Extending the Web Services Architecture (WSA) for Video Streaming

by

Gibson Lam

A Thesis Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfillment of the Requirements for

the Degree of Doctor of Philosophy

in Computer Science and Engineering

January 2012, Hong Kong

<u>**Authorization**</u>

Gibson Lam

16th January 2012

Extending the Web Services Architecture (WSA) for Video Streaming

by

Gibson Lam

This is to certify that I have examined the above PhD thesis

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by

the thesis examination committee have been made.

Professor David ROSSITER, Thesis Supervisor

Professor Mounir HAMDI, Head of Department

Department of Computer Science and Engineering

16th January 2012

Acknowledgements

I would like to thank my family – my wife, my parents and my kids for encouraging me during my PhD study. I have to thank my wife and my kids for sacrificing our family time so that I can work on my PhD study. Although my father is not around anymore I have to thank him for his support through my entire life.

I have to thank very much my supervisor, Dr. David Rossiter, for his excellent guidance during my PhD study. Thank you very much for his patience and useful suggestions and recommendations for everything, not just for my PhD study.

I have to thank Prof. Andrew Horner, Prof. Shing-Chi Cheung and Dr. Wilfred Ng for taking their precious time to be in the examination committee of my PhD thesis proposal defence. I have to thank Prof. Shing-Chi Cheung and Dr. Wilfred Ng again for being in my final thesis defence examination committee. For Prof. Richard So and Prof. Patrick Hung I would like to thank both of you to come to my final thesis defence as part of the examination committee. Especially for Prof. Patrick Hung, thank you very much for coming all the way from Canada to Hong Kong to take part in the defence.

TABLE OF CONTENTS

Title Page i

Authorization Page ii

Signature Page iii

Acknowledgements iv

Table of Contents v

List of Figures ix

List of Listings xii

List of Tables xiii

Abstract xiv

Chapter 1 Introduction 1

1.1 Service Oriented Architecture 1

1.2 Multimedia Web Services 1

1.2.1 An Example Application of Multimedia Web Services 3

1.3 Thesis Objective 3

1.3.1 Multimedia Metadata 4

1.3.2 Streaming Data Transfer between Web Services Endpoints 5

1.3.3 Inefficiency of XML-based Data Communication 6

1.4 Organization 8

Chapter 2 Background 9

2.1 Web Services Architecture 9

2.1.1 SOAP 12

2.1.2 SOAP Message Exchange Pattern 13

2.1.2.1 Request-Response MEP 13

2.1.2.2 Response MEP 14

2.1.3 SOAP HTTP Binding 15
2.1.3.1 HTTP GET Method 16
2.1.3.2 HTTP POST Method 18
2.1.4 Binary Data in SOAP Messages 19
2.1.4.1 SOAP Messages with Attachments 19
2.1.4.2 Direct Internet Message Encapsulation 20
2.1.4.3 SOAP Message Transmission Optimization Mechanism 21
2.1.5 WSDL 23
2.1.6 WS-Addressing 25
2.1.7 WS-Notification 26
2.2 Streaming Multimedia 27
2.2.1 Multimedia Streaming over Internet Protocols 28
2.2.1.1 UDP 29
2.2.1.2 TCP 29
2.2.2 RTP 30
2.3 Compression Techniques for SOAP Messages 31
2.3.1 Generic Text Compressors 32
2.3.2 XML Compression Techniques 33
2.3.3 SOAP Compression Techniques 33
2.3.4 Binary XML 34
Chapter 3 Related Work 36
3.1 Single-Request/Multiple-Response Messaging in Web Services 36
3.1.1 Publish-and-subscribe Systems in Asynchronous Web Services 36
3.1.2 Clearinghouse Web Service 38
3.2 Multimedia Web Services Architecture 39

3.2.1 QoS Capable Multimedia Web Services 39
3.2.2 SOAP-Oriented Component-Based Framework 41
3.2.3 Web Service-Oriented Transmoding Service 43
3.2.4 Personalized Universal Media Access Architecture 45
Chapter 4 Design of the Multimedia Streaming Web Services Architecture 47
4.1 Overview 47
4.2 Content Description for Multimedia Web Services 49
4.2.1 Metadata Query Services 50
4.3 Streaming Data Transfer 52
4.3.1 Message Exchange Patterns 53
4.3.1.1 Single Request-Multiple Response (SRMR) MEP 53
4.3.1.2 Multiple Request-Multiple Response (MRMR) MEP 54
4.3.2 SOAP HTTP Binding 57
4.3.2.1 SOAP Packaging for a Multimedia Stream 57
4.3.2.2 SOAP HTTP Binding for the SRMR MEP 60
4.3.2.3 SOAP HTTP Binding for the MRMR MEP 61
Chapter 5 Compression of SOAP Messages 63
5.1 A Modified A-SOAP Compression Method 65
5.2 Performance of Compression Techniques 69
Chapter 6 Experiments and Results 72
6.1 Experimental Setup 72
6.1.1 Simulation Setup 73
6.1.2 Transmission Methods 76
6.1.2.1 HTTP Streaming 76
6.1.2.2 SOAP-Oriented Component-based Framework 77

6.1.2.3 Proposed Multimedia Streaming Web Services Architecture 78

6.2 Results 79

6.2.1 RTP Packet Delay 81

6.2.2 PSNR of Video Sequences 94

Chapter 7 Conclusion 102

7.1 Future Work 102

7.1.1 Multimedia Web Service Composition 102

7.1.2 QoS Aware Multimedia Web Services 103

Bibliography and References 104

LIST OF FIGURES

Figure 1. An example scenario of two devices using three multimedia Web services. 2

Figure 2. RPC invocation in the RPC-oriented Web services... 10

Figure 3. The service roles relationship in the Web services approach of SOA. 11

Figure 4. The state transition diagram of the SOAP nodes in the Request-Response MEP. 14

Figure 5. The state transition diagram of the SOAP nodes in the Response MEP. 15

Figure 6. The components inside a WSDL file. 24

Figure 7. An example of the publish-and-subscribe system in WS-Notification. 27

Figure 8. A comparison of synchronous and asynchronous Web services. 37

Figure 9. The relationship among the components in the CWS framework........................ 38

Figure 10. The architecture of the QoS server, QoS broker and client in QCWS. 40

Figure 11. The architecture of the transmoder service. 44

Figure 12. The service-oriented framework for personalized multimedia adaptation. 46

Figure 13. The flow of the extended Web services architecture. 48

Figure 14. The state transition diagram of the Single Request-Multiple Response (SRMR) MEP. 54

Figure 15. The state transition diagram of the Multiple Request-Multiple Response (MRMR) MEP. 56

Figure 16. The sequence diagram of the asynchronous process of sending and receiving streaming data between the requesting SOAP node and the responding SOAP node. 62

Figure 17. The SOAP response structure of the streaming data transmission mechanism in the SRMR MEP. 64

Figure 18. The dictionary snapshots of the original A-SOAP method and the modified A-SOAP method. 66

Figure 19. An example representation of an open tag using the modified A-SOAP method in this study with the dictionary shown on the right. 69

Figure 20. The simple single bottleneck network used in the multimedia streaming simulation. 73

Figure 21. Description of the video sequences used in the simulation. 75

Figure 22. An illustration of the bitrate of the interfering traffic and the video sequences used in the simulation 76

Figure 23. Cumulative distribution function of RTP packet delay of each transmission method for the video sequence akiyo using a 2.8Mbps bandwidth. 81

Figure 24. Cumulative distribution function of RTP packet delay of each transmission method for all video sequences except akiyo using a 2.8Mbps bandwidth. The axes are the same as those shown in Figure 23. 83

Figure 25. Average RTP packet delay of all transmission methods for all video sequences using a 2.8Mbps bandwidth. 86

Figure 26. Approximate playback buffer size required by all transmission methods for all video sequences using a 2.8Mbps bandwidth. 88

Figure 27. The average RTP packet delay for each transmission method against different available bandwidth for the video sequence akiyo. 90

Figure 28. Average RTP packet delay of each transmission method against different bandwidth values for all video sequences except akiyo. The axes are the same as those shown in Figure 27. 91

Figure 29. The average PSNR of each video sequence using different transmission methods with a 2.8Mbps available bandwidth. 97

Figure 30. The MOS of each video sequence using different transmission methods with a 2.8Mbps available bandwidth. 99

Figure 31. The average PSNR of all video sequences for each transmission method against different levels of available bandwidth. 100

LIST OF LISTINGS

Listing 1. An example SOAP message. 12

Listing 2. An example of a HTTP GET request in the Response MEP. 16

Listing 3. An example of a HTTP response in the Response MEP. 17

Listing 4. An example of a HTTP POST request in the Request-Response MEP. 17

Listing 5. An example of a HTTP response in the Request-Response MEP. 18

Listing 6. An example of a SwA package with two attachments. 20

Listing 7. An example of a SOAP message containing two encoded binary data. 22

Listing 8. A XOP package of a SOAP message with two binary data. 23

Listing 9. An example usage of WS-Addressing in a SOAP message. 26

Listing 10. An example fragment of the first SOAP message in a sequence of messages. 42

Listing 11. An example of part of a MPEG-7 metadata for a video. 50

Listing 12. An example WSDL description of the operation 'GetMedia' showing an associated metadata query service called 'GetMediaDesc'. 51

Listing 13. An example MTOM serialization of a data packet of a multimedia stream. 59

Listing 14. A sequence of SOAP messages encoded using the MIME Multipart/Related content type in a HTTP response. 60

LIST OF TABLES

Table 1. The RTP packet structure. 31

Table 2. Performance of compression techniques per packet for the SOAP packaging of a multimedia stream. 71

Table 3. Transmission methods used in the simulations. 77

Table 4. Statistics for the RTP packet delay (in seconds) of each transmission method for the video sequence akiyo using a 2.8Mbps bandwidth........ 84

Table 5. Average RTP packet delay (in seconds) and the standard deviations of the HTTP method and the MSWS-EXI method for all video sequences using a 2.8Mbps bandwidth. 85

Table 6. Approximate playback buffer size required by all transmission methods for all video sequences using a 2.8Mbps bandwidth. 87

Table 7. The average RTP packet delay for each transmission method against different available bandwidth for the video sequence akiyo. 89

Table 8. The ratio between the RTP delay values of the MSWS-EXI method and the SCWS method for each video sequence, expressed as a percentage. 93

Table 9. Frame loss percentage of all transmission methods for all video sequences using a 2.8Mbps bandwidth........ 94

Table 10. Average PSNR (in dB) and standard deviation of PSNR (in dB) of the video sequences and their transmission methods using a 2.8Mbps available bandwidth. 96

Table 11. The perceived quality rating of the Mean Opinion Score (MOS) and the mapping between PSNR (in dB) and MOS. 98

Extending the Web Services Architecture (WSA) for Video Streaming

by Gibson Lam

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

Abstract

Multimedia systems are typically monolithic systems that are expensive and challenging to maintain and develop. With the advances of Service Oriented Architecture (SOA) a component-based approach can be applied to multimedia systems development. A complex and large scale multimedia system can be split into smaller and reusable components. These components can be composed in different ways so that the development of multimedia systems can be flexible and the resulting component-based system is more maintainable, reliable and scalable than a monolithic system.

One of the common implementations of SOA is Web services. However the transfer of streaming data, which is a major part of multimedia systems, is not well supported by current Web services standards. To reap the benefits of the component-based approach for multimedia systems development, this thesis proposes a novel extension for multimedia streaming support in the Web services architecture. The proposed extension focuses on three issues: the discoverability of a multimedia Web service, the implementation of streaming data transfer between two Web service endpoints and the efficiency of the streaming data transfer.

First, the discoverability of a Web service relies on a standardized and descriptive metadata for its content. The proposed extension provides a query service so that the description of multimedia content input to and output from a multimedia Web service can be published via an extension of WSDL. Second, the proposed extension implements the streaming data transfer between two service endpoints using two new Message Exchange Patterns (MEPs) and their corresponding HTTP bindings. The implementations use MIME and Message Transmission Optimization Mechanism (MTOM) to transfer streaming multimedia as a stream of SOAP messages. Third, to reduce the transfer overhead introduced by the packaging method, this thesis investigated extensively the application of various compression schemes to the SOAP messages as well as to the packaging of the binary packet data. Experiments show that the proposed extension, coupled with the use of the binary XML encoding scheme, has a significant improvement in performance over current approaches for multimedia streaming transfer.

CHAPTER 1

INTRODUCTION

1.1 Service Oriented Architecture

Service oriented architecture (SOA) is a collection of principles related to the development of systems using capabilities provided by services in a distributed environment [1]. SOA emphasizes the reusability, growth and interoperability of solutions developed as a composition of the distributed services. This facilitates better scalability and manageability, and less cost for the development of large-scale systems. One of the common implementations of the SOA model is Web services. Web services allow machines or software to communicate over the network on different platforms in a standardized manner, using a set of XML-based standards. Data is commonly encapsulated and transmitted using SOAP, which is a lightweight protocol for exchanging structured information in an XML-based format. Under the Web services architecture (WSA) a given task can be achieved by consuming a set of services from one or more different service providers.

1.2 Multimedia Web Services

Multimedia systems are typically monolithic systems that are expensive and challenging to maintain and develop. Given the advances of SOA the development of complex multimedia systems should be expected to reap the benefits of using SOA. Using the SOA concepts a complex multimedia system can be built from the composition of multiple small reusable components. In the Web services architecture these components are multimedia Web services, which process and/or provide multimedia data as a service.

As opposed to a monolithic system this approach supports reusability of the individual services and flexibility in the development of multimedia applications. Figure 1 shows an example scenario.

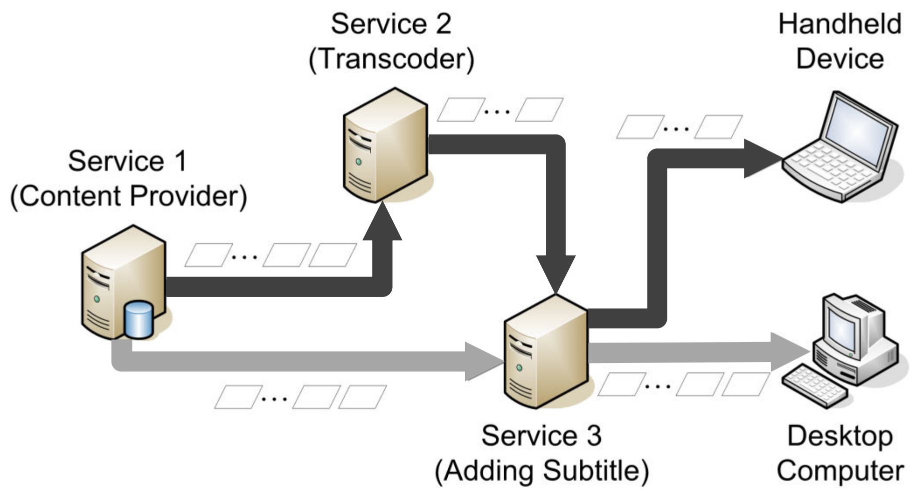


Figure 1. An example scenario of two devices using three multimedia Web services.

In Figure 1, both devices request a video stream from a particular video provider. They also need to have subtitles added to their video. Because of limitations such as display capability the handheld device may choose to receive a video stream of a lower quality. To add subtitles the video stream being sent to the desktop computer goes through an adding subtitle service before reaching the destination. The video stream destined for the handheld device goes through an additional transcoder service before having the subtitles added.

However, because of the complexity of multimedia workflow and the rich semantic structure of multimedia data SOA has yet to make significant impact on multimedia application development [2]. In particular, real-time applications such as multimedia streaming systems are not well supported by Web services standards. A key problem is the lack of support for the transfer of streaming data between two Web service endpoints.

1.2.1 An Example Application of Multimedia Web Services

YouTube API [3] is a Web API for developers to programmatically use functions provided by the YouTube video sharing website [4]. The YouTube API is divided into two parts, the Data API and the Player API. The Data API allows a program to use YouTube operations such as searching for YouTube videos, retrieving RSS feeds and managing user profiles. The Player API lets developers put videos into their own applications through the use of a customizable YouTube video player.

While it is a great API for anyone to display videos in his/her own website the API only allows the videos to be displayed inside the YouTube video player. If, for example, a developer wants to show a video inside another video, the so-called Picture-in-Picture feature, he/she will not be able to achieve it using the YouTube API. A possible improvement would be to supply the videos as multimedia streams through the Data API which is exposed as a multimedia Web service. Via this service developers could retrieve YouTube videos using the YouTube Web service and then process the content using their own program, or combine the YouTube Web service with other multimedia Web services to suit their own applications.

1.3 Thesis Objective

This thesis proposes a novel extension in the Web services architecture supporting multimedia streaming. The proposed extension allows multimedia Web services to be published, composed and consumed just like other Web services. It includes a new communication mechanism for multimedia streaming content. The multimedia content is transmitted between two Web service endpoints in a provider and requester relationship. However, the extension does not include any details about the underlying multimedia compression techniques because the streaming communication is independent of the data compression.

Development of large multimedia systems can be achieved using a composition of services within the Web services architecture. This distributed component-based development has the benefits of the SOA paradigm such as maintainability, reliability, reusability and scalability of the systems and their components. To support multimedia streaming in the Web services architecture several fundamental issues have to be investigated, which are the main contributions of this thesis.

1.3.1 Multimedia Metadata

To sustain the interoperability of services the service interfaces have to be well defined and described. These definitions and descriptions are typically contained inside WSDL which are published in service registries. Therefore these WSDL interface descriptions could also be applied to multimedia Web services. However, XML Schema, the language used to describe data in WSDL, is not sufficient for the description of multimedia resources. Because of that, a specialized metadata scheme [5] such as MPEG-7 [6], MXF [7] and Dublin Core [8] is used so that a better description of multimedia data can be facilitated.

One of the issues of using multimedia metadata schemes is that the metadata created by these schemes are relatively larger than the XML Schema of an ordinary Web service operation. Moreover, since the inputs and outputs of different Web service operations may have their own distinct metadata the growth of the size of WSDL will make it unmanageable if all metadata information is stored within WSDL. To solve this problem the proposed extension provides a simple query mechanism so that service requesters can obtain multimedia metadata efficiently.

1.3.2 Streaming Data Transfer between Web Services Endpoints

In multimedia applications, when the multimedia data is large or contains live data, streaming is typically used as the delivery method. However, current Web services standards do not support streaming data transfer between two Web service endpoints. That implies multimedia Web services have to transfer multimedia data in its entirety, which reduces the efficiency of the application and sometimes cannot be achieved because the total size of the data is not known at the start.

Various studies [9][10][11][12][13][14] have investigated the development of multimedia applications in the context of SOA. These studies consider different ways to separate a multimedia application into a control path and a data path. Application controls are transmitted using standard Web services implementation. The multimedia data is transmitted in another channel such as raw TCP connection or a multimedia streaming server, which are not part of the Web services architecture. One of the advantages of these approaches is that they can take advantage of some readily available technologies for multimedia transfer and at the same time a service-oriented development can be employed. However, the major shortcomings of these approaches are that, because of the need to maintain external entities, the interoperability, maintainability and scalability of the application are compromised. Moreover, some WS-* specifications such as WS-Security are not applicable to the multimedia data, which hinders the extensibility of the underlying multimedia system.

In order to have a streaming data transfer compatible with the WS-* extensions the data has to be carried inside the XML infosets of the SOAP messages. However, streaming data by definition is a collection of time-dependent small data packets. To transfer these small packets the Web services communication needs to have the capability to transmit multiple objects.

Ruth et al. [15] suggested that a proxy service can be used to incorporate a Single-Request/Multiple-Response message exchange pattern in Web services. In addition, Web services extensions such as WS-Notification [16] and WS-Eventing [17] provide mechanisms that can be used to send multiple messages from a provider to a requester. However, these approaches are not designed for time-critical message transmission. When multiple messages are required a separate request has to be made for each message, which is time-consuming.

In this thesis the proposed extension removes the dependency on any external entity by streaming multimedia data encapsulated in SOAP messages. A combination of MIME structure and MTOM attachment scheme is used to transmit the streaming data in a single connection. This transmission arrangement is created in a new HTTP SOAP binding. Because the streaming data is transmitted within the Web services architecture, the multimedia Web services in the proposed extension can be published, composed and consumed just like any other Web services. In addition, WS extensions can be applied to the transmitted data because the data is encapsulated in the XML infoset.

