service

[45] A second generation cellular technology
[46] The term *mobile client* is interchangeable with the term *mobile station* in many cases.
[47] The term *base-station* is a more familiar term in many cellular network papers than the term *radio tower*.

or the floating point number addition service, using the service bearer's gateway machine as a portal point.

The connection bandwidth is governed by the slowest connection, which in practice will be the cellular network. As we discussed in Chapter 3, a second generation cellular network, which is the base technology of the Sprint service bearer, provides at most 14.4Kbps. We evaluate the prototype system using the TCP/IP socket connection as the high-performance communication channel transport layer. In this case, we expect messages to be delivered in order and that the TCP connection guarantees delivery since we assume there is no link or node failure during the test.

3.2 Measurement Methodology

We measure the total message exchange time of both the HHFR and the conventional SOAP and compare them to evaluate the performances of the two systems for our particular applications. We define the total message exchange time as the summation of Round Trip times (RTTs) of the individual message transactions and any overheads. We measure total message exchange times for 100 ~ 400 repetitions varying the number of messages per stream (the first independent variable). We also vary the size of messages (the second independent variable). The resolution of the Java timer, `MIDP 2.0/CLDC 1.1`'s `System.currentTimeMillis()`, used is 10 milliseconds, and it produces a reasonable number of significant numbers for testing the high latency wireless connection, where each transaction ranges from 1,000 ~ 15,000 milliseconds.

Table 6.2 Negotiation Times Summary

Application	Average±error (sec)	Stddev (sec)
String Concatenation	5.133±0.036	0.825
Floating Point Number Addition	5.127±0.029	0.676

Table 6.3 Context-store Access Times Summary

	Set 1 (sec)	Set 2 (sec)	Set 3 (sec)	Set 4 (sec)	Set 5 (sec)
Average±error (sec)	4.194±0.083	4.197±0.093	4.177±0.123	4.028±0.066	4.036±0.097
Stddev (sec)	0.457	0.511	0.676	0.363	0.530

4 Observed Performance Measurements and Performance Analysis

In this section, we present detailed performance measurements and analyze them. The values of the t_1, t_2, and O_b parameters in our performance model can be analyzed from the measurements. We use a value for the O_a parameter in Table 6.3 from the Context-store performance evaluation, which we will present in the following chapter.

4.1 Observed Measurements of The Benchmark Applications

Table 6.2 shows the average negotiation times, O_b, of the benchmark applications. O_b times were collected during the same experiments which measured the total message exchange times, both of which are needed to analyze t_1 and t_2. The measurement test code, inserted for the benchmark applications, measures both the negotiation time and the total message exchange times. The values of both negotiation and Context-store accessing overheads are similar. We assume the similarity comes from the fact that they use the same conventional Web Service transaction, which is a HTTP `call()` of the kSOAP. So

we differentiate the overhead parameters of mobile environments and conventional Web Service environments as O(mobile) and O(ws)[48].

A total message exchange time of the HHFR, i.e. C_{hhfr}, consists of a time for message exchanges, a negotiation overhead, and a Context-store accessing overhead. On the other hand, a time for message exchanges in the conventional SOAP is equivalent to a total message exchange time, i.e. C_{soap}, since the message exchange doesn't involve any negotiation or Context-store access. Table 6.4 and 6.5 show the times for message exchanges of the benchmark applications as the number of messages per stream in the HHFR varies. The total message exchange time of the applications in the Apache Axis with kSOAP is shown in Table 6.6 and 6.7. There could be a bandwidth difference between wireless connections at different times of day because the traffic to the cellular gateway may vary. But the differences are nominal compared to the total measurement. For instance, one of the measurements of the total message exchange time during the most crowded time is 4.563 second on average, and other measurements with the same control parameter values made early in the morning or late in the night are 4.451 seconds on average, which is only 2.5% faster than the measurement from the most crowded time.

4.2 Performance Analysis

Figures from Figure 6.5 to Figure 6.12 show the total message exchange times of the application in the HHFR as the number of messages varied per stream compared with the times of the application using the conventional Web Service framework i.e. Apache Axis with kSOAP. The total message exchange times of the HHFR in the figures includes the

[48] For instance, O_a(ws), which was measured in experiments outside the scope of our dissertation, is roughly 20 milliseconds.

Table 6.4 Summary of the Total Times to Finish Streams of the HHFR : String Concatenation Application

Message Size	Number of Messages Per Stream					
	n = 1 (sec)		n = 2 (sec)		n = 4 (sec)	
	Ave±error	**Stddev**	**Ave±error**	**Stddev**	**Ave±error**	**Stddev**
2 Strings	1.782±0.028	0.176	1.810±0.019	0.117	4.482±0.025	0.155
4 Strings	1.804±0.018	0.111	2.039±0.036	0.227	4.668±0.028	0.180
8 Strings	1.973±0.031	0.196	2.296±0.039	0.247	4.892±0.032	0.205
16 Strings	2.128±0.019	0.118	2.680±0.016	0.103	5.383±0.024	0.151

Message Size	Number of Messages Per Stream					
	n = 8 (sec)		n = 16 (sec)		n = 32 (sec)	
	Ave±error	**Stddev**	**Ave±error**	**Stddev**	**Ave±error**	**Stddev**
2 Strings	4.897±0.043	0.272	6.333±0.064	0.406	7.745±0.083	0.524
4 Strings	5.186±0.036	0.225	7.832±0.073	0.462	9.027±0.081	0.510
8 Strings	5.814±0.056	0.352	8.406±0.083	0.523	10.653±0.085	0.535
16 Strings	7.929±0.031	0.193	10.410±0.070	0.443	14.467±0.313	1.981

Table 6.5 Summary of the Total Times to Finish Streams of the HHFR: Floats Addition Application

Message Size	Number of Messages Per Stream					
	n = 1 (sec)		n = 4 (sec)		n = 8 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**	**Ave.±error**	**Stddev**
2 Floats	1.571±0.011	0.067	4.241±0.013	0.085	4.598±0.016	0.104
4 Floats	1.688±0.022	0.141	4.280±0.017	0.106	4.687±0.019	0.119
8 Floats	1.827±0.021	0.132	4.474±0.025	0.158	5.079±0.024	0.153
16 Floats	1.991±0.031	0.198	4.672±0.028	0.175	6.521±0.056	0.356

Message Size	Number of Messages Per Stream			
	n = 16 (sec)		n = 32 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**
2 Floats	5.016±0.024	0.152	6.482±0.090	0.572
4 Floats	5.407±0.071	0.450	6.873±0.119	0.753
8 Floats	6.422±0.032	0.205	8.890±0.059	0.375
16 Floats	8.533±0.055	0.350	12.920±0.367	2.324

Table 6.6 Summary of the Total Times to Finish Streams of the Apache Axis with kSOAP : String Concatenation Application

Message Size	Number of Messages Per Stream					
	n = 1 (sec)		n = 2 (sec)		n = 4 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**	**Ave.±error**	**Stddev**
2 Strings	2.940±0.023	0.217	5.915±0.037	0.354	11.986±0.074	0.700
4 Strings	3.222±0.029	0.273	6.161±0.029	0.272	12.146±0.075	0.710
8 Strings	3.520±0.031	0.291	6.777±0.024	0.230	13.479±0.036	0.337
16 Strings	3.765±0.020	0.191	7.447±0.033	0.315	15.082±0.114	1.079

Message Size	Number of Messages Per Stream					
	n = 8 (sec)		n = 16 (sec)		n = 32 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**	**Ave.±error**	**Stddev**
2 Strings	24.244±.0129	1.227	48.191±0.261	1.568	96.108±0.325	2.391
4 Strings	25.417±0.145	1.374	50.151±0.248	1.486	100.902±0.547	4.022
8 Strings	27.483±0.108	1.020	55.199±0.283	1.701	112.679±1.127	6.761
16 Strings	30.344±0.163	1.549	60.419±1.033	6.196	126.016±1.076	6.453

Table 6.7 Summary of the Total Times to Finish Streams of the Apache Axis with kSOAP : Floats Addition Application

Message Size	Number of Messages Per Stream					
	n = 1 (sec)		n = 4 (sec)		n = 8 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**	**Ave.±error**	**Stddev**
2 Floats	3.061±0.016	0.151	11.818±0.044	0.416	22.934±0.168	1.593
4 Floats	3.071±0.023	0.218	12.283±0.075	0.712	23.555±0.151	1.434
8 Floats	3.387±0.018	0.174	13.387±0.078	0.740	27.801±0.176	1.674
16 Floats	3.709±0.023	0.219	15.213±0.169	1.606	29.692±0.136	1.286