1.3.3 Inefficiency of XML-based Data Communication

Web services, because of their reliance on XML-based protocols, have suffered from performance issues for time-critical applications. Research has investigated the limitations of using SOAP in various applications such as high-performance scientific applications [18] and real-time trading systems [19]. For multimedia applications, Heinzl et al. studied the performance of SOA-based multimedia applications by transmitting multimedia data using various SOAP schemes [20]. They compared the performances of multimedia data transmission using Flex-SwA [21], SwA and normal SOAP messages with base64 encoding.

Flex-SwA, which is essentially an attachment scheme which works by putting the attachments in external locations, excels against SwA and normal SOAP messages. However, this comparison did not take into account the various ways to apply compression to the SOAP transmission.

The videoconferencing system discussed by Oh et al. [22] used SOAP messages to encapsulate and transmit a sequence of pictures, which is transmitted as frames of a video stream. To improve the efficiency of the system the authors used Fast Web Services [23], a binary encoding scheme for SOAP messages based on ASN.1 [24], when transferring the SOAP messages. However, in their study they do not have a detailed comparison of performances based on the use of the ASN.1 representation.

The transfer overhead of the streaming data transfer includes both the transmitted SOAP message content and the overhead in the packaging method. If an appropriate compression scheme is applied to one or both of these the efficiency of the data transfer will be improved. In this thesis a comprehensive investigation has been carried out on the application of compression schemes to the streaming data transfer in the proposed extension. A wide variety of compression schemes have been tested. They include generic text compressors, XML compressors, SOAP compressors and binary XML representations. The compression methods are applied to a single package, which includes the packaging overhead, the SOAP message and the binary data, of a packet. Given the results, the best compression scheme is then applied to a multimedia Web service in the proposed extension. The performances of the proposed extension, both with and without using a compression scheme, are compared against a simple HTTP streaming method and a naïve Web services transmission method.

1.4 Organization

The thesis is organized as follows. This chapter discusses the objective and overview of the thesis. Chapter 2 covers background information about multimedia Web services. At the start of the chapter various related topics in Web services architecture are described. After that, multimedia streaming and multimedia metadata description using MPEG-7 are discussed. At the end of Chapter 2 a survey on compression techniques for SOAP messages is presented. Chapter 3 discusses previous work on multimedia application development using the SOA model. The design of the proposed multimedia Web services extension is presented in Chapter 4. This chapter covers the two major components of the proposed entension. First, it introduces a metadata query service. This service is used to describe the input and/or output of a multimedia Web service when streaming multimedia content is involved. Second, it explains the need to use streaming data transfer within the Web services architecture. The details of this streaming data transfer for streaming multimedia data is described in detail in 4.3. The transfer is implemented using two new message exchange patterns (MEPs) and their corresponding SOAP HTTP bindings within the Web services architecture. Using the proposed SOAP HTTP bindings a considerable amount of size overhead is introduced into the transmission of the multimedia data. Because of this, compression is applied to the transmission so as to improve the efficiency of the transfer. Chapter 5 compares the performance of using various SOAP compression techniques on the proposed multimedia Web services extension. Chapter 6 presents details of the experiment models and setup, and the results. Finally in Chapter 7, this thesis concludes with a summary and possible future research directions.

CHAPTER 2

BACKGROUND

2.1 Web Services Architecture

The Web services architecture (WSA) is a standard architecture for software applications to cooperate using interoperable and reusable functionalities over different platforms. From the definition of the World Wide Web consortium (W3C) a Web service is "a software system designed to support interoperable Machine to Machine interaction over a network" [25]. The key principle of Web services is the interoperability of services among different machines, platforms or frameworks. Each Web service contains a well-defined and self-contained functionality which can be invoked by another service. To ensure interoperability for these services a set of open standards is employed to define the interface of the services as well as the underlying communication between these services. These standards do not enforce how and where the services are implemented but they form the basis of the interoperability within the Web services architecture. Several key standards, such as SOAP and WSDL, will be explained in the next few sections.

Web services can be used in two different styles, the RPC-oriented style and the document-oriented style. In the RPC-oriented style a Web service describes and implements the functionality of a remote procedure call (RPC). The invocation of the RPC involves two roles, the service requester and the service provider. The service requester consumes the service. In other words, it invokes the operation of the RPC. The service provider defines the interface and contains the implementation of the operation provided by the RPC. Figure 2 illustrates the two-way communication when the service requester invokes a RPC func(X) from the service provider. When the service requester invokes the operation func(X) it asks the service provider for the interface of the RPC. The service provider exposes the interface, which describes the service, to the requester. Then the requester constructs an invocation request of the RPC with the required parameters and sends the request to the provider. The result of the invocation is returned to the requester. The RPC Web Services are defined and communicated using the open standards of Web Services. And therefore they are reusable and interoperable regardless of the underlying implementation of the operations provided by the RPC. Nevertheless an implicit requirement is that the requester and the provider have a common knowledge of the RPC interface, for example, the operation name and its parameters. This violates the loose coupling requirement in Service Oriented Architecture (SOA), to be discussed in the next paragraph.

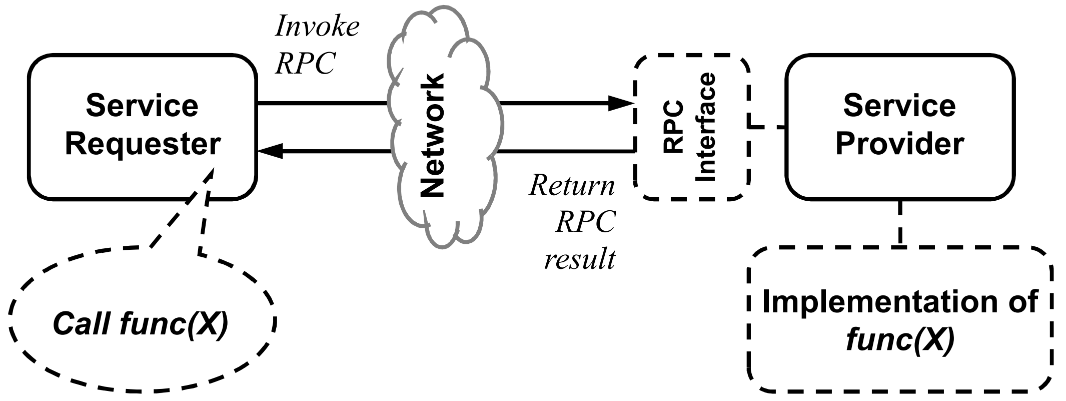


Figure 2. RPC invocation in the RPC-oriented Web services.

The document-oriented style of Web Services is used to implement a Service Oriented Architecture. The Organization for the Advancement of Structured Information Standards (OASIS) has the following definition for Service Oriented Architecture: "Service Oriented Architecture is a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains" [26]. In essence SOA is a design for developing software applications from a collection of services offered in a distributed environment, without knowing the underlying implementation of the services [1]. Services in

SOA are well-defined, self-contained and loosely coupled functionalities with an emphasis on their reusability and interoperability. Three service roles are defined in the Web Services approach of SOA. They are the service requester, the service provider and the service broker. Similar to the RPC-oriented style the service requester consumes the service while the service provider is the entity who supports and implements the service. The service broker has a public registry of services, which is a collection of service descriptions. These service descriptions are published by the service providers who want to expose their services. The service registry allows the service requester to find appropriate services and bind with the corresponding service provider. Their relationship is illustrated in Figure 3.

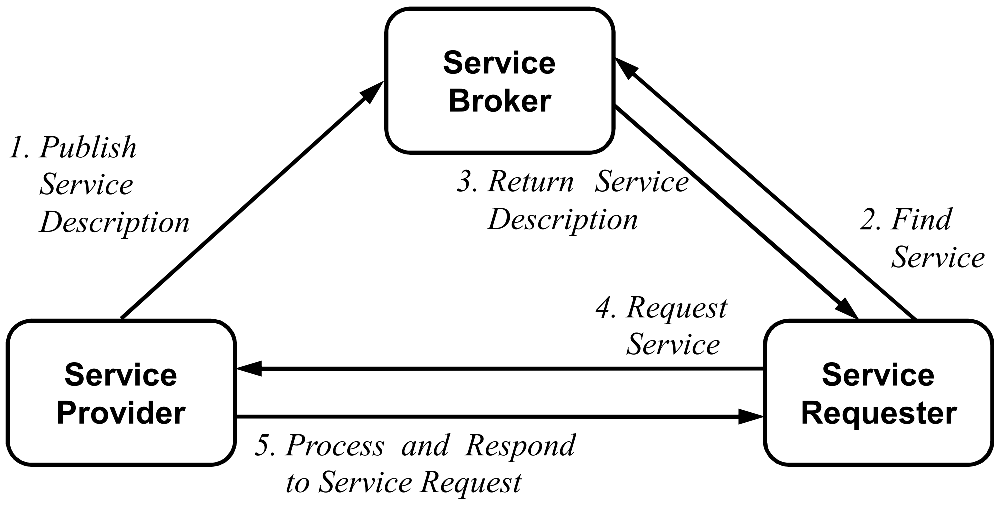

Service Broker
1. Publish Service Description
3. Return Service Description
2. Find Service
4. Request Service
Service Provider
Service Requester
5. Process and Respond to Service Request

Figure 3. The service roles relationship in the Web services approach of SOA.

In document-oriented style, communication between the requester and the provider is based on a document instead of an operation. Each message contains an autonomous document and therefore this ensures the loose coupling property of each service as well as the messaging between the services. The services offered by service providers are therefore differed by their corresponding processing of the document.

2.1.1 SOAP

SOAP is a lightweight protocol for exchanging structured and typed information in a decentralized and distributed environment [27]. It is an XML based messaging protocol which is a standardized, extensible, human-readable serialization of data. It forms the messaging standard of the Web services architecture.

SOAP provides a message construct which can be exchanged in a variety of transport protocols such the Simple Mail Transfer Protocol (SMTP) [28] and the Hypertext Transfer Protocol (HTTP) [29]. SOAP messages are exchanged between two nodes, the SOAP sender and the SOAP receiver. The message exchange can be a one-way, a request/response interaction or a peer-to-peer conversation according to the message exchange pattern (MEP) to be discussed later. The method of message transmission in the underlying protocol of the SOAP message exchange is called a SOAP binding. SOAP messages can be bound to various protocols. The most commonly used protocol is HTTP because of the popularity of the protocol in the Internet.

Listing 1. An example SOAP message.

```
 1: <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 2:   <env:Header>
 3:     <h:mailheader xmlns:h="http://example.org/mailheader">
 4:       <h:priority>1</h:priority>
 5:     </h:mailheader>
 6:   </env:Header>
 7:   <env:Body>
 8:     <m:mail xmlns:m="http://example.org/mail">
 9:       <m:author>John Chan</m:author>
10:       <m:subject>Reminder</m:subject>
11:       <m:content>Remember the meeting is at 9am!</m:content>
12:     </m:mail>
13:   </env:Body>
14: </env:Envelope>
```

A SOAP message consists of an outer envelope, a header block and a body. The header block can be used to disseminate information to any intermediate SOAP nodes which relay the SOAP message to the SOAP receiver. The SOAP body contains the message to be transmitted between the initial sender and the ultimate receiver. An example of a SOAP message is shown in Listing 1.

2.1.2 SOAP Message Exchange Pattern

The SOAP message exchange pattern describes the pattern of messages exchanged between the SOAP nodes [30]. It can be a one-way exchange, a request-response interaction or a peer-to-peer conversation. MEPs define only the pattern of the message exchange. The detail of the SOAP message transmission is described by the SOAP binding on which the MEPs operate. The current SOAP specification defines two SOAP MEPs, the Request-Response MEP and the Response MEP.

2.1.2.1 Request-Response MEP

The Request-Response MEP defines a pattern consisting of two SOAP messages, the request message and the response message. These two messages are transmitted between two SOAP nodes acting as the requesting node and the responding node. First, the requesting node sends a request message to the responding node. If the responding node successfully processes the request message, a response message will be transferred back to the requesting node. Figure 4 illustrates the state transition diagram of the MEP for the two SOAP nodes.

The MEP involves exactly two SOAP messages in a normal operation. For example, a client can send a SOAP message containing the keywords for a product search to a company. The company then searches its product database and returns the matching products to the client using a SOAP message.

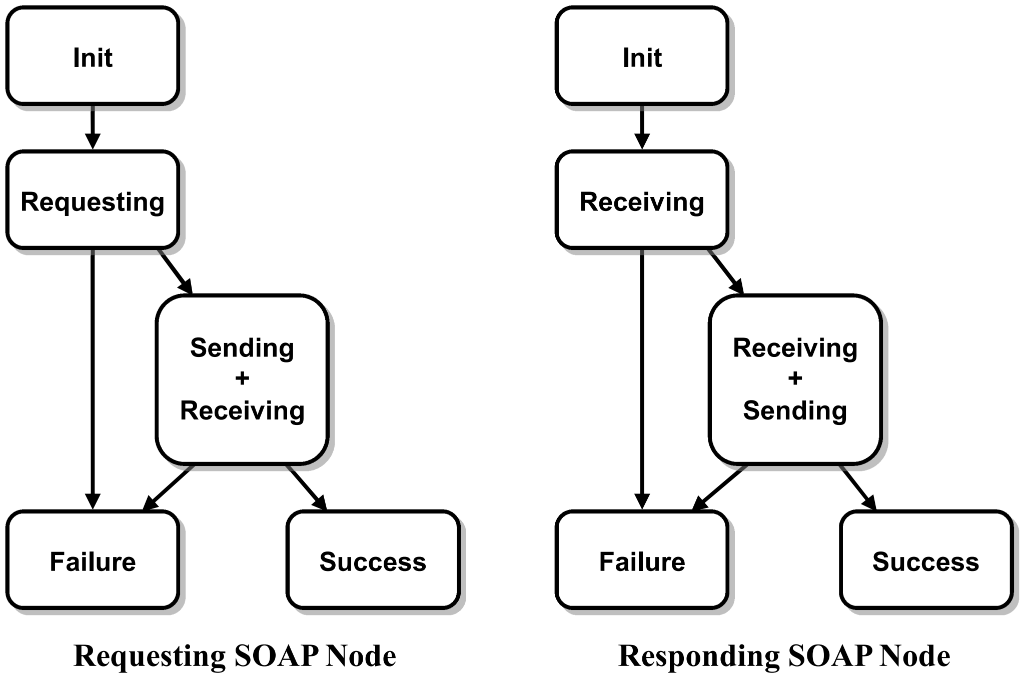
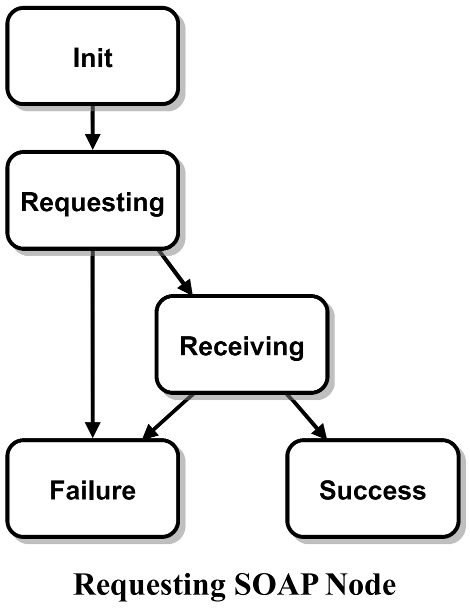


Figure 4. The state transition diagram of the SOAP nodes in the Request-Response MEP.

2.1.2.2 Response MEP

The operation of the Response MEP is similar to the Request-Response MEP where there are a requesting SOAP node and a responding SOAP node. The MEP is initiated by a request from the requesting SOAP node. However the request in this MEP does not consist of a SOAP message. It is sent in a binding-specific manner which is defined in the SOAP binding applying the MEP. The request is analogous to an empty request with no content. When the request reaches the responding SOAP node the node transmits the response message to the requesting node. Figure 5 shows the state transition diagram of the Response MEP.



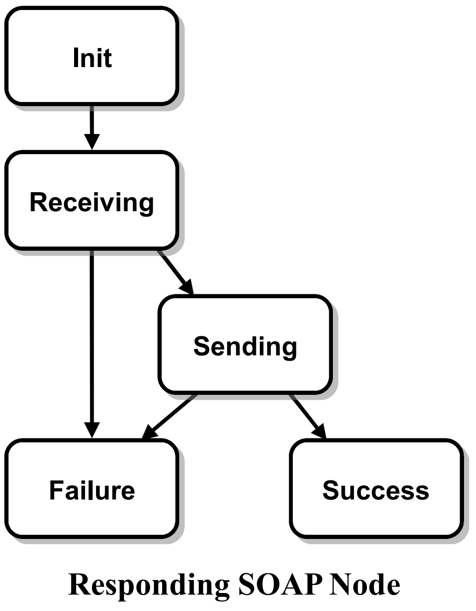


Figure 5. The state transition diagram of the SOAP nodes in the Response MEP.

For example, the observatory can create a Web service which returns the current temperature. If anyone wants to know the temperature he/she can submit a request to the service. The observatory then returns the current temperature reading via a SOAP message. Since the service does not require any input the person who wants to read the temperature can just put in a request without sending any additional information.

2.1.3 SOAP HTTP Binding

The SOAP HTTP binding is a SOAP binding for transferring SOAP messages on top of the HTTP protocol [27][31]. Because the HTTP protocol is one of the most dominant Internet protocols most of the Web Services are bound using the SOAP HTTP binding. The binding supports both MEPs described in the previous sections. The HTTP protocol uses a request-response communication model. The client first specifies the server using an URI. The client then sends an HTTP request to the server, the server processes the request and finally the server responds by returning an HTTP response to the client. The SOAP HTTP binding uses the HTTP GET method and the HTTP POST method to model the Response MEP and the Request-Response MEP respectively.

2.1.3.1 HTTP GET Method

In the SOAP HTTP binding, the HTTP GET method is used to implement the Response MEP. The HTTP request of the GET method is a URI representing a resource. No data is transmitted along with the HTTP request. The purpose of the HTTP GET request is for information retrieval which fits into the request of the Response MEP. Listing 2 contains an example of a GET method requesting a mail from a Web Service bound to an HTTP server. The request shows that the client only accepts a SOAP message with a type of ‘application/soap+xml’. This conforms to the response of the Response MEP, which returns a SOAP message to the requesting SOAP node from the responding SOAP node.

Listing 2. An example of a HTTP GET request in the Response MEP.

```
GET /mailservice?id=138639 HTTP/1.1
Host: example.org
Accept: application/soap+xml
```

Upon receiving the HTTP request, the HTTP server returns an HTTP response containing a SOAP message. Using the HTTP request in Listing 2 the returned SOAP message is the requested mail in the example response shown in Listing 3.

Listing 3. An example of a HTTP response in the Response MEP.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 455

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <h:mailheader xmlns:h="http://example.org/mailheader">
      <h:priority>1</h:priority>
    </h:mailheader>
  </env:Header>
  <env:Body>
    <m:mail xmlns:m="http://example.org/mail">
      <m:id>138639</m:id>
      <m:author>John Chan</m:author>
      <m:subject>Reminder</m:subject>
      <m:content>Remember the meeting is at 9am!</m:content>
    </m:mail>
  </env:Body>
</env:Envelope>
```

Listing 4. An example of a HTTP POST request in the Request-Response MEP.

```
POST /mailservice HTTP/1.1
Host: example.org
Accept: application/soap+xml
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 339

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Body>
    <n:readMail
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
      xmlns:n="http://example.org/mailservice">
      <m:mail xmlns:m="http://example.org/mail">
        <m:id>138639</m:id>
      </m:mail>
    </n:readMail>
  </env:Body>
</env:Envelope>
```

2.1.3.2 HTTP POST Method

The HTTP POST method allows data to be transmitted in both the HTTP request and response. Therefore it can be used to implement the Request-Response MEP where both the request and the response contain a SOAP message.

The HTTP response in the Request-Response MEP is similar to the Response MEP which contains the SOAP response message. The corresponding response for the POST request in Listing 4 is shown in Listing 5.

Listing 5. An example of a HTTP response in the Request-Response MEP.

```
HTTP/1.1 200 OK
Content-Type: application/soap+xml; charset="utf-8"
Content-Length: 611

<?xml version='1.0'?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
  <env:Header>
    <h:mailheader xmlns:h="http://example.org/mailheader">
      <h:priority>1</h:priority>
    </h:mailheader>
  </env:Header>
  <env:Body>
    <n:readMailResponse
      env:encodingStyle="http://www.w3.org/2003/05/soap-encoding"
      xmlns:n="http://example.org/mailservice">
      <m:mail xmlns:m="http://example.org/mail">
        <m:id>138639</m:id>
        <m:author>John Chan</m:author>
        <m:subject>Reminder</m:subject>
        <m:content>Remember the meeting is at 9am!</m:content>
      </m:mail>
    </n:readMailResponse>
  </env:Body>
</env:Envelope>
```

2.1.4 Binary Data in SOAP Messages

SOAP is an XML-based messaging protocol which means a SOAP message can only contain text-based content. If binary data has to be included in a SOAP message the data will be encoded using either the `xs:hexBinary` encoding or the `xs:base64Binary` encoding [32]. Since these encodings change binary data into a text representation they typically occupy at least an additional one third of the original size of the data. Therefore transmitting SOAP messages with binary data is not efficient.

2.1.4.1 SOAP Messages with Attachments

SOAP messages with attachments (SwA) is a method to improve the transmission of SOAP messages with binary data [33]. In SwA binary data is not included within the XML infoset [34] but is instead sent as attachments.

SwA uses the MIME Multipart/Related content type [35] to package a SOAP message and its attachments. The Multipart/Related content type composes different objects into one single package. These objects are divided by a MIME boundary. Each attachment is referred by a content identifier in the SOAP message. Listing 6 shows an example of a SwA package with two attachments. Although quite a number of lines of the header are used for stating the MIME content type the SwA package is still very efficient for transmitting large binary data. The shortcoming for SwA is that the XML infoset is completely separated from the binary data and therefore any applications have to process them separately. In addition, other WS extensions can only be applied to the XML infoset but not the binary data.

Listing 6. An example of a SwA package with two attachments.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
  type="application/soap+xml"; start="<example.xml@example.org>"

--MIME_boundary
Content-Type: application/soap+xml; charset=UTF-8;
Content-Transfer-Encoding: 8bit
Content-ID: <example.xml@example.org>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope">
  <soap:Body>
    <m:data xmlns:m="http://example.org/data">
      <m:image href="cid:http://example.org/example.jpg"/>
      <m:audio href="cid:http://example.org/example.wav"/>
    </m:data>
  </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/example.jpg>

... binary data for the jpeg image file ...
--MIME_boundary
Content-Type: audio/wav
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/example.wav>

... binary data for the wav audio file ...
--MIME_boundary--
```

2.1.4.2 Direct Internet Message Encapsulation

Direct Internet Message Encapsulation (DIME) is a lightweight message format for encapsulating different type of payloads into a single message contruct [36]. The goal of DIME is to provide a simple and efficient way to encapsulate different documents into a single entity that can be transmitted as a single message. For example, a DIME message can contain a SOAP message together with its attachments so that the DIME message can be transmitted between Web service endpoints.

DIME is a lightweight format. A DIME message can contain separate DIME records, where the payload is store in each record. The DIME records are described by a type, an optional identifier and its data length. The payload and DIME headers are stored in binary format within a DIME record. Since DIME is a binary format the efficiency of DIME is an advantage over SwA. DIME also supports efficient boundary detection via the data length header and the padding and byte alignment scheme within the record.

DIME was created by Microsoft and has been used in Microsoft's Web Service Enhancements (WSE) library. However the poor support for DIME across different platforms has made it a poor choice considering one of the major attributes of Web services is interoperability. Since WSE 3.0 the support for DIME has been dropped by the library and thus making the DIME format obsolete. The attachment scheme used in the WSE library has become MTOM, which is described in the next section.

2.1.4.3 SOAP Message Transmission Optimization Mechanism

The SOAP Message Transmission Optimization Mechanism (MTOM) is an optimization of the transmission of binary data in SOAP messages [37]. MTOM makes use of the XML-binary Optimized Packaging (XOP) [38] to optimize the SOAP message transmission. XOP is a packaging scheme of XML messages with base64-encoded binary data, i.e. data with type `xs:base64Binary`. XOP transforms an XML message with base64-encoded binary data into two entities. The first entity is a XOP document which is the XML message without the encoded binary data. The second entity is the group of extracted binary data in their original binary format. Similar to SwA, a XOP package puts together the XOP document and the extracted binary data using the MIME Multipart/Related content type.

Listing 7. An example of a SOAP message containing two encoded binary data.

```
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xmlmime="http://www.w3.org/2004/11/xmlmime">
  <soap:Body>
    <m:data xmlns:m="http://example.org/data">
      <m:image
        xmlmime:contentType="image/jpeg">
        ... base64-encoded data of the jpeg image file ...
      </m:image>
      <m:audio
        xmlmime:contentType="audio/wav">
        ... base64-encoded data of the wav audio file ...
      </m:audio>
    </m:data>
  </soap:Body>
</soap:Envelope>
```

Listing 7 contains an example of a SOAP message with two encoded binary data. To transmit the message the base64-encoded binary data are first extracted from the message and are replaced with two `xop:Include` elements. The result is a XOP document with two references to the extracted binary data. The XOP document and the binary data are then packaged using the Multipart/Related content type and transmitted to the destination, as shown in Listing 8.

At the destination the original SOAP message is reconstructed from the XOP package. In some situations binary data can be directly put into or read from a XOP package and therefore the encoding and decoding of the data are not required. Conceptually XOP does not separate the XML infoset from the binary data. It is merely a representation of the SOAP message during its transmission and therefore it does not suffer from the same shortcomings as SwA.

Listing 8. A XOP package of a SOAP message with two binary data.

```
MIME-Version: 1.0
Content-Type: Multipart/Related; boundary=MIME_boundary;
  type="application/xop+xml"; start="<example.xml@example.org>"

--MIME_boundary
Content-Type: application/xop+xml; charset=UTF-8;
  type="application/soap+xml"
Content-Transfer-Encoding: 8bit
Content-ID: <example.xml@example.org>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
  xmlns:xmlmime="http://www.w3.org/2004/11/xmlmime">
  <soap:Body>
    <m:data xmlns:m="http://example.org/data">
      <m:image xmlmime:contentType="image/jpeg">
        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
          href="cid:http://example.org/example.jpg"/>
      </m:image>
      <m:audio xmlmime:contentType="audio/wav">
        <xop:Include xmlns:xop="http://www.w3.org/2004/08/xop/include"
          href="cid:http://example.org/example.wav"/>
      </m:audio>
    </m:data>
  </soap:Body>
</soap:Envelope>

--MIME_boundary
Content-Type: image/jpeg
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/example.jpg>

... binary data for the jpeg image file ...
--MIME_boundary
Content-Type: audio/wav
Content-Transfer-Encoding: binary
Content-ID: <http://example.org/example.wav>

... binary data for the wav audio file ...
--MIME boundary--
```

2.1.5 WSDL

The Web Services Definition Language (WSDL) is a XML-based language for describing Web Services [39]. The description is called a service contract for a Web Service. When a service requester wants to understand a Web Service the requester can read the service contract created by the service provider. When there is no service broker, as in the RPC-oriented Web Services, the service requester retrieves the WSDL directly from the service provider. In case a service broker is acting as a service registry, such as UDDI [40], WSDLs can be published to and retrieve from the registry. The registry stores a list of WSDL published by their service providers. A service requester can then search the registry and retrieve a suitable WSDL.

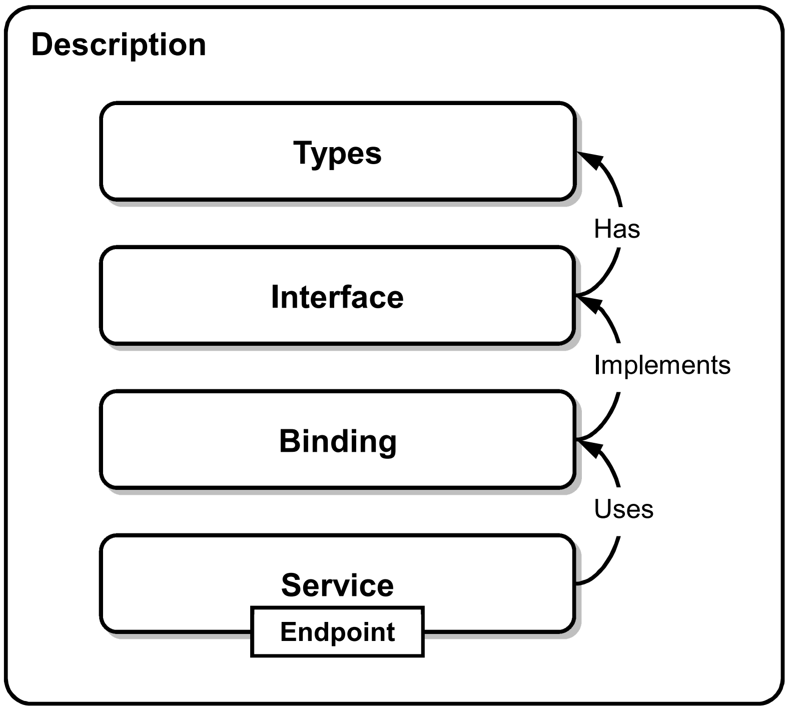


Figure 6. The components inside a WSDL file.

WSDL defines a Web Service as a collection of service endpoints. The Web Service is bound to a particular transmission protocol, such as HTTP, described by the binding component of the WSDL. The binding component contains the implementation details of the service in a transmission protocol. For each Web Service an interface component is used to describe the operations in the service as well as the structure of the input and output messages in these operations. The data types of the messages described in the interface component are typically defined using the XML Schema language [32][41]. The data types are first defined within the WSDL before the messages are described. Figure 6 shows the components in a WSDL file. The root element of WSDL is called the description, which contains all components of the file.

2.1.6 WS-Addressing

WS-Addressing [42] is one of the Web service extension in the WS-* family. It provides mechanisms to describe the Web service endpoints and messaging properties which is independent of the underlying transport protocols and messaging systems. WS-Addressing has two separate contructs.

The first contruct is called endpoint references. An endpoint reference describes the address and specific properties of a Web service endpoint. In its simplest form, it can be used to specify the address of a Web service endpoint. More importantly it can also be used to include additional properties or metadata associated with a Web service endpoint at a particular instance.

The second contruct is the message addressing properties. It defines the end-to-end relationship of a message transmitted between two Web service endpoints. A simple analogy of the message addressing properties is email headers. The message addressing properties provide information between the two endpoints of a message without any dependency on the underlying transport. Message addressing properties can be used to specify, for example, where the message is sent to, from where the message is sent, the action associated with the message and the identifier of the message. Some of these properties are expressed using an endpoint reference. Listing 9 shows an example SOAP message with a header block of message addressing properties. The example is a SOAP message with a message identifier of '12345678@example.org' as shown on line 4. The receiving address 'http://example.org/webservice' is shown on line 5 and the associated Web service action 'http://example.org/Search' is shown on line 6.

Listing 9. An example usage of WS-Addressing in a SOAP message.

```
 1: <soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"
 2:   xmlns:wsa="http://www.w3.org/2005/08/addressing">
 3:   <soap:Header>
 4:     <wsa:MessageID>12345678@example.org</wsa:MessageID>
 5:     <wsa:To>http://example.org/webservice</wsa:To>
 6:     <wsa:Action>http://example.org/Search</wsa:Action>
 7:   </soap:Header>
 8:   <soap:Body>
 9:     <media:Search xmlns:media="http://example.org/media">
10:       <media:keyword>example</media:keyword>
11:     <media:Search>
12:   </soap:Body>
13: </soap:Envelope>
```

2.1.7 WS-Notification

WS-Notification [16], like WS-Addressing, is a suite of specifications in the WS-* family. The purpose of WS-Notification is to create a notification-based messaging system, also called a publish-and-subscribe system, among a set of consumers and producers. WS-Notification supports a variety of the publish-and-subscribe system such as push/pull notifications and centralizing notifications using a notification broker.

Figure 7 shows an example of the publish-and-subscribe system in WS-Notification. A subscriber, who does not necessarily be the consumer, can subscribe to a topic in a notification producer. This subscription specifies who the receiver of the notifications is, i.e. the notification consumer, and the topic that the consumer is interested in. Then when a notification in the notification producer arises the notification is sent to the consumers who are interested in the topic of the notification. The notification messages can be pushed from the producer to the consumer or they can be pulled by the consumer from the producer. All of these are accomplished within SOAP messages communicated between the Web service endpoints of the subscribers, producers and consumers.

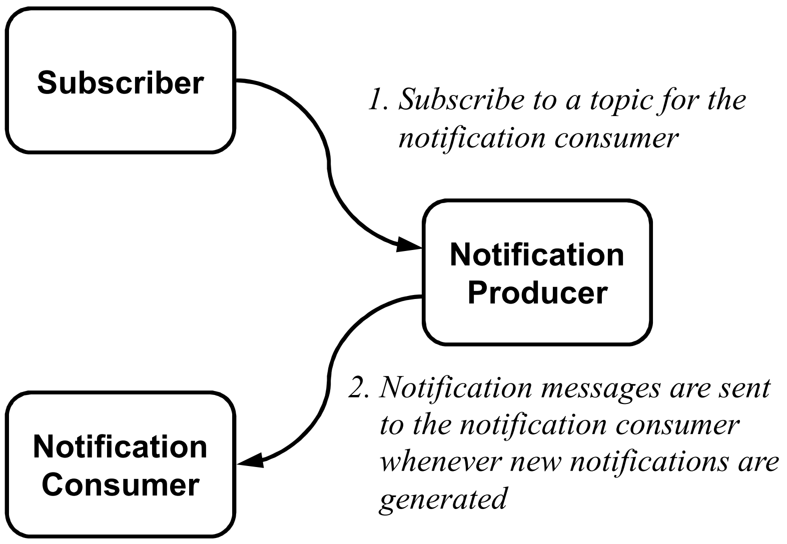

Subscriber
1. Subscribe to a topic for the notification consumer
Notification Producer
2. Notification messages are sent to the notification consumer whenever new notifications are generated
Notification Consumer

Figure 7. An example of the publish-and-subscribe system in WS-Notification.

2.2 Streaming Multimedia

Streaming multimedia is the delivery of multimedia content such as audio and video over a network in real-time. It means that the recipient of the multimedia transmission receives the content constantly, with little or no delay.

Multimedia content is generally very large in size because it contains huge amount of information. Even though the content is usually highly compressed, it is inappropriate to use full file download, i.e. to completely download and save the file before it is played, as a means for online multimedia delivery. There are two reasons not to use the full file download approach. First, to download a multimedia file the recipient must have very large storage space to cater for large multimedia files. For example, the size of a 10-minute video with an average bit rate of 300Kb/s is approximately 22MB. Second, the transfer of a complete multimedia file typically takes a long time to finish because of the limitation of network bandwidth. Using the above video example the time to transfer the video in a high speed network of 1000Kb/s is 3 to 4 minutes. That means that, for example, the recipient has to wait for 3 minutes before he/she can watch a 10-minute video.

In contrast, using streaming a multimedia file is broken into very small pieces. Each piece contains a fraction of the original multimedia and can be transmitted by itself. During streaming the multimedia provider encodes the pieces of multimedia and transmits the pieces one-by-one to the recipient continuously. The recipient can then play back the multimedia while receiving the multimedia. In this way the recipient does not need to have large storage because multimedia is played on the fly. In addition, the recipient can start playing once the first piece of data is received, with only a small amount of delay. One challenge for streaming multimedia is to design a streaming protocol which ensures continuous transmission of multimedia under networks with jitter and delay.

2.2.1 Multimedia Streaming over Internet Protocols

The delivery of streaming multimedia is different from the delivery of other form of information over the Internet. Static data, such as text and web pages, requires a lossless transmission from the source to the destination. Although data is subdivided into packets during transmission no lost packet is allowed and the order of the data packets must be preserved. On the contrary, streaming multimedia requires an on-time delivery. An occasional loss of data can be tolerated in order to have a guarantee of the timeliness of the arrival of streaming data.

2.2.1.1 UDP

The User Datagram Protocol (UDP) [43] is a connectionless transport protocol for transmitting data over the Internet Protocol (IP) [44]. UDP does not guarantee the arrival or order of the transmitted packet. Once sent a UDP packet is transmitted to the destination in a simple and efficient way. The advantage of using UDP for streaming data is that the UDP uses a best effort approach for packet transmission so that packets can reach the destination in a very short time, provided that the network does not suffer from heavy congestion. UDP does not guarantee packet delivery which means packet may go missing. Such packet loss incurs a dropout of the multimedia stream but this is tolerable by most streaming applications. The problem associated with UDP is that Internet firewalls commonly do not permit UDP traffic and therefore multimedia streaming over UDP is not possible in the networks behind such firewalls.

2.2.1.2 TCP

The Transmission Control Protocol (TCP) [45] is connection-oriented transport protocol which provides reliable and in-order data transmission. Similar to UDP, data is transmitted as packets. The difference in packet transmission between UDP and TCP is that the transmission of each UDP packet is independent while TCP packets arrive at the destination in exactly the order that they are sent. In addition, TCP retransmits packets which are lost during transmission. Although TCP ensures reliability and accuracy it does not work well with streaming multimedia. This is because a timeout can occur in TCP when packets are retransmitted. Therefore, in this situation, multimedia streams which are being sent via TCP will stall when there is congestion in the network. The advantage of using TCP over UDP for multimedia streaming is that TCP traffic can get through most Internet firewalls.

2.2.2 RTP

The real-time transport protocol (RTP) [46] is a protocol for delivering multimedia data such as audio and video in real-time over the Internet. RTP is used together with the real-time control protocol (RTCP) in providing the transmission of multimedia data as well as the following services: payload type identification, sequence numbering, time-stamping and delivery monitoring services. RTP specifies packet structure in the multimedia data transport. It does not require the use of a particular network protocol although it is commonly used with UDP. Table 1 shows the structure of RTP packets.

RTCP is responsible for monitoring the underlying network and providing feedback on the quality-of-service (QoS) of the RTP transfer to both the sender and the receivers. Typically control packets of RTCP are transmitted in a separate UDP channel between the sender and the receivers. RTCP packets contain various statistics such as bytes sent, lost packets, jitter and round trip delay of the data transmission. These statistics are used by the sender to adjust the multimedia stream when necessary.

Table 1. The RTP packet structure.

```
      0                   1                   2                   3
Bits
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
     +---+-+-+-------+-+-------------+-------------------------------+
     | V |P|X|  CC   |M|     PT      |        sequence number        |
     +---+-+-+-------+-+-------------+-------------------------------+
     |                           timestamp                           |
     +---------------------------------------------------------------+
     |           synchronization source (SSRC) identifier            |
     +---------------------------------------------------------------+
     |            contributing source (CSRC) identifiers             |
     |                             ....                              |
     +---------------------------------------------------------------+
     |                  header extension (optional)                  |
     |                             ....                              |
     +---------------------------------------------------------------+
     |                            payload                            |
     +---------------------------------------------------------------+
```

V	Version of the RTP packet
P	Padding bit identifying the existence of padding at the end of the payload
X	Header extension bit identifying the existence of a header extension after the RTP header
CC	Contributing source count showing the number of mixed sources for this packet
M	Reserved for application-specific marker
PT	Payload format of the packet [47]
Sequence number	Sequence number of the RTP packet, which is increased by one for each transmitted packet in the RTP session
Timestamp	Timestamp of the current packet
SSRC	Synchronization source field identifying the current RTP session
CSRC	Contributing source field listing the contributing sources who are contributing data in the packet
Header extension	Application-specific header extension
Payload	The payload data transmitted in the packet

2.3 Compression Techniques for SOAP Messages

A SOAP message is an XML-based communication protocol. With headers and namespace specifications occupying a considerable portion of the entire message, the actual payload is only a small part of the transmitted message. For example, by looking at the SOAP message shown in Listing 1, the actual payload of the message is in line 9-11. The majority of the example message is XML tags and namespace information. The ratio of the actual payload to the entire SOAP message is 48 bytes to 412 bytes, i.e. approximately 12%. In the context of a time critical application, such as streaming multimedia transmission, the size overhead is huge compared to other specialized multimedia transmission protocols. To improve the efficiency of SOAP transmission various compression and encoding techniques can be employed.