Message Size	Number of Messages Per Stream			
	n = 16 (sec)		n = 32 (sec)	
	Ave.±error	**Stddev**	**Ave.±error**	**Stddev**
2 Floats	48.336±0.288	1.776	94.306±0.479	2.876
4 Floats	49.279±0.251	1.547	97.410±0.513	3.080
8 Floats	56.146±0.316	1.896	107.698±0.743	4.455
16 Floats	61.331±0.855	5.129	115.493±0.593	3.560

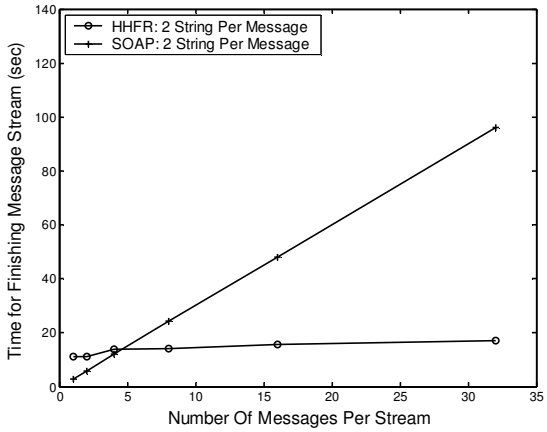

Figure. 6.5. Total Times to Finish Streams of The String Concatenation Application (2 Strings Per Message)

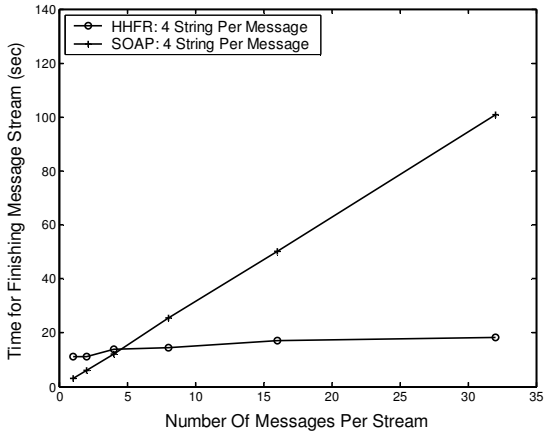

Figure. 6.6. Total Times to Finish Streams of The String Concatenation Application (4 Strings Per Message)

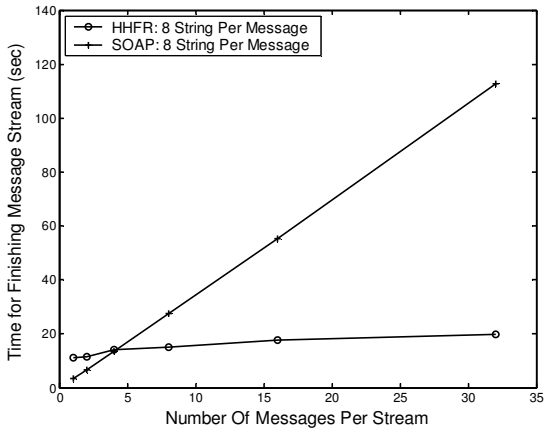

Figure. 6.7. Total Times to Finish Streams of The String Concatenation Application (8 Strings Per Message)

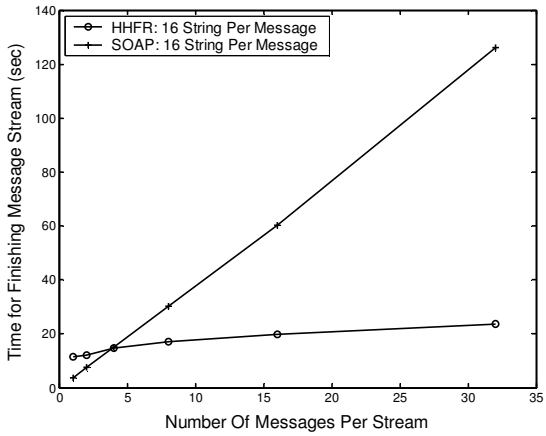

Figure. 6.8. Total Times to Finish Streams of The String Concatenation Application (16 Strings Per Message)

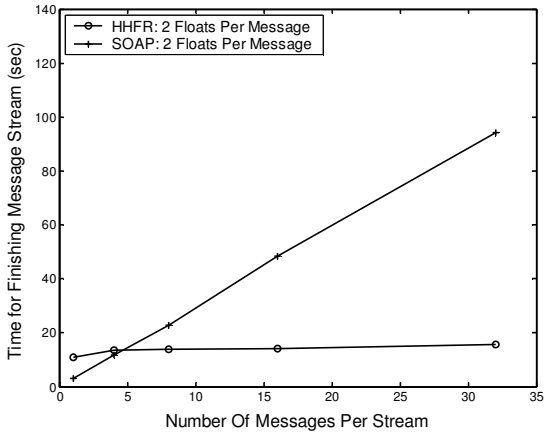

Figure. 6.9. Total Times to Finish Streams of The Floats Addition Application (2 Floats Per Message)

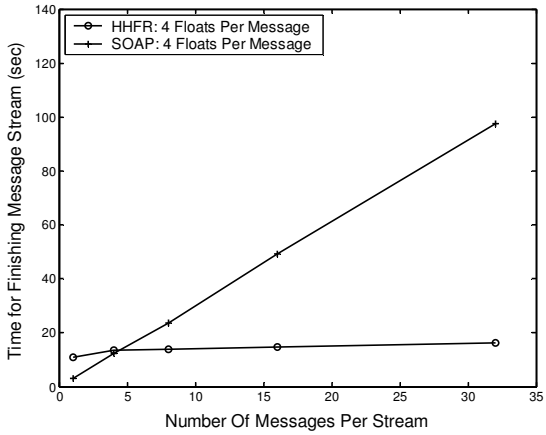

Figure. 6.10. Total Times to Finish Streams of The Floats Addition Application (4 Floats Per Message)

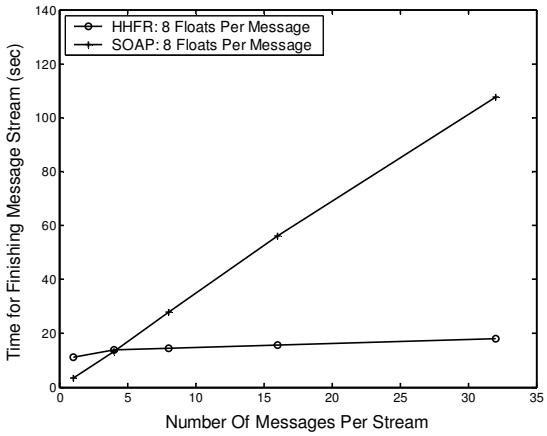

Figure. 6.11. Total Times to Finish Streams of The Floats Addition Application (8 Floats Per Message)

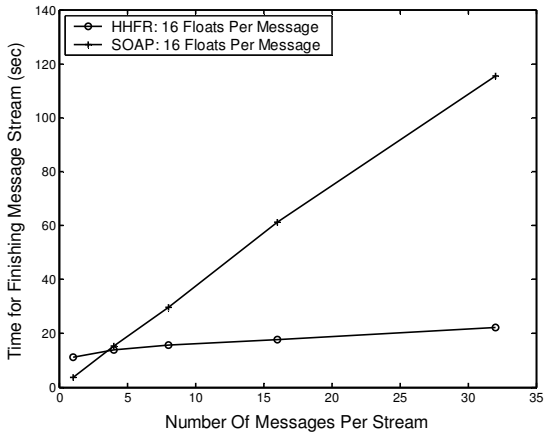

Figure. 6.12. Total Times to Finish Streams of The Floats Addition Application (16 Floats Per Message)

combined time of the message exchange with the negotiation overhead and the Context-store accessing overhead[49] presented in the Table 6.2 and Table 6.3[50].

The figures show the total message exchange times of the applications using the conventional Web Service framework (i.e. C_{soap}) increase much faster than the time of the applications using HHFR (i.e. C_{hhfr}) as the number of messages per stream increases past the breakeven points. The primary reason of the C_{soap}'s rapid increase is the high latency of the HTTP-based communication used in C_{soap} tests. As we discussed in previous chapters, HTTP isn't the one and only or a mandatory transport protocol for Web Service communication. But, it is most popular transport protocol in mobile computing and the most of mobile Web Service implementations uses HTTP as their transport protocol. So the HTTP-based results show valid performance comparisons. The secondary reason of the C_{soap}'s rapid increase is the SOAP serialization overhead. Recall that serializing in a SOAP message includes structuring data and data-conversion, and parsing the SOAP message includes de-structuring and data-conversion. This causes the applications to take more time to finish a message exchange using the conventional Web Service framework. In the stream with one message, it takes longer for the message exchanges in the conventional Web Service framework, C_{soap}, than it does for the message exchanges using the HHFR. This is shown in Tables 6.4 through 6.7. The differences show the SOAP serialization and de-serialization overheads as well as some of transport performance difference between TCP/IP and HTTP.

Although the process of the message exchange in the HHFR outperforms the one in the conventional Web Service framework (i.e. Axis with kSOAP in this benchmark),

[49] We assume there is only one Context-store accession in this particular test.
[50] This is based on our performance model, which is presented in Section 4.1.

there exist in non-zero locations one breakeven point for each parameter change. The breakeven point is that point below which the conventional Web Service framework outperforms the HHFR. The initial negotiation overheads (i.e. O_b) and having zero or more Context-store accessions (i.e. O_a) causes breakeven points to exist to the right of the 'one message per stream' point on the x-axis. In our benchmark tests, the breakeven points range from three to five messages per stream. We demonstrate how to calculate a breakeven point (n_{be}) using our performance model.

The demonstration uses values of the floating point addition application test with four floats in a message. First, we take $t_1 = 0.135$ and $t_2 = 3.051$ from the MatLab Basic Fitting tools [128]. Then, using overhead values from the tables ($O_a = 4.126$ and $O_b = 5.127$) and assuming $O_c = 0$, we calculate a breakeven point where $C_{hhf} = C_{soap}$.

$$t_1 n + O_a + O_b + O_c = t_2 n$$

$$0.135n + 4.126 + 5.127 + 0 = 3.051n$$

$$n_{be} = 3.17$$

Thus, if we have more than 4 messages per stream in the floating point addition application, the HHFR as a communication framework performs better than the conventional framework, i.e. Apache Axis with kSOAP.

Similarly, the breakeven point of the string concatenation application with two strings in a message is calculated as:

$$t_1 = 0.181,\ t_2 = 3.010,\ O_a = 4.126,\ O_b = 5.133,\ O_c = 0$$

$$0.181n + 4.126 + 5.133 + 0 = 3.010n$$

$$n_{be} = 3.27$$

Similarly, if we have more than 4 messages per stream in this application, the HHFR as a communication framework performs better than the conventional framework.

5 HTTP Persistent Connection in Mobile Computing Environment

In this section we discuss the use of a persistent connection in the mobile computing environment. The TCP setup overhead is a significant overhead in mobile communication, and it could take two to four seconds using a high latency cellular network. Since the TCP connection is a basic layer of HTTP, it suffers the same overheads when it sets up the connection. To ease this condition, the persistent connection HTTP 1.1 combined with pipelining can be used for a service client in constrained environments.

A persistent connection enables a single TCP connection to be maintained over several request/response pairs. This is a mandatory feature of HTTP 1.1 implementations, so an HTTP 1.1 client can assume that the server and the proxy in the channel would maintain a persistent connection. But there are several situations where a persistent connection doesn't work in cellular networks. As described in the HTTP specification [80], a client, server, or proxy can close the connection at any time for any reason. For instance, a participant may have a very short timeout period, which avoids an unnecessary resource drain by disconnecting the mobile client. The intermittent nature of the cellular network makes this choice sound practical. Alternatively, a gateway or a

proxy server in the cellular service provider's network may disallow a persistent connection in order to ensure efficient network resource management.

Another requirement, pipelining, is not a mandatory feature of HTTP 1.1. It is mostly used for browser style applications, which fetch many small pictures or objects from the same target. The specification states:

"A client that supports persistent connections MAY "pipeline" its request (i.e. send multiple requests without waiting for each response). [80]"

Many HTTP implementations in mobile devices do not support pipelining features, or they may only allow very few requests over a persistent connection. Even the same MIDP implementations could behave differently from one implementation to another. So a design based on using a persistent connection and pipelining is not portable across mobile computing environments.

6 Summary

In this chapter, we presented detailed performance evaluations. Through two benchmark applications, we have demonstrated that the HHFR framework provides more efficient communications for applications which exchange messages in a stream fashion. We found that the negotiation overhead and the Context-store accession overhead are the primary causes of the breakeven points, which indicate that the HHFR is more efficient for longer message exchanges. The results in this chapter show the effect of both message

streaming and the flexible representation of the message. We present the effect of the Context-store in the following chapter.

Chapter 7.

Optimizing Web Service Messaging Using a Context Store for Static Data

In the previous chapters, we have focused on the efficiency and flexibility of the HHFR architecture with respect to the prototype implementation, and have described the design and implementation issues of the prototype. In this chapter we concentrate on the Context-store implementation of the HHFR architecture design. As discussed, the Context-store in which redundant / unchanging SOAP message parts are stored provides database semantics to our HHFR design.

Adapting the WS-Context compliant information management framework, FTHPIS [110], for the context-store of the HHFR has advantages. These advantages are following: First, obviously this allowed us to reduce the HHFR prototype development period. Second, since adaptation makes the Context-store interoperable with other FTHPIS

clients, the future HHFR design expansion is able to collaborate with any FTHPIS compatible-composite applications.

First, we describe the WS-Context compliant information service implementation of CGL. Second, we detail our design and implementation of the Context-store using the service. Then we evaluate the system performance and scalability.

1 The Overview of WS-Context Service Implementation of Community Grids Lab (CGL)

Community Grids Lab (CGL) developed the WS-Context service[51] which is based on the WS-Context specification defined by OASIS [95] as a part of Fault Tolerant High Performance Information System Project. The WS-Context implementation supports static, semi-dynamic data, and dynamic data. Combined with the extended UDDI [103] service, forms a hybrid information service that is applied to the Geographical Information System (GIS) service [129] at CGL, the Global MultiMedia Collaboration System (GlobalMMCS) [125], or the Sensor Grid.

The researches at CGL are developing an Information Service that satisfies the information requirements of service oriented architectures (SOA). In summary, the requirements are: the service should support a dynamic collection of services, it should be expandable in scale, and it provides a uniform interface for metadata access. The service is based on two Web Service Specifications, WS-Context and Universal Description, Discovery and Integration (UDDI). The approach used is to support information in

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

dynamically assembled set of Grids/Web Services to solve a particular problem is given in the following steps:

1. WS-Context system receives querying or publishing metadata requests.
2. The system separates dynamic and static portions of the metadata based on predefined categories. For example, two locations of a service and throughput are categorized as static data while stream related identifiers are dynamic data.
   2.1. UDDI handles the static portion of metadata.
   2.2. The system itself handles the dynamic portion of metadata.

2 WS-Context service as a Context-store of the HHFR and its usage.

Integrating a Web Service Based Information System (i.e. WS-Context compliant information service of FTHPIS) with HHFR brings a dependency on SOAP-Java binding on the Apache Axis library. As we overviewed in Chapter 3, the Axis version for the J2ME environment 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 porting the code into J2ME. Unfortunately, replacing J2SE APIs with J2ME APIs isn't possible. So we must find an alternative solution.

The solution to the problem includes a direct serialization of the 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[52] library for those

[52] kSOAP is described in Chapter 3 in detail.

processes. SOAP serialization using the kSOAP library needs an ad-hoc method to integrate with the Information Service while SOAP parsing is straightforward. Because Axis SOAP-java binding is not available for the J2ME environment, we focus on SOAP messages generated by the WS-Context client using Axis. The Axis-Java binding adds a hierarchically referenced element to the structure if the binding process meets a Java wrapper class when it serializes a SOAP message. As a result, the Axis based SOAP binding code for a WS-Context Service client generates a multi-referenced XML. Unfortunately, kSOAP doesn't support such advanced binding APIs; rather it provides more direct SOAP serialization APIs. For example, the piece of Java code on the next page will result in the XML fragment[53] in Figure 7.1(b). In both figures, the relevant code is highlighted in 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);
```

Figure 7.1(a) shows the `getContext` SOAP request message of the conventional WS-Context client using Axis and Figure 7.2(b) shows the flattened SOAP request message produced by the mobile WS-Context client using kSOAP.

[53] To help readers, it should be noted that the XML fragment is presented to illustrate to simplification process and it is not the XML fragment which is used in the performance evaluation.