The compression and encoding techniques can be generally categorized into four categories: generic text compressors, XML compression techniques, SOAP compression techniques and binary XML.

2.3.1 Generic Text Compressors

Generic text compressors are general compression techniques which can be applied to any text-based content. Since SOAP messages are text-based messages generic text compressors can be used to reduce the size of a SOAP message. Many different generic text compressors are available commercially or as open source package. The most commonly used compressors include gzip, PPM and bzip2.

Using a generic text compressor on SOAP messages can compress the messages with an acceptable compression ratio. However, generic text compressors do not take into account the semantic meaning of the content of a SOAP message. This is an area where improvement in compression performance can be achieved. In the next few sections different strategies are employed to improve the compression rates on SOAP messages over the use of generic text compressors.

2.3.2 XML Compression Techniques

An XML document contains a lot of repetitive tag names and structural information. By exploiting this information an XML document can be compressed further when comparing to generic text compressors. Moreover, information such as tag names and document structure is well defined by the schema of an XML document. When an XML schema is available the corresponding XML document can then be encoded and highly compressed based on the information obtained from the XML schema.

XMLPPM [48] and XMill [49] are two of the most commonly used XML compressors. Both of the techniques try to rearrange the content of XML documents into separate parts. For example, XMLPPM extracts the structural information from the content of an XML document. This can improve the compression performance by encoding the repetitive information of the extracted structural information. Each of these parts of an XML document is then compressed using generic compressors such as PPM and gzip. An added advantage of XMLPPM is its ability to support incremental encoding and transmission of an XML document. That means one can start parsing and building up the content of a compressed XML document while it is still being transmitted.

A summary of general XML compression techniques can be found in [50].

2.3.3 SOAP Compression Techniques

XML compression techniques can be very efficient when dealing with XML documents. Since SOAP messages are XML-based messages applying XML compression techniques on SOAP messages will result in a more efficient compression than using generic text compressors.

When comparing XML documents and SOAP messages both of them can have useful information extracted from their corresponding XML schemas. In addition to that SOAP, by nature, is a messaging protocol used in communication. Comparing to a standalone XML document a SOAP message may look very similar to another SOAP message transmitted in the same communication channel. For instance, a lot of namespaces used by a sequence of SOAP messages are potentially the same. Also, the structure of the header and payload is likely to be the same across a similar type of Web services.

To take advantage of these redundancies, Werner et al. [51] have introduced a SOAP message compression technique using differential encoding. Their idea is to transmit only the difference between a SOAP message and a skeleton message generated using the WSDL of the Web service. Instead of using a skeleton message, Rosu proposed A-SOAP [52], which is a dictionary based technique for the compression of SOAP messages. A dictionary of commonly used tags is built up incrementally between the sender and the receiver. A SOAP message is then transmitted as an encoded entity based on the current dictionary.

2.3.4 Binary XML

Binary XML is a compact encoding scheme of an XML document using binary format. Unlike the compressors discussed in the previous sections compactness is not the sole purpose of using a binary XML scheme. Binary XML schemes also focus on the efficiency, flexibility and interoperability of the encoded format. Two prominent binary XML schemes are Fast Infoset [53] and Efficient XML interchange (EXI) [54]. Both of them aim at minimizing the encoding size and at the same time maximizing the processing speed of an XML document.

There is a distinct advantage of using the binary XML techniques when dealing with binary data. When binary attachments are sent together with an associated SOAP message the binary attachments can be directly inserted into the encoded SOAP message. This completely eliminates the need to use any packaging method, such as MTOM in section 2.1.4.3, to group the SOAP message and its attachments together during transmission. Because of this no additional headers are required for binary attachments when using binary XML as opposed to the additional headers created in any of the packaging methods.

CHAPTER 3

RELATED WORK

Chapter 1 has looked at research which focuses on the high level usage of multimedia Web services such as service composition and QoS control. In this chapter, previous work on single-request/multiple-response messaging systems, the design and implementation of multimedia Web services architecture will be discussed.

3.1 Single-Request/Multiple-Response Messaging in Web Services

3.1.1 Publish-and-subscribe Systems in Asynchronous Web Services

Web services can work in two different models: the synchronous model and the asynchronous model. Typical Web service applications use a synchronous model. That means the client sends a request to the service provider, keeps blocking other operations on the client system, waits for the response and then processes the response upon receiving it. If the client do not want to wait, or in some cases, cannot wait after sending the request to the service provider, it will have to take the asynchronous approach. In the asynchronous model the client sends the request to the service provider and then continues to work on other tasks. The client will switch back to process the response only when the provider sends the response to the client.

When a Web service is used in the synchronous model the request and response are restricted to each carry a single payload. However when the asynchronous model is employed the service provider can send the client as many payload messages as possible if the messages are properly addressed and transmitted. Thus an asynchronous Web service can be modeled as a single-request/multiple-response system. A comparison of synchronous and asynchronous Web services is shown in Figure 8.

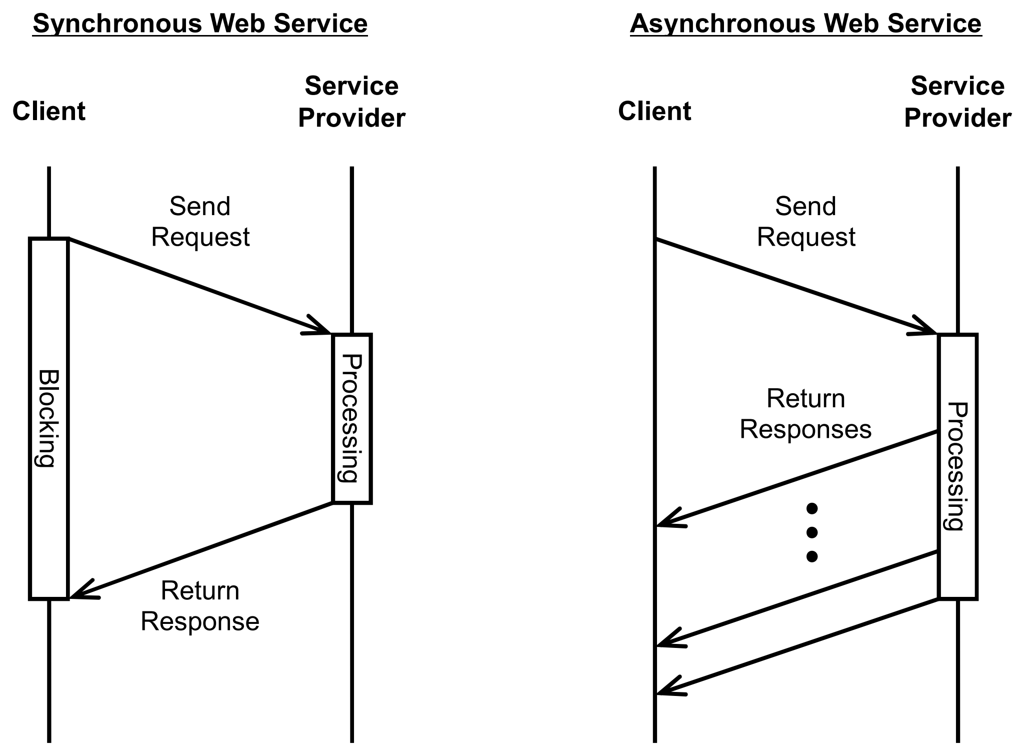


Figure 8. A comparison of synchronous and asynchronous Web services.

An ad hoc approach to implement an asynchronous Web service is to use threading on the client side. This can create a non-blocking call in the context of the client program. However, using a separate thread to handle a Web service operation does not alter the basic single request and response messaging model. Brambilla et al. discussed different ways to support asynchronous Web services and the interaction patterns supported by them [55]. They suggested that a publish-subscribe with acknowledgement pattern can be used to implement a single-request/multiple-responses messaging system. In their work various forms of correlation and coordination for the asynchronous operation were discussed. However none of those are standardized methods within the Web services architecture. Nevertheless, with the emergence of WS-Notification, as discussed in section 2.1.7, and WS-Eventing [17] an asynchronous publish-and-subscribe system can be created in a standardized manner. For example, in WS-Notification, the client of the single-request/multiple-response messaging system is in the role of both the notification subscriber and notification consumer in WS-Notification whereas the service provider is in the role of the notification provider. Then the client can request for (subscribe to) a specific topic and then keep receiving responses (notifications) from the service provider.

3.1.2 Clearinghouse Web Service

Ruth et al. provided an application-level solution for a Single-Request/Multiple-Response (SRMR) MEP to be implemented in the Web services architecture [15]. The idea of the work focuses on modeling real world problems, business applications in particular, which involve multiple responses to a single request. There are two major components. One of them is a specially designed Web service called the Clearinghouse Web Service (CWS). It is a message messenger which correlates and distributes messages between the clients and the service providers. Another component is an agent object created in the client-side framework. This agent object acts as a communication proxy between the client, CWS and the target Web services when single-request/multiple-response operations are involved.

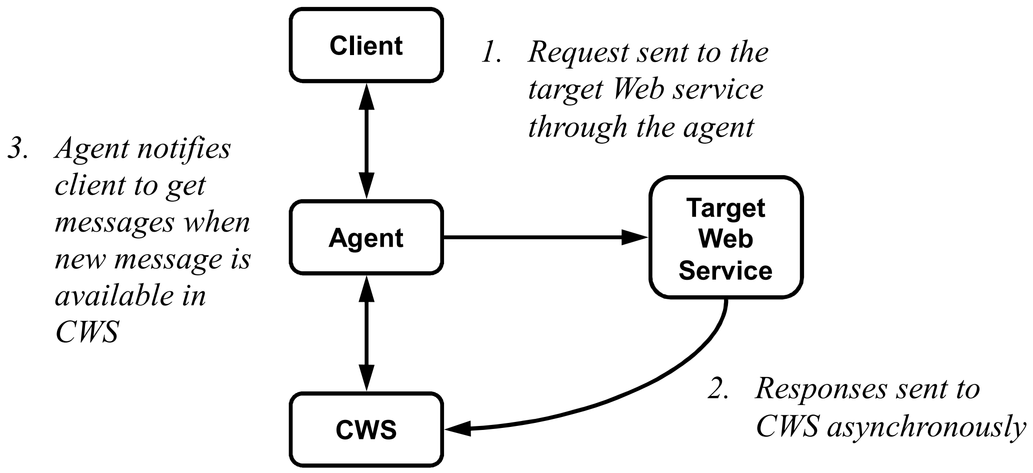

Client
1. Request sent to the target Web service through the agent
3. Agent notifies client to get messages when new message is available in CWS
Agent
Target Web Service
2. Responses sent to CWS asynchronously
CWS

Figure 9. The relationship among the components in the CWS framework.

Figure 9 shows the relationship among the components in the messaging system. A client sends a request to a target Web service, through the intermediate agent, and the responses are sent from the target Web service to the CWS at a later time. The CWS collects responses from the target Web services then returns the responses when requested by the corresponding clients, again through the intermediate agent. This implementation does not require the use of any Web service sub-standard or extension to the current Web service standards. However it creates an agent object in the client framework. In addition to that, a separate socket connection has to be maintained between the agent object and CWS for notification purpose. Because a message from the service provider has to pass through a couple of intermediaries before reaching the client this additional layer of Web services cooperation is not specifically designed for time-critical applications such as multimedia Web services.

3.2 Multimedia Web Services Architecture

3.2.1 QoS Capable Multimedia Web Services

Yu and Lin [9] investigated the use of a quality of service (QoS) broker to make decisions in service selection and service composition in multimedia Web services, which is called the QoS Capable Multimedia Web Services (QCWS) framework. The purposes of the QoS broker include tracking QoS information of servers, making service selection and composition decisions and negotiating QoS agreement between clients and servers.

Figure 10 shows the architecture of the framework. The QoS broker periodically checks the QoS information given by the QoS server. This information is stored locally in the QoS broker so that service selection and composition decisions can be made based on the stored QoS information. Instead of invoking a set of Web service directly between a client and a server the client initially sends functional and QoS requirements to the QoS broker. Upon receiving the requirements the QoS broker devises a service composition plan based on the QoS information collected from the QoS server earlier as well as some dynamic QoS parameters from the servers. At the same time the QoS broker negotiates with the QoS server for a QoS agreement based on the requirement of the client. From the resulting service composition plan the client then follows the plan to invoke the corresponding Web services from the QoS server.

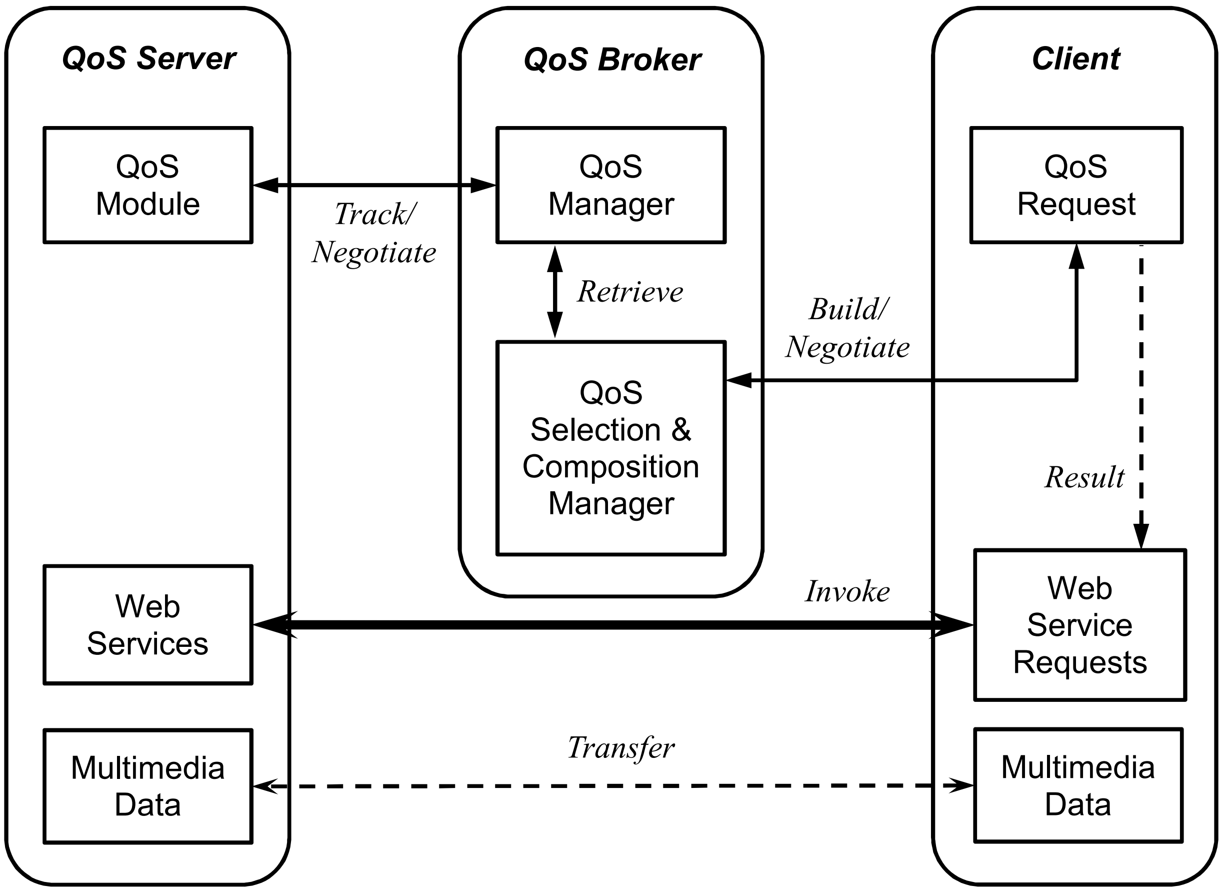


Figure 10. The architecture of the QoS server,

QoS broker and client in QCWS.

In this work, when multimedia data is involved in the output of a multimedia Web services the data is delivered in a separate socket connection. The response of the multimedia Web service includes necessary information such as connection type and port number. The client can then connect to the socket using the returned information. The authors believed that using a socket connection is a reliable way of multimedia data delivery whereas using the current Web service interface is inefficient because of various overheads. However it is unclear on how the output of a multimedia Web service can be forwarded to another multimedia Web service in the composition graph using a socket connection.

3.2.2 SOAP-Oriented Component-Based Framework

Zhang and Chung have proposed a SOAP-oriented component-based framework to support device-independent multimedia Web Services [10][11]. Components in the framework include a service provider, a proxy server and a group of service requesters. The contributions of the framework are an enhancement of the SOAP protocol for multimedia Web Services, the introduction of two intelligent agents at the proxy server and the service provider, and a user profile management using Composite Capability/Preference Profiles (CC/PP) [56], to be explained later.

SOAP is a simple protocol in the sense that it is designed for transferring data which can be serialized into a XML representation. Because of the large size and the binary nature of multimedia content it is difficult to fit the entire content into a SOAP message. It is more practical to separate the big piece of information into multiple SOAP messages. However, SOAP does not provide a mechanism for the organization of a set of related SOAP messages. Therefore in this work an enhancement to the SOAP protocol was proposed in the framework. This enhancement uses an additional first message, which contains only metadata information, when transferring multiple SOAP messages for multimedia content. The metadata information is used for the identification of the transferred messages and for providing information about relationships among the messages.

Listing 10. An example fragment of the first SOAP message in a sequence of messages.

```
<SOAP-ENV:Envelope
  …
  SOAP-ENV:mustSendBack="RequestId0001, Profile001"
  SOAP-ENV:id="Msg00001"
  SOAP-ENV:index="1"
  SOAP-ENV:total="5"/>
  <SOAP-ENV:Body>
    <m:Metadata>
      <component>Msg00001</component>
      <component>Msg00002</component>
      <component>Msg00003</component>
      <component>Msg00004</component>
      <component>Msg00005</component>
    </m:Metadata>
  </SOAP-ENV:Body>
  …
</SOAP-ENV:Envelope>
```

Listing 10 contains an example of a fragment of the first SOAP message within a sequence of multiple SOAP messages. The first SOAP message indicates the metadata of the rest of the SOAP messages. Global attributes are introduced in the SOAP envelope. These attributes identify and give an index to the SOAP message, indicate the total number in the set of messages and give a sequential list of the messages.

The generation and transmission of the set of SOAP messages for multimedia data are handled by two intelligent agents embedded in the proxy server and the service provider respectively. At the service provider, a Multimedia Web Service Server Agent (MWSSAgent) is used to generate a complex result involving multiple multimedia files based on the Web Service request. The result is in the form of the SOAP enhancement described above. At the proxy server, a Multimedia Web Service Agent (MWSAgent) supports the multimedia Web Services for a group of clients connecting to the proxy server. The MWSAgent receives SOAP messages coming from the service provider and distributes these SOAP messages to the connected clients. Each of these clients has a user profile stored in the MWSAgent using CC/PP, a XML based language for specifying device capabilities and user preferences. With these user profiles the MWSAgent can provide adaptation of the multimedia content before it is delivered to the clients. Lastly the MWSAgent acts as a caching manager for the SOAP messages received from the service provider.

3.2.3 Web Service-Oriented Transmoding Service

With regard to the growing set of client configurations and heterogeneous dynamic networks, Decneut et al. have proposed a Web Service-oriented architecture for providing a scalable publication and distribution of multimedia content [12]. The architecture provides transmoding services at the proxy server in a network. Transmoding is a general multimedia adaptation process which alters the bit rate of a multimedia stream by changing the resolution of a frame, changing the frame rate, converting the stream from one format to another or any combination of these. A transmoded multimedia item can take a completely different form than its original. For example, an audio speech can be transmoded into a text transcript of the speech.

The architecture is shown in Figure 11. It has three major components: the transmoding service, the download manager and the broker. When a client requests multimedia content it sends its capabilities via the requesting SOAP message using CC/PP. Upon receiving the request the broker cooperates with the download manager and the transmoding service before sending the response to the client. The broker first requests the download manager to retrieve the required multimedia content from the appropriate multimedia content provider. The download manager acts as an abstraction layer over the variety of protocols used by the content provider. The broker then analyzes the capabilities of the client and determines the formats and descriptions of the transmoding service. The transmoding service can work in two modes. The function of the first mode, called the item transmoder, is to return the requested multimedia items through HTTP when the items are small and they do not span a long duration. The function of another mode, called the stream transmoder, is to stream the multimedia data together with a set of SOAP packaged metadata over TCP or UDP channels. This is useful when the multimedia content is large and requires a long time to be transmitted. During the transmission of multimedia data the client can request a change of capabilities. Based on the changes the broker adjusts the transmoding requirement of the transmoding service in real-time and the multimedia delivery can be adapted to the changes accordingly.

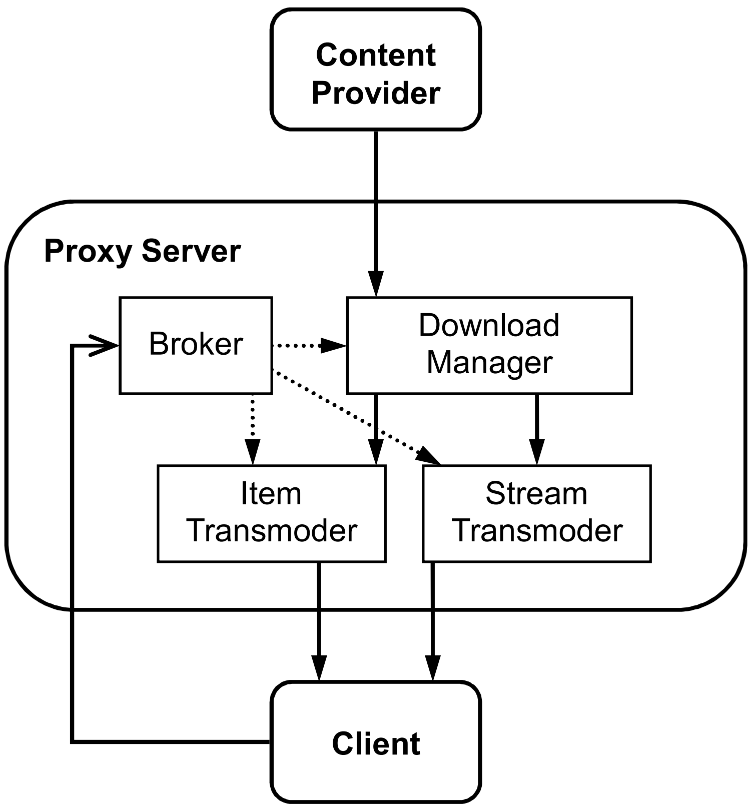


Figure 11. The architecture of the transmoder service.

To transmit binary multimedia data the architecture uses an attachment method, WS-Attachments [57], similar to the one discussed in section 2.1.4. WS-Attachments uses the

DIME protocol, as discussed in section 2.1.4.2, to send binary data as an attachment along with a SOAP message. The construction of the DIME package is similar to the MIME protocol but DIME supports fast lookup and is more compact. The transmoding service in the architecture uses DIME to transmit binary multimedia data between different parties. When the transmoding service is used as an item transmoder attachments are transmitted over HTTP. Otherwise when the transmoding service is used as a streaming service the transmoding service sends DIME attachments together with the SOAP packaged metadata over TCP or UDP. Since the publication of MTOM, as discussed in section 2.1.4.3, the WS-Attachments method has gradually replaced by MTOM.

3.2.4 Personalized Universal Media Access Architecture

Similar to the work discussed in the section 3.2.3, two studies [13][14] looked into the determination of service workflow and system performance in personalized adaptation of multimedia content. Nowadays multimedia content providers such as video-on-demand systems typically deliver multimedia streams to a wide variety of devices. These studies discussed the use of a service-oriented framework for personalized multimedia adaptation and delivery. The framework uses MPEG-21 DIA [58] to specify user preferences and terminal capabilities. Based on this information the framework selects and devises a service workflow where the multimedia stream is processed and transferred from the content provider to the user device. In addition, during the service consumption a monitoring engine closely monitors the QoS offered by the services as well as any failure along the service path. Figure 12 shows the flow of information in the framework. The dotted arrows in the figure indicate Web service flow whereas the solid lines show other data flow such as multimedia data flow. The multimedia data flow starts from the content provider through the adaptation service path until it reaches the client.

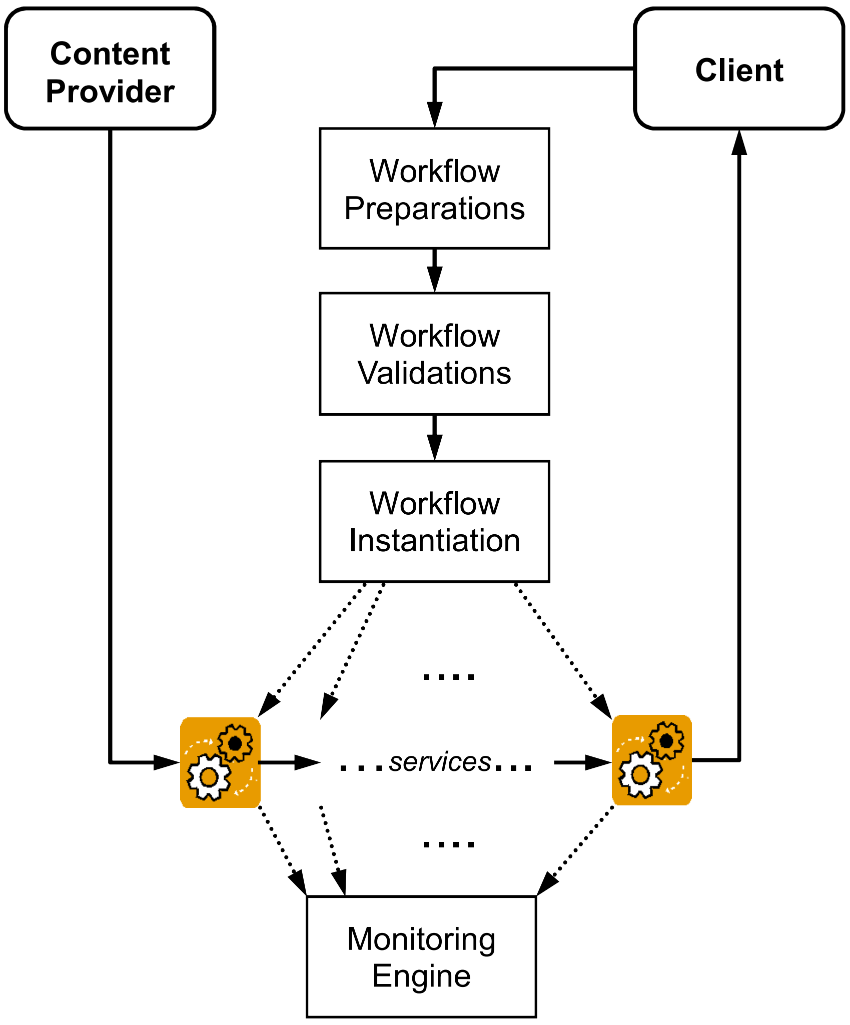


Figure 12. The service-oriented framework for personalized multimedia adaptation.

The framework focuses on the determination and monitoring of the multimedia services under a service-oriented architecture. Due to the "lack of support for continuous data streams in the current Web service infrastructures" as claimed by the authors, the data transfer in the framework is not achieved within the service-oriented framework. The data is streamed via TCP connections between services. The Web services are merely used for information exchange during the workflow instantiation and, service QoS reporting and monitoring.

CHAPTER 4

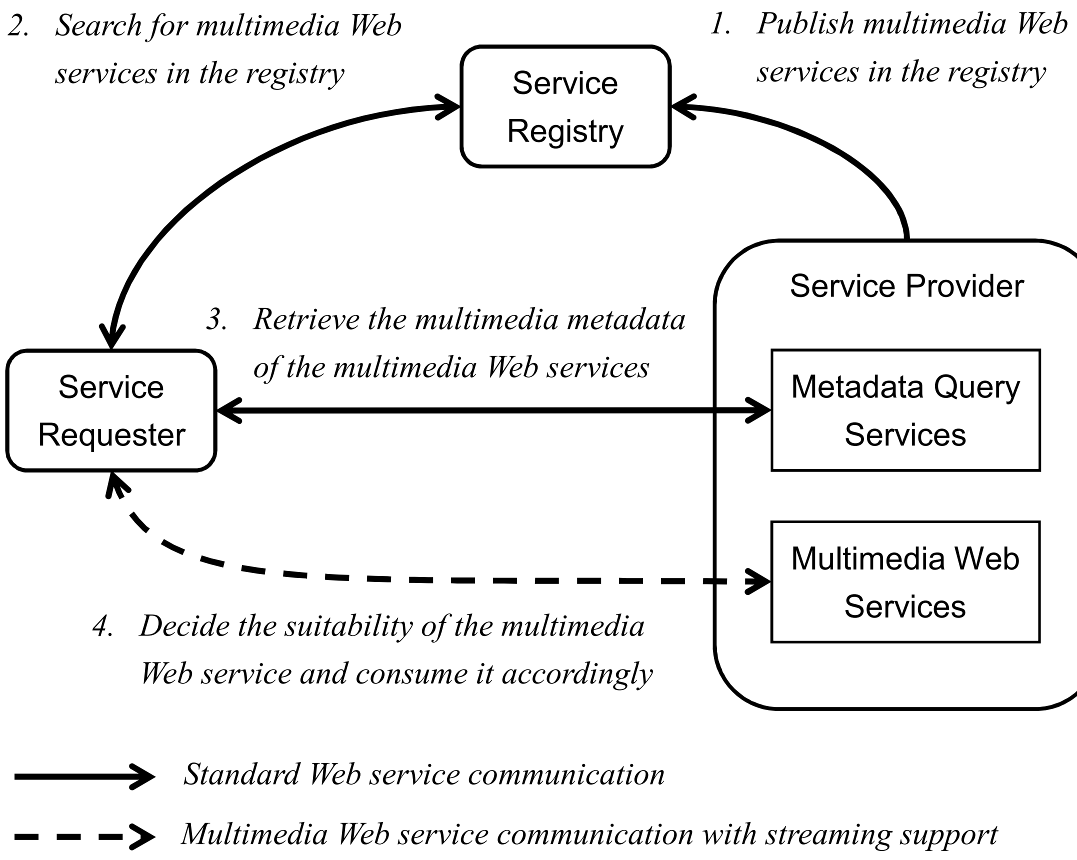
DESIGN OF THE MULTIMEDIA STREAMING WEB SERVICES ARCHITECTURE

4.1 Overview

A multimedia Web service should be able to be published, discovered and consumed like any other Web service. However there is a major difference between a multimedia Web service and an ordinary Web service in the way that the data is transmitted between the two endpoints of a service. A multimedia Web service involves multimedia content. If the size of the multimedia content is small it can be delivered as a binary attachment within the response. However if the size of the content is large it will not be reasonable to transfer it in its entirety within the response. For large multimedia content it is appropriate to have a streaming capability between the two endpoints of the multimedia Web service.

In this thesis a novel multimedia streaming extension is proposed for the Web services architecture. An overview of the flow of the multimedia Web services architecture is shown in Figure 13. Step 1 and step 2 in the figure are responsible for Web services publishing and discovery. Since Web services publishing and discovery can be achieved under current Web services standards they do not require any modification. Step 3 and step 4 are the two major parts of the proposed extension.

First, in the publishing and discovery of a multimedia Web service a description of the input and output of multimedia content has to be supplied by the service provider. This is to ensure that the service consumers understand the expected qualities and information of the content that they need to send to or receive from the provider. In a multimedia Web service this is necessary because different receiving devices may require different delivering qualities. It is highly desirable that a service consumer can select a multimedia Web service with the most suitable qualities for the underlying device. This query of multimedia qualities is achieved by a metadata query service as illustrated in step 3 of Figure 13.



2. Search for multimedia Web services in the registry
1. Publish multimedia Web services in the registry
Service Registry
Service Provider
3. Retrieve the multimedia metadata of the multimedia Web services
Service Requester
Metadata Query Services
Multimedia Web Services
4. Decide the suitability of the multimedia Web service and consume it accordingly
Standard Web service communication
Multimedia Web service communication with streaming support

Figure 13. The flow of the extended Web services architecture.

Second, after selecting the appropriate multimedia Web service the multimedia content has to be transferred between the service provider and the service consumer. In the case of streaming multimedia the two endpoints of the service connection, such as those shown in step 4 of Figure 13, have to be capable of transmitting the streaming multimedia data between themselves. The proposed extension provides a mechanism for the transmission of streaming multimedia between two service endpoints.

In the proposed extension the multimedia streams are sent in their entirety. This does not mean the service requester has to wait for the whole stream before processing its data. On the contrary, the service requester can process any chunk of data while the entire stream is arriving at the destination. This thesis does not include the use of streaming control protocol such as RTSP [59] in the service so that a service requester can do common multimedia control operations such as play, pause, fast forward and rewind. Nevertheless, an application-specific Web service can be added on top of the multimedia Web service, which can be used to control the underlying streaming data transfer transparently. The main shortcoming of this approach is that the Web services are not loosely coupled and this affects the scalability and reliability of the provided services.

4.2 Content Description for Multimedia Web Services

In a multimedia Web service it is important to provide a description of the underlying multimedia content. Information such as the streaming format, compression codec, resolution and frames per second are factors that have to be considered when a service consumer attempts to select the best possible multimedia service among a set of published services. In addition, during service composition, the multimedia information is required in the determination of the service workflow. For example, an additional service such as a transcoding service may be required when the ideal multimedia Web service in terms of the desired format and qualities is not available.

Listing 11. An example of part of a MPEG-7 metadata for a video. The example shows the video frame information.

```
<?xml version="1.0"?>
<Mpeg7 xmlns="urn:mpeg:mpeg7:schema:2001">
  <Description xsi:type="ContentEntityType">
    <MultimediaContent xsi:type="VideoType">
      <MediaFormat>
      . . .
        <VisualCoding>
          <Pixel aspectRatio="0.75" bitsPer="8"/>
          <Frame height="288" width="352" rate="25"/>
        </VisualCoding>
        . . .
      </MediaFormat>
    </MultimediaContent>
  </Description>
</Mpeg7>
```

Multimedia content can be described by MPEG-7 [6]. MPEG-7 metadata contains a wide range of XML-based languages for the description of different aspects of multimedia data. For example, Listing 11 shows part of MPEG-7 metadata which describes the frame format of a video stream. Since MPEG-7 is an XML-based language it can be easily inserted in the Web services stack. In the proposed extension a simple method is provided to query the description of the multimedia Web service using an extended WSDL of the service.

4.2.1 Metadata Query Services

MPEG-7 metadata can be used to describe the input and output multimedia content of a multimedia Web service. A straightforward approach is to insert the metadata inside the description of a WSDL operation. However, MPEG-7 metadata is relatively large compared to the size of a WSDL file. Therefore the size of a WSDL file will increase significantly if MPEG-7 metadata are inserted. In addition, the multimedia content description may vary from time to time. If an item of metadata is changed the corresponding Web service has to review and publish its WSDL to all related service registries.

Listing 12. An example WSDL description of the operation 'GetMedia' showing an associated metadata query service called 'GetMediaDesc'.

```
<?xml version="1.0"?>
<wsdl:definitions name="MediaService"
  targetNamespace="http://example.org/media"
  xmlns:media="http://example.org/media"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:mmws="http://schemas.xmlsoap.org/wsdl/mmws"
  xmlns:xlink="http://www.w3.org/1999/xlink">
  . . .
  <wsdl:portType name="GetMediaPortType">
    <wsdl:operation name="GetMediaDesc">
      . . .
    </wsdl:operation>
    <wsdl:operation name="GetMedia">
      <wsdl:input message="media:GetMediaRequest"/>
      <wsdl:output message="media:GetMediaResponse"/>
      <wsdl:fault message="media:GetMediaFault"/>
      <mmws:metadata operation="media:GetMediaDesc" />
    </wsdl:operation>
  </wsdl:portType>
  . . .
</wsdl>
```

Instead of including the entire MPEG-7 metadata an accompanying metadata query service is provided for each operation of a multimedia Web service. The sole purpose of the metadata query service is to provide a channel for the distribution of the metadata associated with the multimedia Web service. For example, Listing 12 shows a description of an operation 'GetMedia' together with the specification of its metadata query service 'GetMediaDesc'. When a service consumer sees the query service, it can ask for the information of the multimedia content and hence choose a service based on the received information. Furthermore, the metadata query service gives decision makers such as service registries and service brokers the ability to devise appropriate algorithms so that multimedia Web services can be selected based on the information of the multimedia content.

4.3 Streaming Data Transfer

In this chapter, an approach for the transmission of streaming data in the Web services architecture is presented. At the moment the Web services standards do not support the transfer of streaming data between two endpoints of a Web service. A typical information exchange sequence involves one endpoint sending a request and then another endpoint replying by sending a corresponding response. Each of the requests and responses are enclosed in a single message which is commonly a SOAP message. However, streaming data, by definition, is a stream of relatively small data packets. That means a single request or response is not sufficient for transferring streaming data.

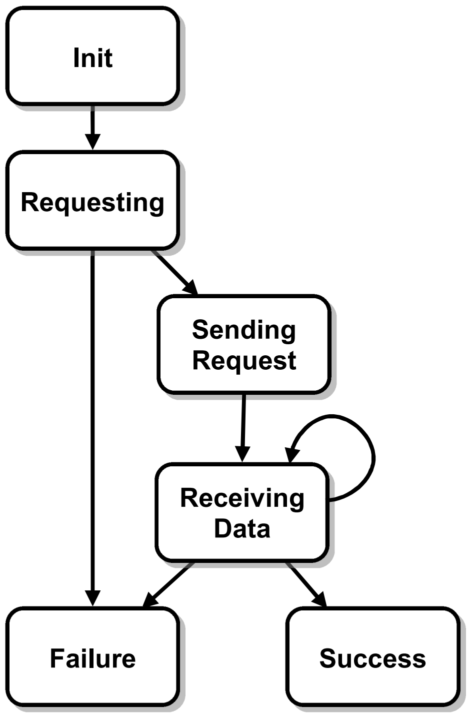
In the proposed extension the streaming data transfer is accomplished by refining the approach described in [60] and [61]. This involves extending two current Web services standards: the message exchange pattern (MEP) and the SOAP binding of a MEP. The way that Web service messages are exchanged between two service endpoints is controlled by a MEP. At the moment no existing MEP supports a stream of data to be transmitted between the endpoints. In order to add support for this, two new types of MEP are created to handle streaming data. These new MEPs can then be used via a communication protocol such as UDP or TCP. In particular, this thesis considers the application of the new MEPs to HTTP, the most commonly used protocol for Web services.

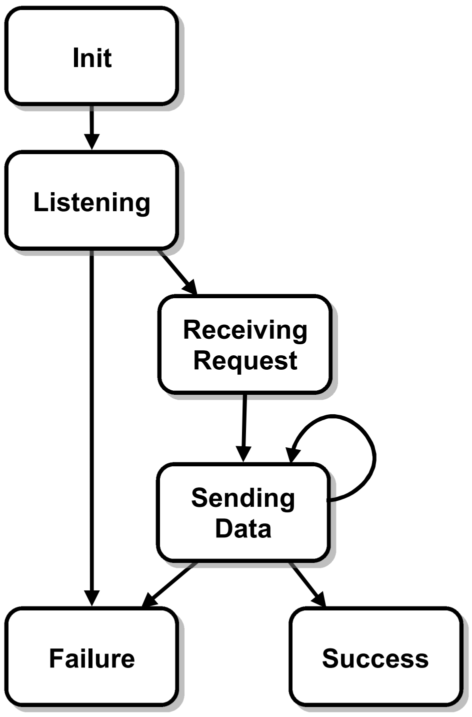
4.3.1 Message Exchange Patterns

As described in section 2.1.2, there are currently two MEPs defined in the SOAP specification, the request-response MEP and the response MEP. Both of these MEPs use a single request and a single response structure. This is not sufficient for the transfer of streaming data because multiple multimedia packets are required in a single stream of data. In this study two new MEPs are created for two types of multimedia streaming Web services. A single request-multiple response (SRMR) MEP caters for a Web service delivering streaming data in response to a single request. A multiple request-multiple response (MRMR) MEP is created for a Web service receiving a stream of data as input and delivering another stream of data as output.