```
<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:getContexts
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xmlns:ns1="http://wsctx_service.WSCTX.services.axis.cgl">
         <body href="#id0"/>
      </ns1:getContexts>
      <multiRef id="id0" soapenc:root="0"
      soapenv:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
      xsi:type="ns2:GetContexts"
      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>
```

Figure. 7.1(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:getContexts 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:getContexts>
   </v:Body>
</v:Envelope>
```

Figure. 7.1(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.

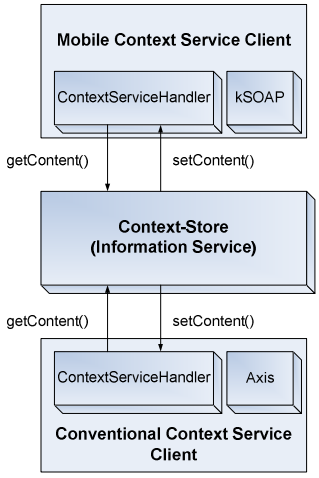

Figure. 7.2. Mobile and Conventional Context Service Clients

As depicted in Figure 7.2, the two primary WS-Context related functionalities of Information Services are the `getContext()` and `setContext()` 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 stream, so they can happen at any time when the HHFR runtime or the HHFR client service needs to create, update, or retrieve context in the Context-store (Information service). Thus, the following Java program 1) creates a `ContextServiceHandler` object with the Context Service URl and the service (implementation) version, 2) stores a given context of any type paired with unique identifier, and 3) retrieves the context. The `ContextServiceHandler` object is a wrapper class and provides `getContext()` and `setContext()` 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…
}
```

`getContext()` and `setContext()` methods throw `java.lang.InterruptedException` since the handler runs as a Thread. Running as a Thread can avoid a possible deadlock situation, which could occur if the network fails or there is an operational error.

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

3 Performance Evaluation.

The goal of this performance evaluation is to demonstrate the effect of using the Context-store in the HHFR communication framework. 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

Table 7.1 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.

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 7.1 shows a summary of our measurements.

3.1 Performance Evaluation Model

In this section, we present the system model and following system parameters to analyze the performance and scalability of our approach. 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

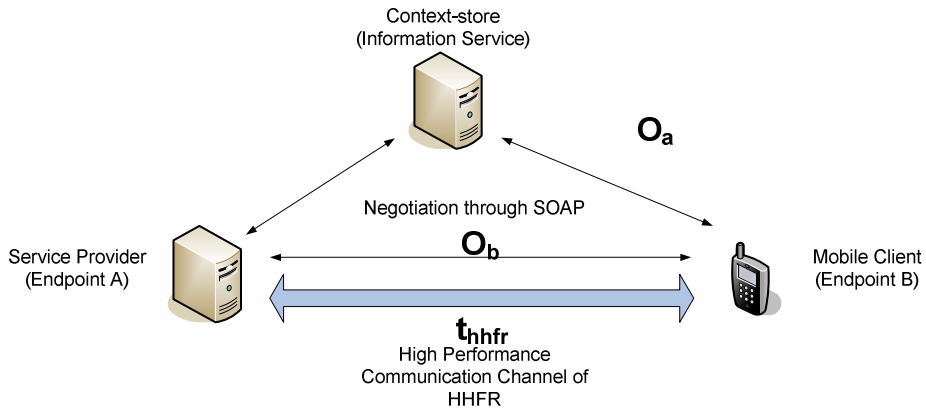

Figure. 7.3. System Parameters

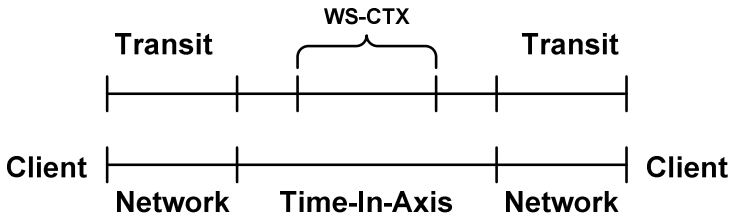

Figure. 7.4. System Parameters with Time Frame

- 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. Figure 7.3 and 7.4 show parameters on the illustrated system model.

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

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:

$$3N/T_{stream} \approx 1/ T_{time\text{-}in\text{-}server}$$

$$N \approx T_{stream} / (3 * T_{time\text{-}in\text{-}server})$$

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} which 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 1.2 Beta 3 with an Axis data binding to measure message processing overhead (i.e. $T_{time\text{-}in\text{-}server}$). However, we expect a better performance (i.e. smaller $T_{time\text{-}in\text{-}server}$) if we use better performing Web Service container with other data bindings such as XMLBeans [139] or JiBX [140]. Then the server will be able to support more simultaneous streams. Finally, the stream length 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

Table 7.2 Summary of Machine Configurations

Context-Store Client: ural.ucs.indiana.edu	
Processor	Intel® Pentium™ 4CPU (3.40GHz)
RAM	1GB total
Bandwidth	100Mbps
Operating System	Microsoft Windows XP Professional Version 2002 SP2
Java Version	Java 2 platform, Standard Edition Version 1.5.0-06

spread over the stream length, thus the longer stream length is, the more simultaneous streams can be supported.

3.2 Evaluation Configuration

We used three machines. Grid Farm 6 was used as a service provider, and it has the same configuration as Grid Farm 8 from the experiments in the Chapter 6. Treo 600 was used again for a service client. Table 6.1 in Chapter 6 contains the configurations of the two machines. In addition to these two previously presented machines, a Windows XP machine (ural.ucs.indian.edu) was used in the scalability test. The configuration of the ural machine is shown in Table 7.2.

3.3 Experiment 1: Context-store Access Time

In this section, we present the time measurements[54] to access a Context-store. To measure the time, we used the `setContext()` operation[55] of the FTHPIS. Similar to the measurement scenario for the negotiation overhead, we measured Round Trip Times of the Context-store accessing transactions. A mobile client sends a sample SOAP message

[54] The overhead is denoted as O_b in our performance model in Chapter 6.
[55] We choose the `setContext` operation as an example. Similar performance evaluation can be made for the `getContext` operation.

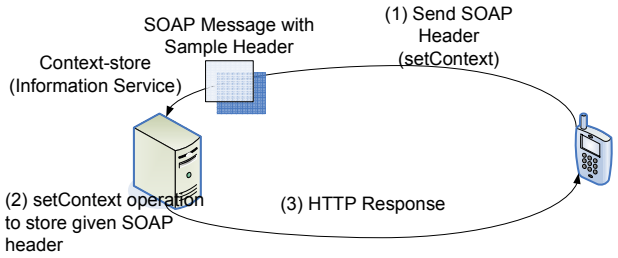

Figure. 7.5. Set Up for Measuring Context-store Accessing Overhead

```
<?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>
```

Figure. 7.6. WS-RM Message Example For Context-store Access Measurement

with Web Service Reliable Messaging (WS-RM) and the Information Service responds back. The experiment is illustrated in Figure 7.5. The size of the headers used in the test, which is shown in Figure 7.6, 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 7.3 shows the average values of the collected data.

Table 7.3 Summary of the Measured Context-store Accessing Overhead [56]

	Ave.±error	Stddev
Overhead (sec)	4.127±0.042	0.516

3.4 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 we discussed in Chapter 4 and 5, 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 [136]. The WS-Addressing Specification defines transport neutral mechanisms to address Web Services and messages[57]. It defines two constructs which 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

[56] This table contains a summary of the Context-store accession performance, which is also presented in Table 6.3.
[57] The definition is from the WS-Addressing Specification [136]

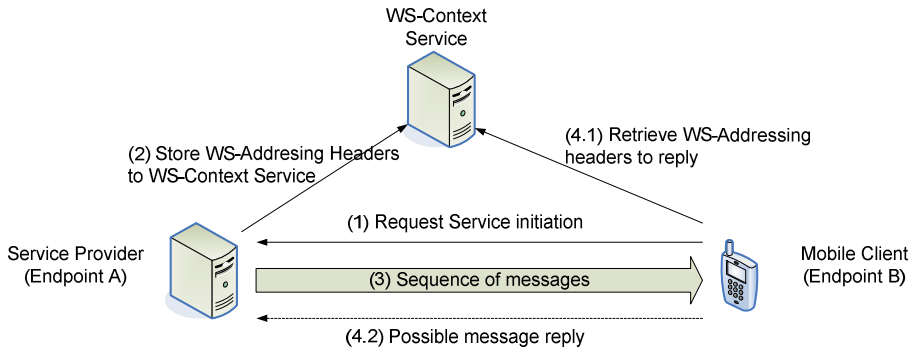

Figure. 7.7. Scenario for WS-Addressing Example