4.3.1.1 Single Request-Multiple Response (SRMR) MEP

Figure 14 presents the state transition diagram of a SRMR MEP. The difference in the new MEP compared to the request-response MEP is the way that data is transferred after communication is established. In the SRMR MEP the requesting SOAP node initializes the operation by making a connection to the responding SOAP node. After the request is transmitted from the Sending Request state in the requesting SOAP node to the Receiving Request state in responding SOAP node, both nodes are locked in looping states in which the streaming data transfer takes place. The looping states, which are called the Receiving Data state and the Sending Data state, represent the transfer of streaming data in the SRMR MEP. Each chunk of data is transmitted and received separately and continuously between the two nodes. This process lasts until the end of the streaming data transfer.



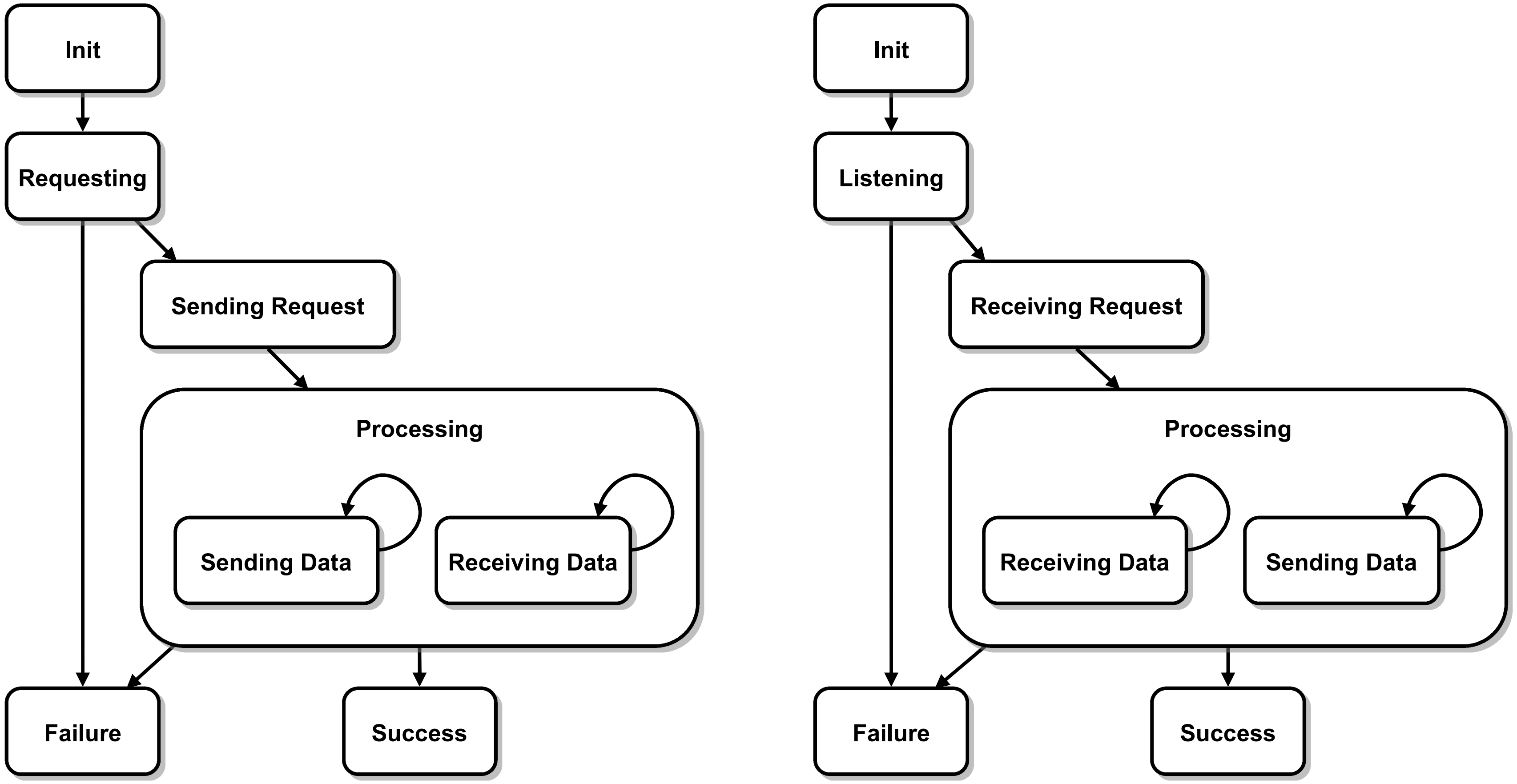


The requesting SOAP node **The responding SOAP node**

Figure 14. The state transition diagram of the Single Request-Multiple Response (SRMR) MEP.

4.3.1.2 Multiple Request-Multiple Response (MRMR) MEP

In the previous MEP shown in Figure 14 the requesting node sends a single request to the responding node and then receives a continuous stream of data. In the MRMR MEP, although a single request is made by the requesting node the request body consists of the request information plus a stream of data, which can be a multimedia data stream, instead of a single chunk of information. To create a state diagram for this MEP the request is split into the request information and its associated data stream as shown in Figure 15. In the diagrams displayed in Figure 15 the Processing states replaces the Receiving Data state and the Sending Data state of Figure 14 for the MRMR MEP. The processing states handle the continuous transmission of the streaming data to and from the SOAP nodes. Within the 'Processing' state the sending and receiving processes occur in parallel. They can run asynchronously or one after another depending on the nature of the provided Web service.



The requesting SOAP node **The responding SOAP node**

Figure 15. The state transition diagram of the Multiple Request-Multiple Response (MRMR) MEP.

4.3.2 SOAP HTTP Binding

In section 4.3.1 two new MEPs are defined for multimedia Web services. These MEPs define the exchange pattern between two SOAP endpoints. They are abstract descriptions of the information exchange sequence. The implementation of a MEP in a Web service program is described by a SOAP binding. A MEP is not restricted to be put in a certain software or network protocol. However, the only SOAP binding provided by the Web services standards is the SOAP HTTP binding, which is described in section 2.1.3, and therefore it is the most commonly used protocol in the implementation of Web services.

In the default SOAP HTTP binding for the request-response MEP a SOAP message is put in the HTTP entity body in each of the requests and responses of the service. These two SOAP messages contain the request and the corresponding response of a Web service. A SOAP message in either the request or response can only be used to store one single item although the item can be of any type such as a number or a data record. However, as shown in Figure 14 and Figure 15 streams of multimedia data are sent and/or received by the SOAP nodes in the newly created MEPs. That means using a single SOAP message is not sufficient for handling these two MEPs. The following sections first describe the packaging method of the data packets. Then new SOAP HTTP bindings are introduced for transferring streaming data in the SRMR MEP and the MRMR MEP.

4.3.2.1 SOAP Packaging for a Multimedia Stream

Multimedia streaming is a continuous transmission of data packets, which are the basic units of transmission. To transfer a multimedia stream from one SOAP endpoint to another the stream has to be encapsulated in some form of SOAP packaging. Since each data packet represents one piece of information within the stream a SOAP message is used to enclose one single packet during the transmission. To do that two issues have to be resolved.

First, the data packet enclosed by a SOAP message requires a way to identify itself. This is because a SOAP message can be transmitted in a variety of bindings and therefore SOAP messages can arrive at a SOAP endpoint asynchronously. If a SOAP message can be identified the SOAP processor will be able to forward the SOAP message to an appropriate handler. In this work the mechanism of identification is achieved by using Web Services Addressing (WS-Addressing), which is described in section 2.1.6. WS-Addressing allows a SOAP message to carry header information such as the destination of the message and the identity of the data packet.

Second, SOAP is an XML-based protocol. That means binary data, for instance multimedia packets, cannot be directly inserted into the body of the message. Nevertheless, binary data can be included in a SOAP message using binary-to-text encodings such as base64 encoding. The disadvantage is that a binary-to-text encoding brings a significant increase in data size and processing time. Alternatively binary data can be transmitted together with a SOAP message using the SOAP Message Transmission Optimization Mechanism (MTOM). MTOM is a standard serialization method which makes use of the MIME Multipart/Related package and the XML-binary Optimized Packaging (XOP). MTOM has been discussed in section 2.1.4.2.

Listing 13 shows a SOAP message with WS-Addressing information and its data packet included in a MTOM MIME package. In this example the identity of the packet is ‘data5608@example.org’, the related message, which can be the request message of the multimedia stream, is ‘msg2056@example.org’. The associated action of the Web service is ‘http://example.org/getmedia’.

Listing 13. An example MTOM serialization of a data packet of a multimedia stream. WS-Addressing is used to indicate the destination and the identity of the packet.

```
Content-Type: Multipart/Related; boundary=data_boundary;
 type="application/xop+xml"

--data_boundary
Content-Type: application/xop+xml; charset=UTF-8;
 type="application/soap+xml"
Content-Transfer-Encoding: 8bit
Content-ID: msg5608@example.org

<env:Envelope
  xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:wsa="http://www.w3.org/2005/08/addressing">
  <env:Header>
    <wsa:MessageID>data5608@example.org</wsa:MessageID>
    <wsa:RelatesTo>msg2056@example.org</wsa:RelatesTo>
    <wsa:Action>http://example.org/getmedia</wsa:Action>
  </env:Header>
  <env:Body>
    <media:packet xmlns:media="http://example.org/media">
      <xop:Include
        xmlns:xop="http://www.w3.org/2004/08/xop/include"
        href="cid:data5608@example.org"/>
    </media:packet>
  </env:Body>
</env:Envelope>

--data_boundary
Content-Type: application/octet-stream
Content-Transfer-Encoding: binary
Content-ID: data5608@example.org

... Packet data...

--data_boundary--
```

4.3.2.2 SOAP HTTP Binding for the SRMR MEP

In contrast to the request-response MEP, the SRMR MEP sends the response from the responding SOAP node to the requesting SOAP node as a stream of data instead a single item. This stream of data is transmitted at the looping state in the responding node in Figure 14.

In the case of the request-response MEP its SOAP HTTP binding transmits the response as a SOAP message in the body of a HTTP response. If a multimedia stream with multiple items is to be continuously transmitted along a HTTP response the SOAP HTTP binding will need to be modified. In this study a MIME Multipart/Related package is used to put the multimedia stream in a HTTP response body. The result is shown in Listing 14.

Listing 14. A sequence of SOAP messages encoded using the MIME Multipart/Related content type in a HTTP response. The content of SOAP messages is not shown.

```
HTTP/1.1 200 OK
Content-Type: Multipart/Related; boundary=packet_boundary;
 type="Multipart/Related"

--packet_boundary

...SOAP message 1...

--packet_boundary

...SOAP message 2...

--packet_boundary

...
```

Using this organization the response can carry as many SOAP messages as required. The requesting SOAP node is then responsible to read each SOAP message continuously when the

SOAP messages are received from the responding node. This is indicated in the ‘Receiving’ state of the requesting SOAP node in Figure 14.

4.3.2.3 SOAP HTTP Binding for the MRMR MEP

The MRMR MEP involves two channels of data streaming. One is from the requesting SOAP node to the SOAP responding node and another one in the opposite direction. The SOAP HTTP binding for the SRMR MEP described in the previous section allows a data stream to be received by a SOAP endpoint in a HTTP response. The same technique can be applied to the SOAP HTTP binding for the MRMR MEP. That means the transmission of data when the responding SOAP node sends the response uses the MIME Multipart/Related package. The remaining part of the SOAP HTTP binding for the MRMR MEP is then how to transmit streaming data when the request is made.

In the request-response MEP the request is sent via the HTTP request body. If the request involves multiple items, these items can be put in the HTTP request body using the MIME Multiple/Related package. Since HTTP is a synchronous protocol the HTTP process has to receive fully the HTTP request before sending the HTTP response. This poses problems for multimedia Web services because the SOAP node may have to wait for a long time before starting to send out the response. For example, with reference to the transcoding service shown in the example scenario in Figure 1, it is not desirable for the handheld device to wait for the entire duration of the multimedia stream before it can receive a transcoded version of the stream.

As mentioned in section 4.3.2.1, WS-Addressing forms part of the information carried by the SOAP message of a data packet in a multimedia stream. Using the information given by the WS-Addressing header an asynchronous transfer mechanism can be employed when transmitting streaming data from the requesting SOAP node to the responding SOAP node. The sequence diagram of the mechanism between the two SOAP nodes is illustrated in Figure 16. The Processing states in the MRMR MEP are then handled by the Sender thread and Receiver thread in Figure 16.

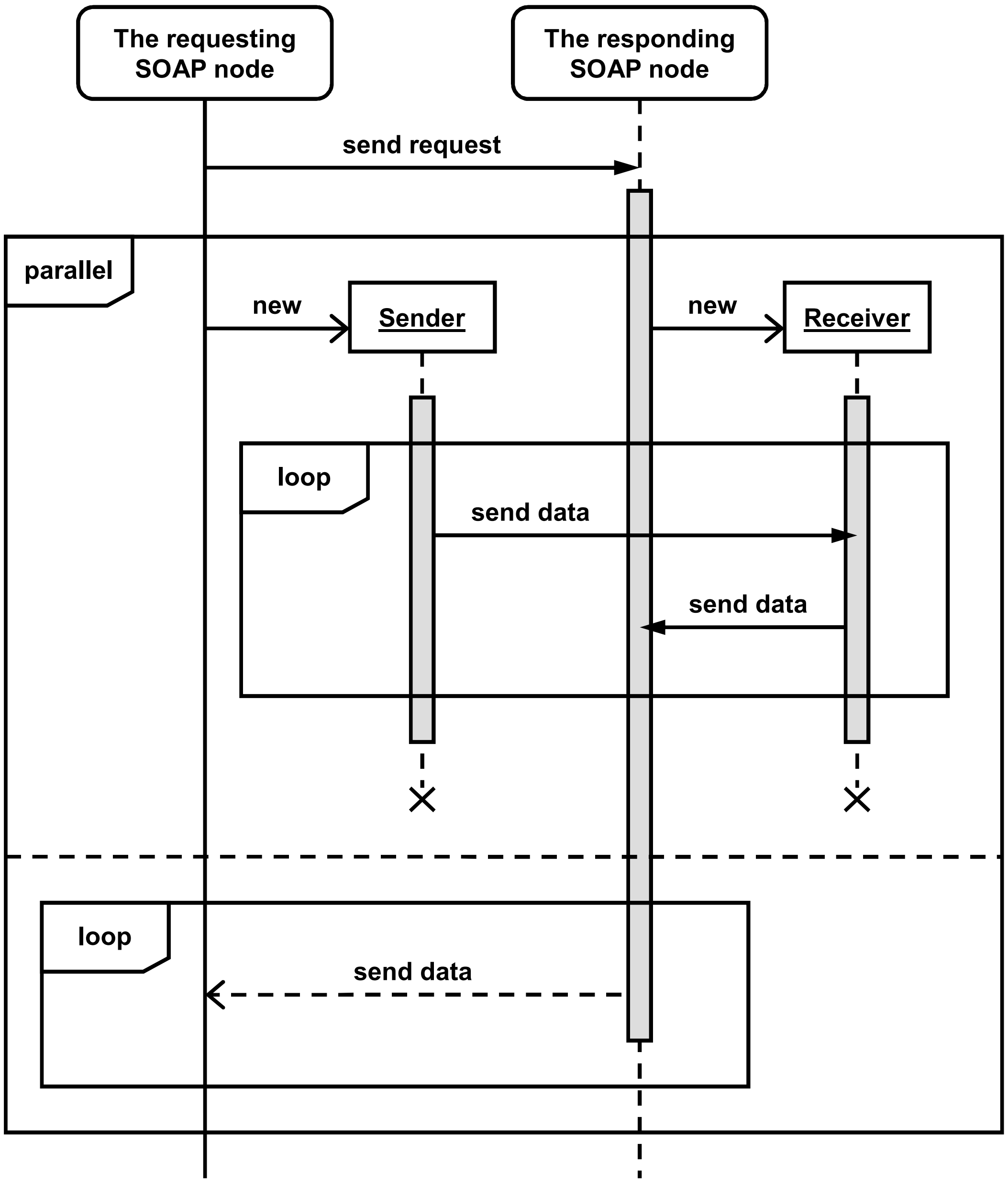


Figure 16. The sequence diagram of the asynchronous process of sending and receiving streaming data between the requesting SOAP node and the responding SOAP node.

CHAPTER 5

COMPRESSION OF SOAP MESSAGES

The SOAP HTTP binding described in section 4.3.2 introduces a considerable amount of transfer overhead to the multimedia transfer. The overall structure of the streaming data transmission mechanism described in section 4.3.2.2 for the SRMR MEP, in the case when MTOM is used, is shown in Figure 17. A first level of MIME Multipart/Related structure contains the SOAP messages of the data packets. For each SOAP message a second level of MIME Multipart/Related structure is used to separate the textual SOAP message and the binary attachment of the message. For each of the MIME packages headers are required to specify information such as the length and the type of content associated with the package. Additional transfer overhead, compared to the original multimedia stream, is also required when each data packet is transmitted together with a SOAP message.

To improve the efficiency in the multimedia transmission in the proposed extension the use of various compression techniques is investigated in this chapter. First a slightly modified version of the A-SOAP compression method is introduced. The original A-SOAP compression method has been introduced in section 2.3.3. Then the performance of various compression methods when used in the multimedia Web services architecture is compared. The goal of the comparison is to find a suitable compression method to be incorporated into the proposed extension.

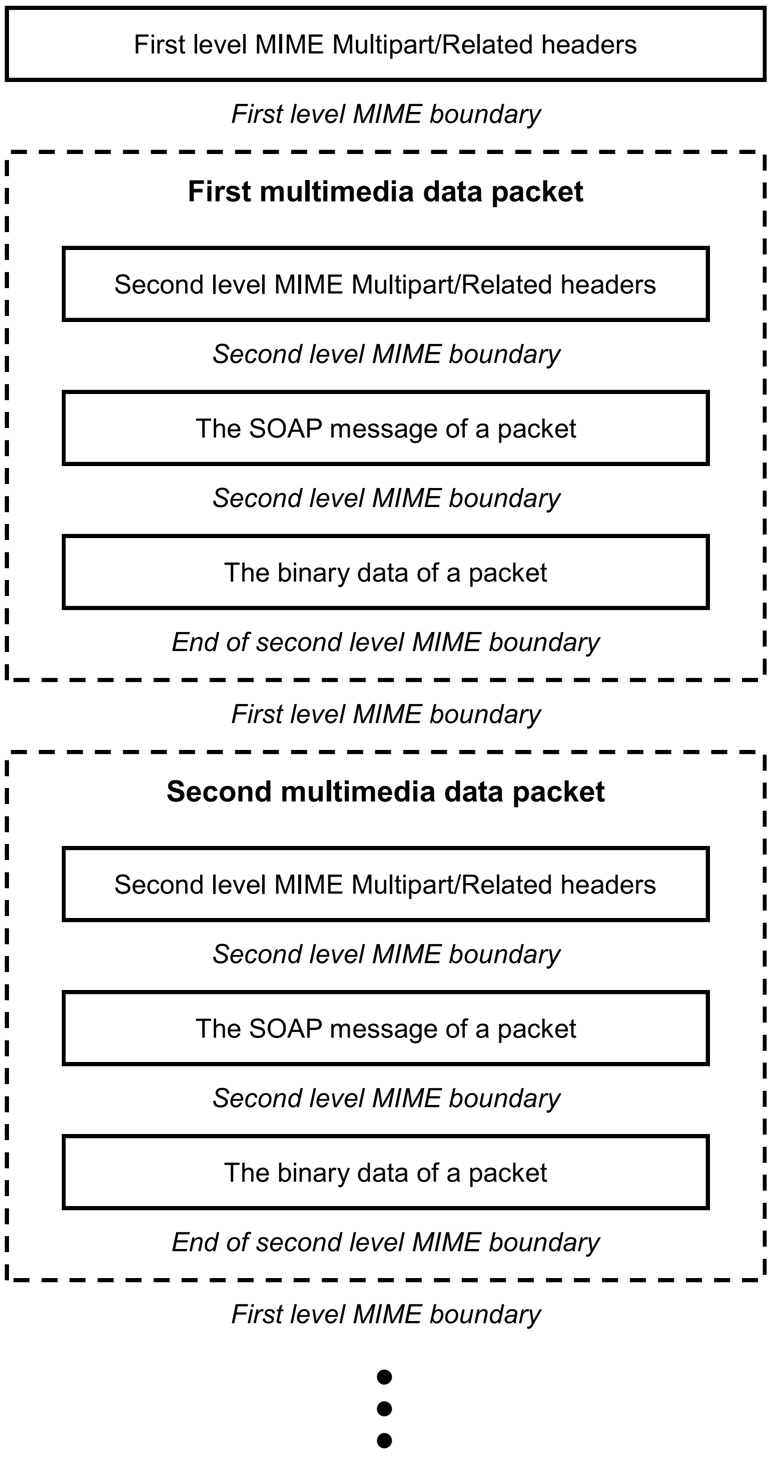

First level MIME Multipart/Related headers
First level MIME boundary
First multimedia data packet
Second level MIME Multipart/Related headers
Second level MIME boundary
The SOAP message of a packet
Second level MIME boundary
The binary data of a packet
End of second level MIME boundary
First level MIME boundary
Second multimedia data packet
Second level MIME Multipart/Related headers
Second level MIME boundary
The SOAP message of a packet
Second level MIME boundary
The binary data of a packet
End of second level MIME boundary
First level MIME boundary

Figure 17. The SOAP response structure of the streaming data transmission mechanism in the SRMR MEP. Only the first two data packets are shown here.