```
<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>
```

Figure. 7.8. Sample SOAP Message for WS-Addressing Example

references convey information which 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

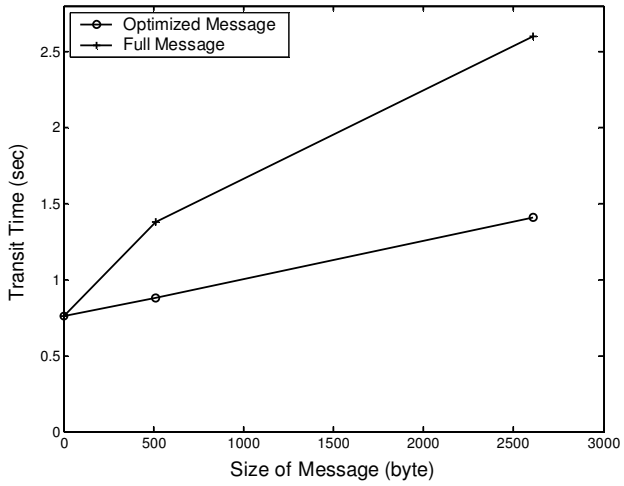

Figure. 7.9. Round Trip Time of Optimized Message Exchange through the HHFR High Performance Channel Compared With Full Message Exchange

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 which 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 Figure 7.7, and Figure 7.8 shows 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.

Figure 7.9 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

Table 7.3 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

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 which 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 which 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 7.3.

3.5 Experiment 3: Scalability of the Context-store

In addition to these two tests (i.e. the test which measures the accession overhead and the test which 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[58] (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 7.4. Since T_{wsctx} is less than one millisecond, figure 7.10

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

Table 7.4 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

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.

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 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:

$$N \approx 600 / (3 * 0.035)$$

$$N \approx 5700$$

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.

4 Discussion

In this chapter, we present the detailed implementation of the Context-store in the HHFR, which is one of three key features of the architecture. We argue that the right way

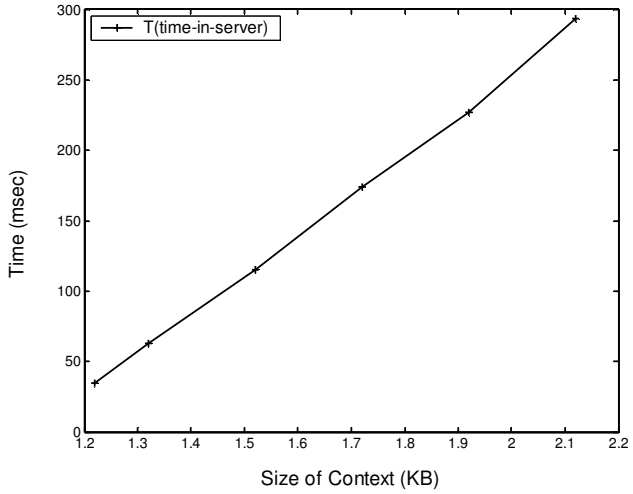

Figure. 7.10. Time to Finish Context-store Request Message Processing (i.e.,$T_{time\text{-}in\text{-}axis}$)

to address the message size reduction issue in the mobile computing is to save redundant or unchanging SOAP message parts (i.e. metadata) to the Context-store and retrieve them when they are needed. This approach can increase the efficiency of message exchange when messages in the stream share many SOAP parts. We design the Context-store based on the WS-Context Specification and adopt the existing Information Service implementation (FTHPIS) of CGL.

We present the effectiveness of using the Context-store in HHFR framework by comparing the transit time of the optimized message (i.e. after storage of redundant or unchanging SOAP parts) and the transit time of the full SOAP message. The example message is chosen from the WS-Addressing Specification. The empirical result shows that the size of message is reduced on average 17% (83% gains) and transit time is saved by 41%. We also measured the Context-store accession overheads, which was used in the performance model analysis from Chapter 6, and the scalability of the Context-store to

verify the feasibility of using the service with multiple mobile clients (or message streams).

Chapter 8.

Future Work

In this chapter, we discuss a few open research issues and possible improvements in the HHFR implementation that we will pursue in the future.

1 SOAP Message Compression

A SOAP message compression approach is a critical for designing a high-performance Web Service framework because the approach affects communication performance, a method to handle message security, and a method to process compressed messages. Our approach to compress SOAP message is distinguishing message semantics and exchanging them through a separated high-performance channel. However, the approach is an open research question to the Web Service community and there could be better approaches for different application domains.

For the performance, the self-contained[59] approach supports individual message processing like the conventional Web Service and it can be used to more general application domains. The approach, however, sacrifices performance for encoding and decoding descriptive information. For applications which want high performance and message structures stay unchanged, non-self contained (or schema-specific parsing) approach provide more performance oriented communication / SOAP compressing method.

In our method to process message, message handlers are not changed and filters for optimized representations are exchanging messages through a second channel (i.e. separated channel). This approach is simple to implement and easy to deploy because it is pluggable to existing SOAP containers such as Axis without making a modification on the container. However, this approach requires more network resource by establishing another channel to communicate and raises questions about a security issue, i.e. a new channel mostly uses non Web Service port (port 80) with which message can not go through a firewall.

A message exchange channel can be shared with the conventional SOAP message. In this case, a SOAP message handler must understand optimized representations (e.g. a transport handler in Axis must understand how to convert the optimized representation data to `Message` Object) and both the conventional SOAP message and the optimized message will be processed through the same transport handler. A SOAP Header handler must understand optimized representations so that intermediary node can process SOAP headers.

[59] i.e. self-described

The compression approach also affects a method to handle the message security. There are different message security models and a compression mechanism works for models for which it is suitable to store and retrieve a signature and a security token. It is an open research issue in the community to build a secure Web Service model by combining pieces (i.e. Web Service specifications) into an overall system level solution.

2 Data Description Language

Our approach to distinguish between message semantics and syntax is based on our Simple_DFDL implementation which includes a language definition and 'interpret-style' stubs, a stream reader and writer. However, the Simple_DFDL implementation has limitations and restrictions which made inevitably to build the prototype in a reasonable period of the time and with a limited programming environment.

Those limitations are: first, our Simple_DFDL implementation doesn't define low level data mapping. Low level mapping is important for designing and implementing a specific system which interoperates among various programming language and hardware platforms. We suggest using fully developed DFDL for this low level mapping in the future system. As it is being developed for describing binary format file and stream, the design defines the low level layer which maps between various types of the concrete representation and the information content. Thus the use of DFDL (or replacing the Simple_DFDL implementation with DFDL) will make HHFR capable to be used among various platforms. Additionally, the use also will make it easier that HHFR become a Web Service Standard while we consider the support to DFDL.

Second, in our HHFR approach, we use a Simple_DFDL document as a sample XML document to build a message structure of messages in a stream in our HHFR approach, which hinders the full utilization of XML Schema features. For instance, in Simple_DFDL documents, complex type is restricted to have only sequence groups, not have choice groups or all groups. This is essential because parsing a Simple_DFDL document must produce a single structure. The more XML feature covers in the future Simple_DFDL implementation or DFDL which replaces the Simple_DFDL, better for application developers because applications can generate messages in more options (e.g. messages which have choice group in it).

It is still an open research question that whether a Schema document can perfectly represent the structure and type (i.e. syntax) of a XML document or not. In our approach, we focus on application domains where the message structure is not changed in a message exchange stream. However, if the message structure is changing dynamically or the message structure can only be represented with the XML Schema definition which is not supported by the Simple_DFDL, our approach is not adequate to those application domains.

3 Filters