5.1 A Modified A-SOAP Compression Method

As discussed in section 2.3.3, A-SOAP is a dictionary-based compression technique for SOAP message exchange between two SOAP endpoints. A dictionary is used to store open tags and close tags. This dictionary is built up incrementally and referred to during the transmission. By transmitting only dictionary indexes of repetitive tags, the compression scheme can reduce the message size to 50% of the original size on average.

The dictionary structure in A-SOAP stores open tags and close tags. If two open tags having the same tag name are associated with different attributes these two tags will be considered as two different patterns. Therefore these two open tags are stored in two separate entries in the dictionary. This creates a problem for SOAP messages that change the same attribute values frequently. The example SOAP message given in Listing 13 contains the href attribute that points to the corresponding data attachment. This attribute value keeps changing between different packets and thus the A-SOAP compression method is not helpful in this situation.

In this section, a modified A-SOAP method is proposed. The original A-SOAP method classifies two open tags with different attribute values as two different patterns in the dictionary. In the modified A-SOAP method, these open tags are considered the same entry. That means they point to the same dictionary index and therefore if they appear one after another during the communication, the latter one will be transmitted as an index only.

```
<staff id="50001">
 <name>John</name>
</staff>
```

(a) SOAP message 1

```
<staff id="90116">
 <name>David</name>
 <room>5210</room>
</staff>
```

(b) SOAP message 2

Original A-SOAP method

Index	Entry
0	<staff id="50001">
1	<name>
2	</name>
3	</staff>

Modified A-SOAP method

Index	Entry
0	<staff id="50001">
1	<name>
2	</name>
3	</staff>

(c) The dictionaries after SOAP message 1 is processed by the original A-SOAP method (on the left) and the modified A-SOAP method (on the right). These two dictionaries are the same at this stage.

Original A-SOAP method

Index	Entry
0	<staff id="50001">
1	<name>
2	</name>
3	</staff>
4	<staff id="90116">
5	<room>
6	</room>

Modified A-SOAP method

Index	Entry
0	<staff id="50001">
1	<name>
2	</name>
3	</staff>
4	<room>
5	</room>

(d) The dictionaries after both SOAP message 1 and SOAP message 2 are processed by the original A-SOAP method (on the left) and the modified A-SOAP method (on the right)

Figure 18. The dictionary snapshots of the original A-SOAP method and the modified A-SOAP method.

(a) is the first SOAP message to be processed,

(b) is the second SOAP message to be processed,

(c) shows the dictionaries of the original A-SOAP method, shown on the left, and the modified A-SOAP method, shown on the right, after processing SOAP message 1, and

(d) is the dictionaries of both methods after processing SOAP message 2.

Figure 18 shows the dictionary snapshots of the original A-SOAP method and the modified A-SOAP method of an example. This example processes the two SOAP messages shown in Figure 18(a) and Figure 18(b) in the same order. Figure 18(c) shows snapshots of the dictionaries after both methods have processed the first SOAP message. Both of them have the same dictionary content after this step. Then the second SOAP message is processed by both methods. The result is shown in Figure 18(d). The original A-SOAP method produces three additional dictionary entries whereas the modified A-SOAP method adds only two new dictionary entries. The reason for this difference is that the modified A-SOAP method considers the ‘staff’ open tag in the second SOAP message as the same pattern as the ‘staff’ open tag in the first SOAP message. Therefore the open tag appears only once in the dictionary, as shown in the dictionary on the right hand side of Figure 18(d).

Based on the dictionary content a SOAP message is encoded with two types of data. First, any repeated tags are represented using the dictionary indexes. Second, open and close tags which are not in the dictionary and text content of the SOAP message are transmitted without any modification. The dictionary indexes, in the original A-SOAP method, are represented by a specific invalid UTF-8 byte sequence [62]. For example, for a dictionary with 128 entries, the byte sequence of 1100000x followed by 10xxxxxx is used to represent an index value. However, the index value alone is not sufficient in the modified A-SOAP method used in this study. This is because a repeated open tag with different attribute values can only be reproduced by: 1) the dictionary index of the same pattern; 2) any attribute values that are not the same as the values stored in the dictionary entry.

For example, if we want to encode the SOAP message in Figure 18(b) using the dictionary shown on the right hand side of Figure 18(d), the first open tag ‘<staff id="90116">’ can only be represented by a dictionary index of 1 as well as the information required to change ‘50001’ to ‘90116’. Certainly one UTF-8 sequence is not adequate in this situation. The encoding is modified as follows. First, an attribute flag is used to indicate whether an attribute modification is required for the open tag. This flag can be put at the least significant bit of the first byte of the dictionary index. Changing the usage of this bit reduces the number of dictionary entries by half. However, if an increased number of dictionary entries is required, the number of UTF-8 bytes can be increased to three or four bytes. Considering to the first byte of the dictionary index again, if the attribute flag is set to 1, two additional UTF-8 byte sequences are added after the dictionary index. These two UTF-8 byte sequences correspond to the position and length of the attribute. The position byte sequence indicates the sequential order of the attribute to be modified. Because there can be more than one attribute modification the least significant bit of the first UTF-8 byte of the position byte sequence is used as an attribute flag. After specifying the position, the length byte sequence contains the length of the new attribute value. Finally the new attribute value is appended. The attribute modification process will be repeated if the attribute flag in the position byte sequence is set to 1. An example of an encoded representation of the tag ‘<staff id="90116">’ is shown in Figure 19.

As shown in the encoded representation of the open tag, the first attribute flag is 1, shown as bold, at the dictionary index. Setting to the attribute flag to 1 means an attribute has to be modified in the open tag. The upcoming byte sequences after the dictionary index are the information of the modified attribute. In the attribute position following the dictionary index, the attribute flag is 0, also shown as bold. That means, in this example, the encoded representation is completed after modifying this single attribute value.

An open tag to be encoded

<staff id="90116">

Encoded representation of the open tag

1100000**1** 10000000	(dictionary index)
1100000**0** 10000000	(attribute position)
11000000 10000101	(attribute length)
"90116"	(attribute value)

Current dictionary before encoding

Index	Entry
0	<staff id="50001">
1	<name>
2	</name>
3	</staff>
4	<room>
5	</room>

Figure 19. An example representation of an open tag using the modified A-SOAP method in this study with the dictionary shown on the right.

Let us consider the situation where we have to transmit the SOAP message shown in Listing 13. We assume that all tags of the SOAP message have been stored in the dictionary, albeit with different attribute values in the href attribute. If we apply the modified A-SOAP method to the SOAP message using the dictionary, the size of the encoded representation is only 167 bytes (that is, twelve encoding of open or close tags and one attribute modification of the href attribute). That is almost 70% reduction in the SOAP message size.

Using this modified A-SOAP method any changes in attribute value would not result in a new dictionary entry. Therefore, greater compression can be achieved by reusing data stored in the dictionary. This gives significant savings in data size for the streaming data transmission.

5.2 Performance of Compression Techniques

In this section the performance of a selected set of compression techniques are evaluated when they are applied to the structure proposed in section 4.3.2. The result is shown in Table 2. The packet size is assumed to be the size of the payload when the packet is transmitted in a TCP connection using a MTU of 1500 bytes. The overhead size as a percentage of the packet size is shown in the last column of the result. This overhead size is not calculated solely based on the compression performance of the compression techniques. The evaluation is based on the compression applied to the transmission of a single packet, which includes the compressed SOAP message, the packet data and various MIME headers. The first two rows show the result when no compression is used on a base64 encoded message and a MTOM package. For all compression techniques, apart from the two binary XML methods listed at the bottom two rows of the table, the transmission of the packet data is achieved using MTOM. Because of this the total overhead size for those compression techniques includes the MTOM header size as illustrated in Listing 13. For binary XML methods they can directly encode base64 encoded content in their binary form. They do not have MTOM headers. Thus the MTOM header size is not included in the calculation of their overhead size.

By comparing the overhead size shown in the last column of Table 2 it is clear that the binary XML methods are significantly better than the other compression techniques. Although the SOAP-specific compression techniques perform better than the general XML compressors, especially the modified A-SOAP method, all of them still have to carry the MTOM headers. These MTOM headers occupy a major portion of the transfer overhead. Without the need to have the MTOM headers the binary XML methods, EXI in particular, are the obvious choices in the proposed extension to support compression in the transmission. Another advantage of using the binary XML encoding scheme is the improvement in processing time of the SOAP messages. In a typical Web service SOAP messages have to be constructed and parsed at both ends of the communication. This contributes to the total processing time of a Web service. When a binary XML scheme, such as EXI, is used the SOAP processing time can be improved by multiple times [63].

Table 2. **Performance of compression techniques per packet for the SOAP packaging of a multimedia stream.**

Compression technique [a]	Packaging method for binary attachment	HTTP MIME header size	MTOM MIME header size	SOAP message size after applying compression	Packet size	Total overhead size	Total overhead size as a percentage of packet size
Packet data transmitted using base64 encoding without compression	Base64 encoding	19	-	2253 [b]	1472	1042	70.8%
Packet data transmitted using MTOM without compression	MTOM	19	404	536	1472	959	65.1%
Gzip	MTOM	19	404	288	1472	711	48.3%
PPM	MTOM	19	404	248	1472	671	45.6%
Bzip2	MTOM	19	404	311	1472	734	49.9%
XMill [c]	MTOM	19	404	281	1472	704	47.8%
XMLPPM	MTOM	19	404	275	1472	698	47.4%
Differential encoding [d]	MTOM	19	404	213	1472	636	43.2%
A-SOAP [e]	MTOM	19	404	233	1472	656	44.6%
Modified A-SOAP [e]	MTOM	19	404	167	1472	590	40.1%
Fast Infoset [f]	Binary XML	19	-	1615 [g]	1472	162	11.0%
EXI [f]	Binary XML	19	-	1547 [g]	1472	94	6.4%

[a] *All compression techniques use the best available compression rate.*

[b] *The SOAP message size includes the size of the packet data which is encoded using base64 encoding.*

[c] *The backend compressor for XMill is PPM.*

[d] *The difference between the SOAP message and its WSDL-generated skeleton message is specified using the Delta Update Language (DUL). The DUL result is then compressed using XMLPPM.*

[e] *The result assumes the dictionary has already been constructed when the SOAP message is transmitted.*

[f] *Both binary XML techniques are optimized using the relevant XML schemas of the SOAP message.*

[g] *The SOAP message size includes the size of the packet data without enabling compression.*

CHAPTER 6

EXPERIMENTS AND RESULTS

6.1 Experimental Setup

In 4.3 two types of multimedia streaming transfer are described. The SRMR MEP has a single object input and a multimedia streaming output whereas the MRMR MEP has multimedia streaming capability in both input and output. While time-criticality is typically associated with viewing streaming multimedia content streaming multimedia input is more tolerable to delay. This is because it is the service provider waiting for the streaming data instead of the service consumer which means the user. Therefore, in this thesis the evaluation of the extension is focused on the multimedia streaming performance at the output of a multimedia Web service.

It is generally accepted that TCP is not the most appropriate protocol for the transfer of streaming multimedia content because of the reliability and congestion control mechanisms of TCP [64]. Nevertheless, due to the popularity of TCP on the Internet it has been the most widely used protocol for multimedia streaming. It is shown in [65] that the performance of using TCP for multimedia streaming is generally acceptable when the achievable TCP throughput is twice the video bitrate, with a few seconds of startup delay. Because Web services typically run via HTTP all experiments in this work are based on HTTP, which is run on top of TCP. In general, when running Web services on HTTP the size overhead of the SOAP envelopes and the need for various headers such as the HTTP headers are not favorable for time-critical applications. Studies have remained critical of using Web services for multimedia streaming applications [9][66]. Because of this, a set of experiments has been devised and executed to highlight the performance of the proposed extension for multimedia streaming. Results show that, using the extended architecture together with binary XML, performance is comparable to multimedia streaming using HTTP.

6.1.1 Simulation Setup

The streaming multimedia transmission was simulated using ns [67]. The simulation was carried out on a machine with an Intel Core i7 quad-core 2.0Hz processor and 8GB ram. The operating system used was Windows 7 but the ns program was compiled and executed inside Cygwin. However the simulation results were not dependent on the hardware setup because the ns simulator is a discrete event network simulation tool.

The simulation examined the performance of the proposed Web services transmission for streaming multimedia content amidst a background of real world traffic. The network topology used for the experiments was the simple single bottleneck network shown in Figure 20.

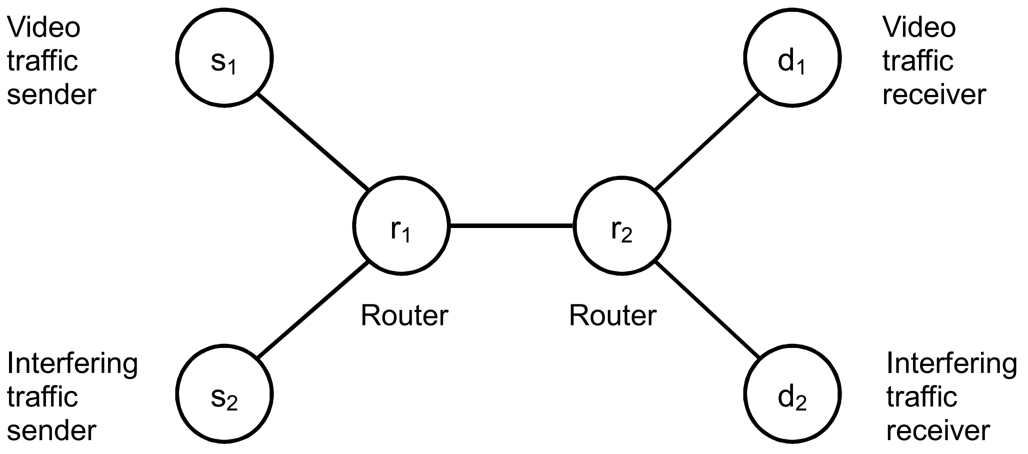


Figure 20. The simple single bottleneck network used in the multimedia streaming simulation.

There are six nodes in the network. s_1 and s_2 are the data sources. d_1 and d_2 are the data sinks. They are connected to r_1 and r_2, which are the routers of the network. Between these two routers is the single link that forms the bottleneck of the network. There are two traffic flows in the network. s_1 generates traffic that is received by d_1 and the same applies for s_2 and d_2. Between s_1 and d_1 is the streaming multimedia flow. In this study a streaming video transmission is used as the multimedia flow. s_1 is the source of the streaming video while d_1 is the receiving end of the video. The traffic between s_2 and d_2 is the interfering traffic. The links connecting the data sources and data sinks to the routers, s_1 to r_1, s_2 to r_1, d_1 to r_2 and d_2 to r_2, are set with a 10Mbps capacity and a delay of 1ms. This is to make sure that these links would not have a significant impact on the simulation result. The bottleneck link has a delay of 50ms. The bandwidth of the link is varied in the experiments. The queue length of the link is set to allow 15Mbps of traffic, which is more than enough for the transmission of the video and the background traffic.

Ten common video sequences, *akiyo*, *coastguard*, *container*, *foreman*, *hall_monitor*, *mother_daughter*, *pamphlet*, *silent*, *stefan* and *tennis*, were used in the simulation. These videos were downloaded in y4m format from [68]. They were chosen to have different types of video content. Example frame content and their descriptions are shown in Figure 21.

Each of these videos has a length of 300 frames with a frame rate of 29.97 frames per second. The resolution of the videos is 352x288. Each video is concatenated to itself 10 times so that more statistically respectable results can be obtained. Therefore the duration of each complete video sequence becomes 100 seconds. After concatenation the videos are transcoded using H.264 and are transmitted as RTP streams using the VLC media player [69]. These RTP streams are converted to RTP traces using the RTP tools [70]. The videos are encoded using a constant bitrate of 300kbps.

The foreground has little movement and the background does not move at all.

a) akiyo

The foreground moves fairly quickly and the background does not move at all.

f) mother_daughter

The background is constantly moving while the foreground maintains a similar location.

b) coastguard

The foreground moves quickly and the background does not move at all.

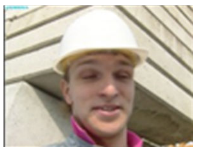
g) pamphlet

The background is constantly moving while the foreground maintains a similar location.

c) container

The foreground moves fairly quickly and the background does not move at all.

h) silent


The foreground and the background move drastically.

d) foreman

The foreground and the background move drastically.

i) stefan

The foreground has little movement and the background does not move at all.

e) hall_monitor

The foreground and the background move quickly and there is a change of scene.

j) tennis

Figure 21. Description of the video sequences used in the simulation.

The interfering traffic is a real network trace obtained from the website of the Computer for Interaction and Communications Research Group (COMICS) of the University of Naples Federico II [71]. A traffic trace of TCP port 80 going out from the university network to the rest of the Internet is selected for analysis. The interfering traffic is switched on at the start of the simulation and continues for the entire duration of the simulation. The average bitrate of the traffic is about 2.2Mbps, with a maximum bitrate of 4.1Mbps. The bitrate of the interfering traffic against time is shown in Figure 22.

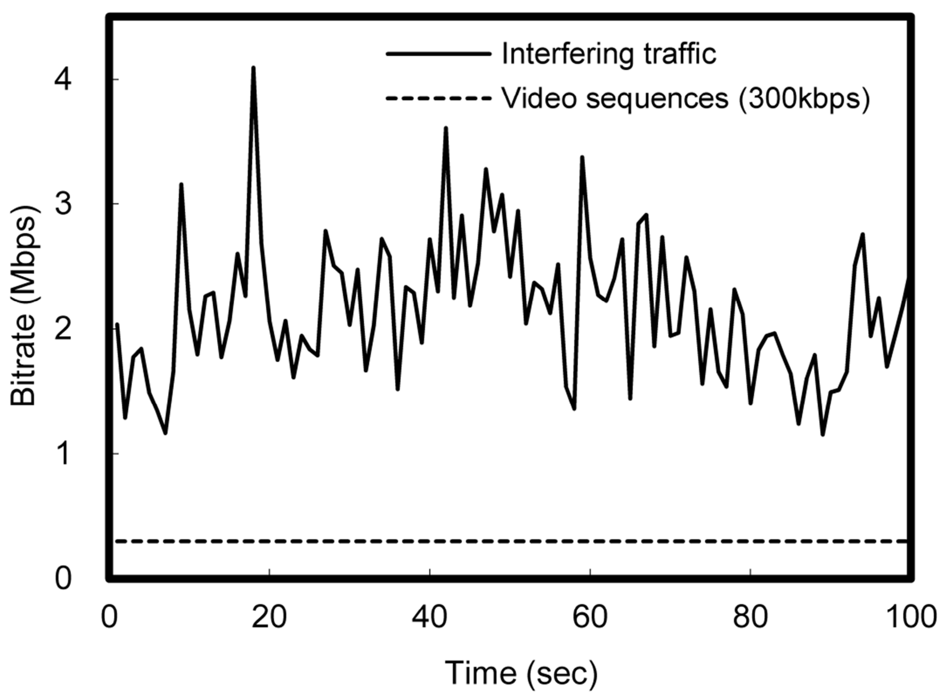


Figure 22. An illustration of the bitrate of the interfering traffic and the video sequences used in the simulation.