Filters are important to transform message representations which enables an application choose any representation which is optimized to given criterion in our HHFR design. The current prototype implementation provides filters based on a basic case-based reasoning, i.e. an application preferred representation is decided in the negotiation stage and message sender node generates and sends the representation which will be consumed

on receiver side. More optimal approach for representation decision is a possible research and implementation issue. For instance, the receiver node should ask to send messages in a representation which make the size of message small then convert to the representation[60] what it need after receiving if sender and receiver exchange messages through a narrow bandwidth connection. Thus the ultimate solution should be the approach which gives receiver more freedom on representations.

4 Multiple Transport Protocol Support for High Performance Channel

The high performance channel implementation in the HHFR prototype is provided as TCP/IP socket transport. This is enough to show the effectiveness of the architecture. Many more transport (e.g. UDP, Publish/Subscribe based Asynchronous messaging service), however, can be added to the HHFR implementation to make the channel choice more flexible.

A message streaming in protocol level is one of two issues in our HHFR design of message stream that can be achieved through many transport protocols. The other issue is a semantic level message streaming which can be achieved by exchanging messages in a flexible representation and storing redundant / unchanging message part at the Context-store. Thus adding transport protocol to the current implementation strengthens the design, though this will make the reliability and security model more complex because each transport protocol has different characteristics for reliability and security model.

[60] The representation may be big in size.

Chapter 9.

Conclusions

This dissertation has addressed some key issues in the design and implementation of Web Service messaging frameworks in mobile computing environments including the reduction of message size while preserving semantics and the proposition of new communication mechanisms to better utilize of the high latency and narrow bandwidth wireless network. We argue that the conventional SOAP-based Web Service messaging through HTTP fails to deliver a sufficient performance model in order to integrate mobile devices as Web Service participants. Particularly, applications on mobile devices which have a frequent message exchange ratio suffer more. This fact creates an increasing need for an architecture which supports more efficient data representation in message exchanges and a high performance communication channel.

In this dissertation, our primary goal is to design and develop an overall system framework, which supports an efficient Web Service message exchange communication model in mobile computing environments. In the Handheld Flexible Representation

architecture, applications or Web Service participants exchange messages in an optimized streaming fashion, and a representation of those messages are flexibly chosen by participants through the negotiation at the beginning of the stream. This flexible representation is achieved by distinguishing between message semantics and message syntax.

We have developed the HHFR prototype and have presented the details of the implementation in this dissertation. We chose Java as the language platform for both mobile and conventional computing sides, but the architecture is not limited to any specific language platforms and can be applied to any message communication between heterogeneous platforms. The prototype implements three distinct things: a) "the interpret-style stubs" to encode and decode messages, b) the high-performance communication channel, and c) the Context-store to store meta-data of a message stream. We presented a description language, the Simple_DFDL, which is a small subset of the XML Schema Definition (XSD) but has a few additions. The Simple_DFDL is used to describe the data structure and types and we have demonstrated the successful automatic data conversion between a descriptive format (i.e. SOAP) and non-descriptive format (i.e. binary) using Simple_DFDL description and "interpret-style stubs". We explained how this can be generalized to DFDL when this is deployed.

The message stream in HHFR framework is achieved in two levels. A semantic level message stream can be achieved by exchanging message in a flexible representation and storing redundant / unchanging message parts in the Context-store. Messages in the stream which shares meta-data (e.g., a message representation, message parts, and stream characteristics) will be related each other.

The primary purpose of using the Context-store in HHFR framework is to provide the Web Service Database semantics. However, since an application of a participating node can retrieve stream meta-data, the use of the Context-store also guarantees a semantically persistent recovery from the connection or node failure by building the semantically same message stream from the disconnection.

We showed that the negotiation using SOAP messages is interoperable with the conventional Web Service messaging paradigm and is a sufficient mechanism to set up an optimized high-performance communication channel. If a responding participant to a negotiation request SOAP message responds with a negative value on the HHFR capability. Or the participant responds with a SOAP fault message, the negotiation initiator will fall back to conventional messaging.

We have evaluated the performance of HHFR messaging through two benchmark applications. We presented our system models and examined the analytic cost models for both HHFR and conventional messaging. Our observed empirical results show that the breakeven point of the two benchmark applications is found after only a few messages are exchanged between participants. Applications in the HHFR significantly outperform the conventional messaging framework after the breakeven point. We found the only major overhead in the HHFR messaging is the negotiation overhead. Additionally, we evaluated the Context-store access performance through our mobile interfaces and presented the scalability analysis of this service, which can serve up to hundreds of mobile sessions concurrently.

We have argued that it is critical to address the performance issues in mobile Web Service messaging at the system level rather than in small pieces. Integrating small

solutions in pieces may produce additional problems in scalability, robustness, or performance. We claim that the HHFR architecture addresses issues at the overall system framework level and provides an alternative message serializing mechanism: a description language, communication channel options, and interfaces to the Information Services in order to store meta-data, all in a single design. Our flexible representation messaging approach is more efficient for mobile applications, which exchange a series of messages, even while it is interoperable to the conventional Web Service messaging.

Efficient Web Service messaging is essential for mobile applications despite recent improvements in device specifications and infrastructure since the gap between wireless and wired computing environments still exists and won't be closed easily for technical and economic reasons. Our experience and evaluation demonstrate that the HHFR messaging approach is efficient in mobile Web Service messaging, which is exchanging a series of messages and helps to close the gap.

Bibliography

[1] 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.

[2] F. Berman, G. C. Fox, and J. G. Hey, *Grid Computing: making the Global Infrastructure a Reality*, Wiley, 2002.

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

[4] W. C. Y. Lee, *Mobile Cellular Telecommunications: Analog and Digital Systems 2nd Edition*, McGraw-Hill, Inc.

[5] The Globus Alliance, http://www-unix.globus.org/.

[6] Bouncy Castle, http://www.bouncycastle.org.

[7] Common API for XML Pull Parsing, http://www.xmlpull.org/.

[8] Java 2 Platform, Micro Edition (J2ME), http://java.sun.com/j2me/.

[9] Java 2 Platform, Standard Edition (J2SE), http://java.sun.com/j2se/.

[10] C Sharp language (C#), http://msdn.microsoft.com/vcsharp/programming/language/.

[11] Visual Basic Language, http://msdn.microsoft.com/vbrun/default.aspx.

[12] XML Schema-Based Compression (XSBC), http://cvs.sourceforge.net/viewcvs.py/xmsf/xsbc/.

[13] XBIS XML Information Set Encoding, http://xbis.sourceforge.net/.

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

[15] kXML, http://kxml.sourceforge.net/.

[16] PKZIP, http://www.pkware.com.

[17] Palm Operating System, http://www.palmsource.com/.

[18] Symbian Operating System, http://www.symbian.com/.

[19] Windows CE Operating System, http://msdn.microsoft.com/embedded/windowsce/default.aspx.

[20] International Business Machines Corporation (IBM), http://www.ibm.com.

[21] Microsoft Corporation (MS), http://www.microsoft.com.

[22] BEA Systems (BEA), http://www.bea.com.

[23] Web3D Consortium, Inc., "Extensible 3D (X3D) ISO/IEC 19775:2004," http://www.web3d.org/x3d/specifications/ISO-IEC-19775-X3DAbstractSpecification/.

[24] Java Servlet, http://java.sun.com/products/servlet/.

[25] .NET Framework, http://msdn.microsoft.com/netframework/.

[26] Apache AXIS Group, "Web Service Security", http://ws.apache.org/axis/java/security.pdf.

[27] Apache AXIS, http://ws.apache.org/axis.

[28] Apache AXIS2, http://ws.apache.org/axis2.

[29] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: an Open Grid Services Architecture for distributed systems integration," *Open Grid Service Infrastructure WG, Global Grid Forum*, June 2002.

[30] S. S. Nair, " XML Compression Techniques: A Survey," Department of Computer Science, University of Iowa, 2004.

[31] H. Liefke and D. Suciu, "XMill: an efficient compressor for XML data," In *Proceedings of ACM SIGMOD International Conference on Management of DATA 2000,* Dallas, TX, USA, May 2000.

[32] 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.

[33] J. Kobielus. Wrestling XML down to size: reducing the burden on networks and servers. *Bussiness Communications Review*. December Issue pp. 35-38, December 2004.

[34] M. Caporuscio, A. Carzaniga, and A. Wolf, "Design and evaluation of a support service for mobile, wireless publish/subscribe applications," *IEEE Transactions on Software Engineering*, vol. 29, pp. 1059-1071, December 2003.

[35] M. Govindaraju, A. Slominski, V. Choppella, R. Bramley, and D. Gannon, "Requirements for and Evaluation of RMI Protocols for Scientific Computing," In *Proceedings of ACM/IEEE Super Computing 2000 SC2000*, Dallas, TX, USA, November 2000.

[36] 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.

[37] 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.

[38] N. Adu-Ghazaleh, M. J. Lewis, and M. Govindaraju, "Performance of Dynamically Resizing Message Fields for Differential Serialization of SOAP Messages," In *Proceedings of International Symposium on Web Services and Applications*, pp. 783-789, Las Vegas NV USA, June 2004.

[39] N. Adu-Ghazaleh, M. Govindaraju, and M. J. Lewis, "Optimizing Performance of Web Services with Chunk-Overlaying and Pipelined-Send," In *Proceedings of International Conference in Internet Copmuting*, pp. 482-485, Las Vegas NV USA, June 2004.

[40] L. Fiege, F. Gartner, O. Kasten, and A. Zeidler, "Supporting mobility in content-based publish/subscribe middleware," In *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware 2003*, Rio de Janeiro, Brazil, June 2003.

[41] L. Fiege, G. Muhl, and F. Gartner, " A modular approach to build structured event-based systems," In *Proceedings of ACM Symposium on applied computing SAC 2002*, Madrid, Spain, March 2002.

[42] P. Eugster, P. Felber, R. Guerraoui, and A. Kermarrec, "The Many Faces of Publish/Subscribe," *ACM Computing Survey*, vol. 35, pp. 114-131, June 2003

[43] S. Oh, G. C. Fox, and S. Ko, "GMSME: An Architecture for Heterogeneous Collaboration with mobile Devices," In *Proceedings of The Fifth IEEE/IFIP Conference on Mobile and Wireless Communications Networks*, Singapore, October 2003.

[44] S. Oh, H. Bulut, A. Uyar, W. Wu, and G.C. Fox, "Optimized Communication using the SOAP Infoset for Mobile Multimedia Collaboration," In *Proceedings of The Fifth International Symposium on Collaborative Technologies and Systems (CTS2005)*, St. Louis, Missouri, USA, May 2005.

[45] S. Oh and G. C. Fox, "Optimizing Web Service Messaging Performance in Mobile Computing," *Community Grids Laboratory Technical Paper*, March 2006.

[46] S. Pallickara and G. C. Fox, " NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids," In *Proceedings of ACM/IFIP/USENIX International Middleware Conference Middleware2003*, Rio de Janeiro, Brazil, June 2003.

[47] C. Werner, C. Bushmann, T. Jacker, and S. Fischer, "Enhanced Transport Bindings for Efficient SOAP message," In *Proceedings of the IEEE International Conference on Web Services ICWS2005*, Orlando, FL, USA, July 2005.

[48] G. H. Forman and J. Zahorjan, "The Challenges of Mobile Computing," *IEEE Computer Magazine*, vol. 27, pp. 38-47, April 1994.

[49] Werner Vogels, "Web Services are not distributed objects," *IEEE Internet Computing Magazine*, vol. 7, pp. 59-66, November/December 2003.

[50] M. Weiser, "Hot topics: Ubiquitous Computing," *IEEE Computer Magazine*, vol. 26, pp. 71-72, October 1993.

[51] M. Satyanarayanan, "Pervasive Computing: Vision and Challenges," *IEEE Personal Communications*, vol. 8, pp. 10-17, August 2001.

[52] D. Sara and A. Mukherjee, "Pervasive Computing: A paradigm for the 21st Century," *IEEE Computer Magazine*, vol. 36, pp 25-31, March 2003.

[53] J. J. Kistler and M. Satyanarayanan, "Disconnected Operation in the Coda File System," *ACM Transactions on Computer Systems*, vol. 10, pp. 3-25, February 1992.

[54] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," In *Proceedings of the I.R.E*, pp. 1098-1102, September 1952.

[55] J. Ziv and A. Lempel, "A universal Algorithm for Sequential Data Compression," *IEEE Transactions on Information Theory*, vol. IT-23, May 1977.

[56] M. Hapner, R. Burridge, R. Sharma, J. Fialli, and K. Stout, "Java Message Service," *JavaTM Message Service specification*, Sun Microsystems, Inc., April 2002.

[57] 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.

[58] M. Beckerle and M. Westhead, "GGF DFDL Primer," Global Grid Forum Data Format Description Language Working Group Memo, May 2004.

[59] R. Williams, "XSIL: Java/XML for Scientific Data," *California Institute of Technology Technical Paper*, June 2000, http://www.cacr.caltech.edu/projects/xsil/xsil_spec.pdf.

[60] P. Sandoz, S. Pericas-Geertsen, K. Kawaguchi, M Hadley, and E. Pelegri-Llopart, "Fast Web Services," *Java developer's Journal Technical Article*, August 2003.

[61] P. Sandoz, A. triglia, and S. Pericas-Geertsen, "Fast Infoset," *Java developer's Journal Technical Article*, June 2004.

[62] P. Sandoz and S. Pericas-Geertsen, "Fast Infoset @ Java.net," In *Proceedings of XTech 2005*, Amsterdam, Netherands, May 2005.

[63] H. F. Nielsen, H. Sanders, R. Butek, and S. Nash, "Direct Internet Message Encapsulation," *Internet-Draft*, June 2002 expires December 2002, http://www.ietf.org/internet-drafts/draft-nielsen-dime-02.txt.

[64] J. H. Gailey, "Sending files, attachments, and SOAP messages via Direct Internet Message Encapsulation," *The Microsoft Journal for Developers*, December 2002, http://msdn.microsoft.com/msdnmag/issues/02/12/DIME/default.aspx.

[65] M. Powell, "Web Services, Opaque Data, and the Attachments Problem," *The Microsoft Developer Network Library*, June 2004, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwebsrv/html/opaquedata.asp

[66] M. Juntao Yuan and J. Long, "Java Readies itself for wireless Web Services", *JavaWorld Magazine Article*, June 2002, http://www.javaworld.com/javaworld/jw-06-2002/jw-0621-wireless.html.

[67] M. Juntao Yuan, "Access Web Services from Wireless Devices," *JavaWorld Magazine Article*, August 2002, http://www.javaworld.com/javaworld/jw-08-2002/jw-0823-wireless.html.

[68] R. Salz, "XML versus the Infoset," *Xml.com Article*, November 2002, http://webservices.xml.com/pub/a/ws/2002/11/20/ends.html.

[69] B. Schneier, "Crypto-Gram Newsletter," Blog post to www.schneier.com, June 15, 2000. http://www.schneier.com/crypto-gram-0006.html.

[70] L. Dodds, "Investigating the Infoset," *Xml.com Article*, August 2000, http://www.xml.com/lpt/a/2000/08/02/deviant/infoset.html.

[71] B. Siddiqui, " Web Services Security," *Xml.com Article*, March 2003, http://webservices.xml.com/pub/a/ws/2003/03/04/security.html.

[72] 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.

[73] J2ME^{tm} Web Services Specification (JSR 172), http://www.jcp.org/en/jsr/detail?id=172.

[74] Extended BNF ISO/IEC 14977 : 1996(E), http://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

[75] T. Boutell, et al., "PNG (Portable Network Graphics) Specification version 1.0," March 1997, http://www.ietf.org/rfc/rfc2083.txt.

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

[77] P.Deutsch, "DEFLATE Compressed Data Format Specification version 1.3," May 1996, http://www.ietf.org/rfc/rfc1951.txt.

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

[79] Information Science Institute at University of Southern California, "Internet Protocol version 4 (IPv4): RFC 791," September 1981, http://www.ietf.org/rfc/rfc0791.txt.

[80] R. Fielding et al., "Hyper Text Transfer Protocol – HTTP/1.1," June 1999, http://www.w3.org/Protocols/rfc2616/rfc2616.html.

[81] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, F. Yergeau, and J. Cowan, "Extensible Markup Language (XML) 1.1," *W3C Recommendation*, February 2004.

[82] J. Cowan and R. Tobin, "XML Information Set (2nd Edition)," *W3C Recommendation*, February 2004.

[83] T. Bray, D. Hollander, A. Layman, R. Tobin, "Namespaces in XML 1.1," *W3C Recommendation*, February 2004.

[84] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and H. F. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework," *W3C Recommendation*, June 2003.

[85] E. Christensen, F. Curbera, G. Meredith, S. Weerawarana, "Web Services Description Language (WSDL) 1.1," *W3C Recommendation*, March 2001.

[86] M. Gudgin, N. Mendelsohn, M Nottingham, and H Ruellan, "SOAP Message Transmission Optimization Mechanism (MTOM)," *W3C Recommendation*, January 2005.

[87] M. Gudgin, N. Mendelsohn, M Nottingham, and H Ruellan, "XML-binary Optimized Packaging (XOP)," *W3C Recommendation*, January 2005.

[88] A. Karmarkar, M. Gudgin, and Y. Lafon, "Resource Representation SOAP Header Block (RRSHB)," *W3C Recommendation*, January 2005.

[89] W3C, "Report From the W3C Workshop on Binary Interchange of XML Information Item Sets," *W3C Report*, September 2003.

[90] O. Goldman and D. Lenkov, "XML Binary Characterization," *W3C Working Group Note*, March 2005.

[91] M. Cokus, "XML Binary Characterization Use Cases," *W3C Working Draft*, March 2005.

[92] M. Cokus, "XML Binary Characterization Properties," *W3C Working Draft*, March 2005.

[93] S. D. Williams and P. Haggar, "XML Binary Characterization Measurement Methodologies, *W3C Working Draft*, March 2005.

[94] S. Anderson et al., "Web Service Secure Conversation Language (WS-SecureConversation)," February 2005, ftp://www6.software.ibm.com/software/developer/library/ws-secureconversation.pdf

[95] D. Bunting et al., "Web Services Context (WS-Context) 1.0," July 2003.

[96] R. Bilorusets et al., "Web Services Reliable Messaging Protocol (WS-ReliableMessaging)," Bea Systems, IBM, Microsoft Corporation, Inc and TIBCO Software Inc., February 2005.

[97] B. Atkinson et al., "Web Services Security (WS-Security) 1.0," IBM, Microsoft Corporation, Verisign, Inc., April 2002.

[98] J. Boyer, "Canonical XML 1.0," *W3C Recommendation*, March 2001, http://www.w3.org/TR/2001/REC-xml-c14n-20010315.

[99] Mapping W3C XML Schema Definitions into ASN.1 (X.694), http://www.itu.int/ITU-T/studygroups/com17/languages/X694.pdf

[100] S. Vinoski, "CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environment," *IEEE Communications Magazine*, vol. 35, Issue 2, February 1997.

[101] Sun Microsystems, Remote Method Invocation (JavaRMI), http://java.sun.com/products/jdk/rmi/

[102] Microsoft Corporation, Distributed Common Object Model (DCOM), http://www.microsoft.com/com/default.mspx

[103] T. Bellwood, L. Clement, and C. Riegen, "Universal Description, Discovery and Integration (UDDI) Version 3.0.1, *UDDI Specification Technical Committee Specification*, October 2003. http://uddi.org/pubs/uddi-v3.0.1-20031014.htm

[104] IEEE Computer Society, IEEE 802.11 Working Group, http://www.ieee802.org/11/

[105] USA Statistics, Telecommunications, http://www.census.gov/prod/2004pubs/04statab/infocomm.pdf

[106] IEEE Computer Society, IEEE Std 802.11b-1999, http://standards.ieee.org/getieee802/download/802.11b-1999.pdf

[107] IEEE Computer Society, IEEE Std 802.11g-2003, http://standards.ieee.org/getieee802/download/802.11g-2003.pdf

[108] Sony Computer Entertainment, PlayStation Portable (PSP), http://www.yourpsp.com/psp/locale.html

[109] R. Fielding, Architectural Styles and the Design of Network-based Software Architectures," PhD Dissertation, University of California, Irvine, California, USA, 2000.

[110] Community Grids Lab, Fault Tolerant High Performance Information System (FTHPIS), http://www.opengrids.org/extendeduddi/index.html

[111] Geography Markup Language (GML) ISO/TC 211/WG 4/PT 19136, http://portal.opengeospatial.org/files/?artifact_id=4700

[112] XML Binary Characterization Working Group, http://www.w3.org/XML/Binary/

[113] DOM4J, http://dom4j.org/index.html

[114] C. Evans et at., "Web Services Reliability (WS-Reliability) Ver 1.0," January 2003.

[115] Sun Microsystems, http://www.sun.com/

[116] Universal Mobile Telecommunications Systems (UMTS), ESTI Standard, http://webapp.etsi.org/exchangefolder/es_201385v010101p.pdf

[117] CDMA Multi-Carrier (CDMA2000), Candidate Harmonized European Standard, http://webapp.etsi.org/exchangefolder/en_30190804v020201p.pdf

[118] General Packet Radio Service (GPRS), ESTI Technical Report, http://webapp.etsi.org/exchangefolder/en_301344v070301p.pdf

[119] Enhanced Data rates for GSM Evolution (EDGE), ETSI Technical Report, http://webapp.etsi.org/exchangefolder/tr_150059v040001p.pdf

[120] J. Postel, "User Datagram Protocol," RFC 768, http://www.ietf.org/rfc/rfc768.txt

[121] T. Berners-Lee, R. Fielding, and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax," Network Working Group (RFC 3986), January 2005, http://www.ietf.org/rfc/rfc3986.txt

[122] W3C, "XML Schema". http://www.w3.org/XML/Schema

[123] K. Rose, "Data Format Description Language (DFDL) Overview, " *IBM Virtual XML Garden Project Report*, November 2005

[124] H. Thompson, C. M. Sperberg-McQueen, N. Mendelsohn, D. Beech, and M. Maloney, "XML Schema 1.1 Part 1: Structures", *W3C Working Draft*, March 2006.

[125] Community Grids Lab, Global Multimedia Collaboration System (GlobalMMCS), http://www.globalmmcs.org/

[126] Nokia, Nokia 3650, http://europe.nokia.com/nokia/0,,2273,00.html

[127] Palm, Treo 600, http://www.palm.com/us/products/smartphones/treo600/

[128] Matlab Basic Fitting, http://www.mathworks.com/access/helpdesk/help/techdoc/data_analysis/f1-8561.html

[129] Community Grids Lab, Geographical Information Systems Research, http://www.crisisgrid.net/

[130] E. Newcomer et al., " Web Service Composite Application Framework (WS-CAF)," http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ws-caf

[131] K. Czajkowski et al., "The WS-Resource Framework," *Globus Project White Paper*, March 2004, http://globus.org/wsrf/specs/ws-wsrf.pdf

[132] K. Ballinger, et al., "Web Services Metadata Exchange," September 2004, http://msdn.microsoft.com/library/en-us/dnglobspec/html/ws-metadataexchange.pdf

[133] M. Aktas, G. Fox, and M. Pierce, "Managing Dynamic Metadata as Context," In *Proceedings of The 2005 Istanbul International Computational Science and Engineering Conference (ICCSE2005)*, Istanbul, Turkey, June 2005.

[134] V. Dialani, "UDDI-M Version 1.0 API Specification," *UDDI Extensions Draft Specification*, March 2001.

[135] A. ShaikhAli, O. Rana, R. Al-Ali, and D. Walker, "UDDIe: An Extended Registry for Web Services," In *Proceedings of the Service Oriented Computing: Models, Architectures and Applications*, Orlando, Florida USA, January 2003.

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

[137] Unified Modeling Language (UML), Object Management Group, http://www.uml.org/

[138] C. Krueger, "Will cell phones replace watches? Time will tell", St. Petersburg Times, December 19, 2005.

[139] JibX: Binding XML to Java Code, http://jibx.sourceforge.net/

[140] XMLBeans, Apache XML Project, http://xmlbeans.apache.org/

[141] Fast schema, Sun Microsystems, https://jwsdp.dev.java.net/fast/

Curriculum Vitae

Sangyoon Oh received B.E. in Mechanical Design from Sungkyunkwan University, Seoul, Korea in 1995. He got his M.S. in Computer Science from Syracuse University, Syracuse in 1999.

During his Ph.D. years, he has worked in Community grids Laboratory at Indiana University, in Computational Science and Information Technology (CSIT) at Florida State University, and in Northeast Parallel Architecture Center (NPAC) at Syracuse University as a research assistant.