6.1.2 Transmission Methods

The video RTP trace is transmitted between s_1 and d_1 using a variety of protocols and methods. Table 3 lists the methods that have been modeled and compared in this study.

6.1.2.1 HTTP Streaming

HTTP streaming is the transmission of streaming data via a HTTP channel. With the correct configurations, putting a multimedia file on a Web server allows modern Web browsers to progressively download the multimedia content while playing. This is an easy way to achieve multimedia streaming because no additional software other than the Web server and the Web browser is required. In addition, in the real world, other protocols such as UDP and TCP using uncommon ports are typically not allowed to pass through Internet firewalls. This makes HTTP a much safer approach for streaming multimedia content.

Table 3. Transmission methods used in the simulations.

Transmission method	Packaging method for binary attachment / compression technique	Abbreviation used in results
HTTP streaming	- / -	HTTP
SOAP-oriented component-based framework	Base64 encoding / -	SCWS
The proposed multimedia streaming Web services architecture	MTOM / -	MSWS
	Binary XML / EXI	MSWS-EXI

In the experiment, the performance of multimedia streaming in a HTTP connection is analyzed. The RTP trace is transmitted using the HTTP chunked transfer encoding [72]. When a client makes a connection to the server, the server sends back the RTP packets separated using the HTTP chunked transfer encoding. Applying the chunked transfer encoding means the size of each packet is sent before the actual content of the packet so that RTP packets can be easily considered as separate units within the stream. Using this scheme the size overhead of the transmission is the size of the HTTP headers plus the chunk size transmitted before each multimedia packet.

6.1.2.2 SOAP-Oriented Component-based Framework

The SOAP-oriented component-based framework is proposed by Zhang et al, which has been discussed in section 3.2. To transmit streaming multimedia a metadata message is first sent to the receiver. The metadata message contains the number of components, RTP packets in this case, and their identifiers. Since there are a huge number of RTP packets in a multimedia stream the metadata is simplified so that only the number of packets and the starting sequence number are transmitted to the receiver. Based on this information the receiver then makes separate requests to the transmitter so that the RTP packets can be retrieved from the transmitter. These requests are sequentially sent to the transmitter. Moreover, each request is sent only after the previous response has been completely received. The RTP packets are transmitted in SOAP messages as base64 encoded entities.

6.1.2.3 Proposed Multimedia Streaming Web Services Architecture

Table 2 shows the transfer overhead of various packaging methods and compression techniques. To improve the performance of the proposed multimedia streaming Web services architecture one of the methods/techniques can be employed. It is shown in Table 2 that when binary XML schemes are used the transfer overhead is the smallest compared with other packaging and compression techniques.

In the tests in this study, first, the streaming content transmission in the proposed extension without any compression is evaluated as the base case. That means the streaming content is carried using the MTOM packaging method without any compression on the SOAP message. Then the performance of incorporating binary XML in the architecture is investigated. According to Table 2, the appropriate choice is EXI, which gives the smallest overhead size to packet size ratio. That means the packet data is put inside the encoded SOAP message in a binary form.

6.2 Results

In the experiments each video sequence was transmitted across the single bottleneck network using the multimedia streaming methods described in the previous section, using various bandwidth values for the bottleneck link. Each multimedia streaming method is tested against the same range of bandwidth values. Given that the average bitrate of the interfering traffic is 2.2Mbps and the video sequences have an average bitrate of 300kbps the assessment starts at a bandwidth value of 2.8Mbps. This is based on the observation of [65] that a TCP-based multimedia streaming method generally has an acceptable performance when the available throughput is twice the bitrate of the transmitted video. The maximum traffic is approximately 4.5Mbps in the entire duration of the transmission and therefore the upper limit for the bandwidth used during the assessment was 4.5Mbps.

The performance of the transmission methods are evaluated using two measurements. The first measurement is the RTP packet delay during the transmission. Packet delay affects the responsiveness of the real-time behaviour of the provided streaming Web service. It is also one of the major considerations in service selection and composition, especially when a streaming service is composed with another streaming service. In the experiments the RTP packet delay is defined as the difference between the time when the RTP packet is received at the video traffic receiver and the time when the RTP packet is sent from the video traffic sender. Although the processing time of the SOAP messages affects the overall processing time at both the client and server the SOAP processing time (in the scale of milliseconds [19]) is negligible compared to the packet delay (in the scale of seconds).

The second measurement compares the resulting video quality using the peak signal-to-noise ratio (PSNR) between the resulting video sequence and the source of the video sequence.

PSNR is one of the most widely used video quality evaluation metrics. It gives a good account of the video quality variation when the video content and encoding method are the same [73]. Given a compressed video frame C and its corresponding reference video frame R in the evaluated video sequence, the PSNR value of the compressed frame is expressed as:

$$PNSR_{frame} = 10 \cdot \log_{10}\left(\frac{{V_{max}}^2}{\frac{1}{mn}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}\left(Frame_C(i,j) - Frame_R(i,j)\right)^2}\right)$$

where m and n are the dimension of a video frame, $Frame(i, j)$ is the luminance of a pixel at the position (i, j) and V_{max} is the maximum possible luminance value.

PSNR values are expressed in dB, where higher PSNR values mean better video qualities. For a video sequence the mean and standard deviation of all PSNR values of the video frames represent the quality of the video. In the experimental results PSNR is used to compare the changes in video quality amongst the various transmission methods for each of the video sequences.

The playback buffer when evaluating the PSNR of each video sequence is the maximum RTP packet delay of the corresponding video transmission using the HTTP method. Any RTP packets arriving later than the size of the playback buffer are dropped. The HTTP method is the benchmark transmission method which is assumed to deliver a video sequence in perfect quality whereas other transmission methods will have varying levels of dropped RTP packets depending on the distribution of the RTP packet delay values.

6.2.1 RTP Packet Delay

In this section the performance of the four transmission methods are compared using the RTP packet delay. Figure 23 shows the cumulative distribution function (CDF) of the RTP packet delay values for each transmission method when the available bandwidth is 2.8Mbps. The results are generated using *akiyo*, one of the video sequences in the experiment.

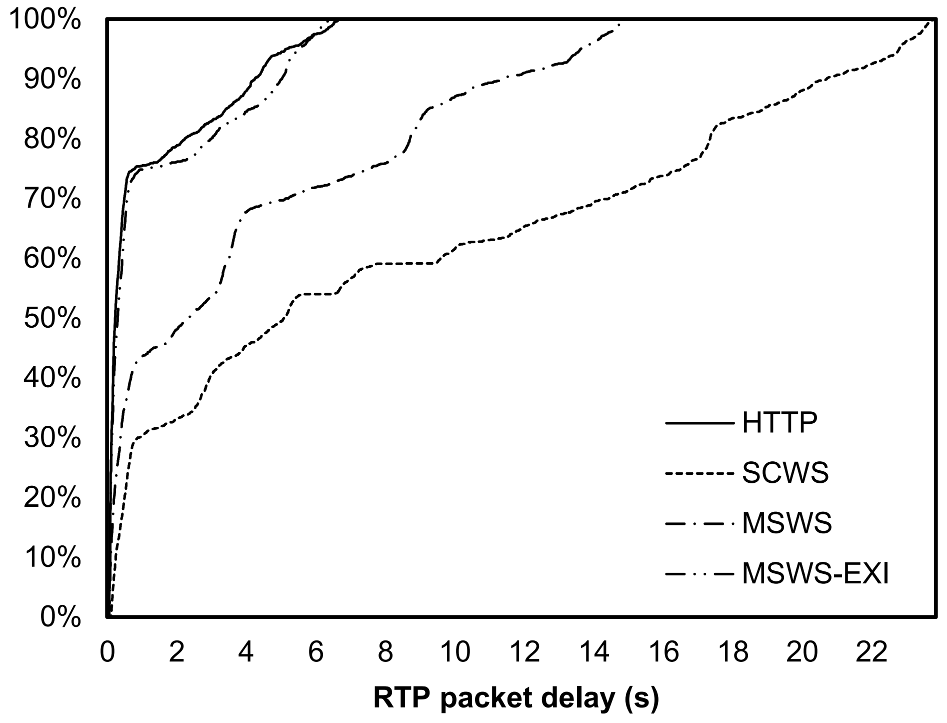


Figure 23. Cumulative distribution function of RTP packet delay of each transmission method for the video sequence akiyo using a 2.8Mbps bandwidth.

In Figure 23, at a bandwidth of 2.8Mbps, it can be seen that the RTP packet delay of SCWS is worst among the four methods. This is because the SCWS method has a relatively large overhead, compared to the other three methods. The major overhead of the SCWS method includes the Web service connection setup required for each packet and the simple but inefficient base64 encoding scheme of the binary data. The MSWS method, that is the proposed extension without using any compression scheme, has a much better performance than the SCWS method. By using the new MEPs and HTTP bindings, the streaming performance has improved over the SCWS method, which is mostly based on the current Web services standard. When the binary XML scheme is used in the proposed extension the performance is very close to that of the HTTP method, which is the best performer in general. In conclusion, in the situation where a bandwidth value of 2.8Mbps and the video sequence *akiyo* are used, Figure 23 shows that the proposed extension can efficiently transmit streaming multimedia data. The proposed extension does not have a significantly performance downgrade when comparing to the HTTP method which has the least transfer overhead.

To study the performance of the four transmission methods for all video sequences Figure 24 shows the CDF of the RTP packet delay for all other video sequences except *akiyo*. Since the shapes of the CDF are the major indicator of the performances of the transmission method the ranges of the x axes in Figure 24 are not the same for all graphs. In Figure 24 it can be seen that the performances of the transmission methods have a similar pattern exhibited by the result produced by using the video sequence *akiyo*. The SCWS method is consistently the worst among the four transmission methods for all video sequences. In the opposite end, the HTTP method is consistently the best performer. These are expected as the SCWS method and the HTTP method have the worst and least transfer overhead respectively. Without using compression the performance of the proposed extension varies across the video sequences but it is mostly closer to the SCWS method than the HTTP method. When binary XML scheme is used the performance of the proposed extension closely follows the one of the HTTP method. Figure 24 shows that while the proposed extension allows multimedia Web services to be implemented within the Web services architecture it can transmit streaming multimedia content from one end to another efficiently.

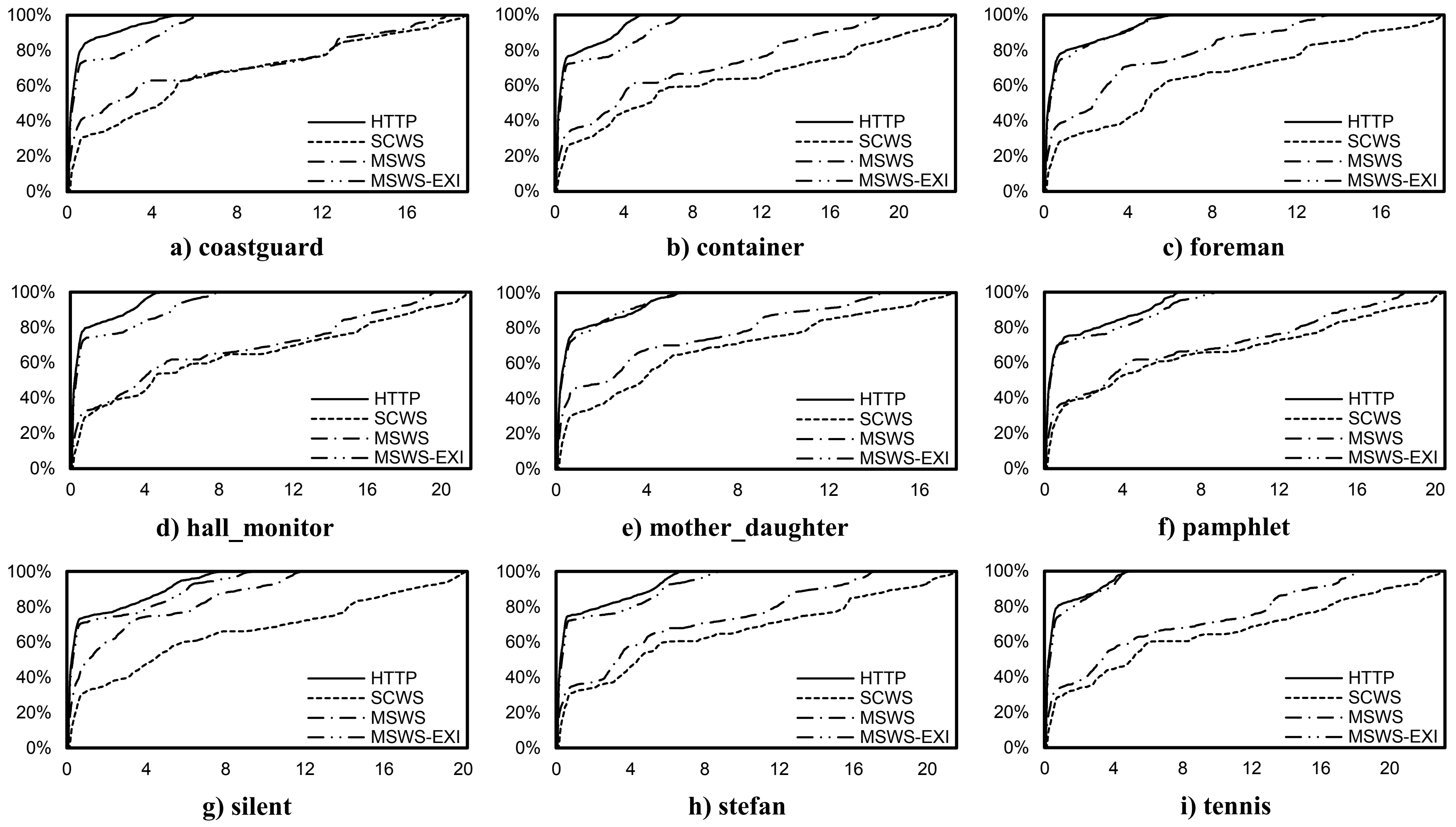


Figure 24. Cumulative distribution function of RTP packet delay of each transmission method for all video sequences except akiyo using a 2.8Mbps bandwidth. The axes are the same as those shown in Figure 23.

Table 4. Statistics for the RTP packet delay (in seconds) of each transmission method for the video sequence akiyo using a 2.8Mbps bandwidth.

Transmission method	Statistics of RTP packet delay			
	Average value (s)	Minimum value (s)	Maximum value (s)	Standard deviation (s)
HTTP	1.12	0.06	6.79	1.74
SCWS	8.36	0.12	23.85	8.04
MSWS	4.00	0.06	14.96	4.49
MSWS-EXI	1.28	0.06	6.44	1.90

To assist in examining the packet delay quantitatively Table 4 was created. It shows the statistics of RTP packet delay of each transmission method using the video sequence *akiyo* and a 2.8Mbps bandwidth. By looking at the average RTP packet delay values it is obvious that the HTTP method and MSWS-EXI method performs much better than their counterpart. The maximum RTP packet delay value of the SCWS method is 23.85 seconds. That means if the Web service has to display the received multimedia stream the playback buffer would have to be longer than 20 seconds long in order to show the resulting video in acceptable quality. Compared to the HTTP method or the MSWS-EXI method only a few seconds of playback buffer is required. When considering the average value and the standard deviation the MSWS-EXI method can perform close to the HTTP method. However, when compared to the SCWS method, which has an average value 8 times more than that of the HTTP method, the MSWS-EXI method drastically improves the efficiency of streaming multimedia within the Web services architecture.

Table 5. Average RTP packet delay (in seconds) and the standard deviations of the HTTP method and the MSWS-EXI method for all video sequences using a 2.8Mbps bandwidth.

Video sequence	HTTP		SCWS		MSWS		MSWS-EXI	
	Average value (s)	Standard deviation	Average value (s)	Standard deviation	Average value (s)	Standard deviation	Average value (s)	Standard deviation
akiyo	1.12	1.74	8.36	8.04	4.00	4.49	1.28	1.90
coastguard	0.65	1.07	5.94	5.82	5.15	5.89	1.21	1.78
container	0.93	1.37	8.26	7.83	5.90	6.08	1.50	2.19
foreman	0.92	1.46	6.32	5.85	3.55	3.92	1.00	1.47
hall_monitor	0.82	1.25	7.43	7.23	6.50	6.61	1.43	2.18
mother_daughter	0.89	1.38	5.34	5.32	3.79	4.39	0.89	1.31
pamphlet	1.32	2.00	6.63	6.84	5.76	6.13	1.63	2.45
silent	1.31	2.04	6.75	6.62	2.87	3.51	1.70	2.56
stefan	1.22	1.94	7.17	7.00	5.28	5.45	1.61	2.45
tennis	0.80	1.24	7.82	7.59	5.80	5.98	0.94	1.33

The average RTP packet delay values and standard deviations of all transmission methods for all video sequences using a 2.8Mbps available bandwidth are shown in Table 5. The average RTP delay values reflect the general performance of the transmission methods. The standard deviations indicate the stability of the transmission. These can be used to estimate the required size of the playback buffer. Given a packet loss rate, one of the major QoS parameters of streaming multimedia, the size of the playback buffer can be approximated by adding certain multiples of the standard deviation to the average packet delay value. For instance, if the acceptable packet loss rate is 2% the approximated playback buffer size will be the average packet delay value plus twice the standard deviation.

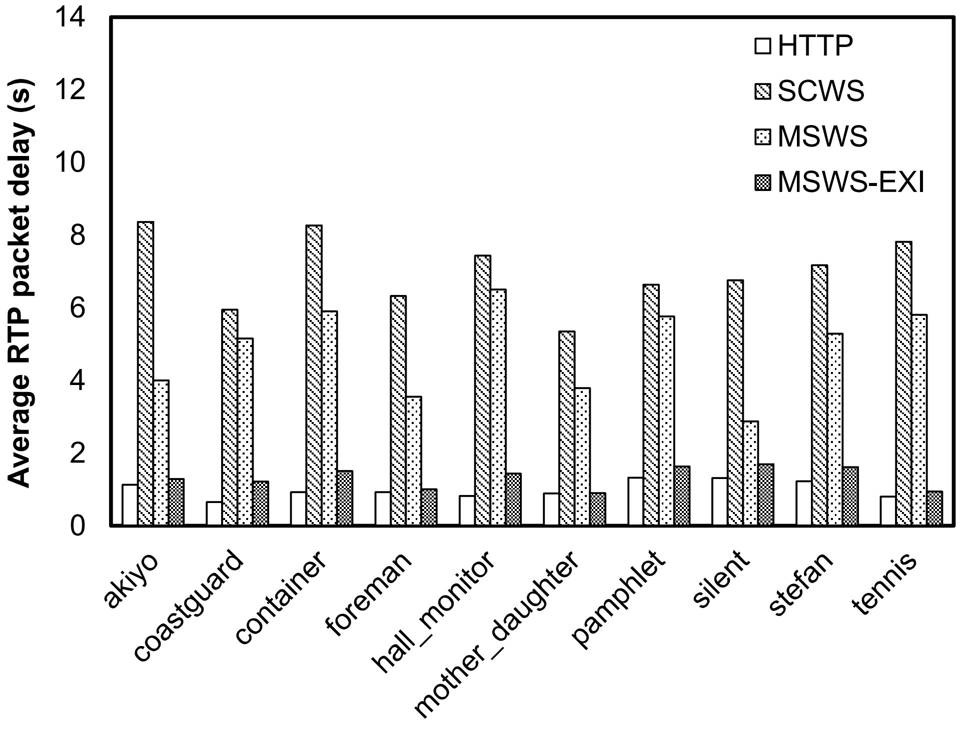


Figure 25. Average RTP packet delay of all transmission methods for all video sequences using a 2.8Mbps bandwidth.

The performances of all transmission methods for all video sequences, based on the average RTP delay values, are shown in Figure 25. It can be observed that the MSWS-EXI method consistently performs close to the HTTP method while greatly outperforms the performance of the SCWS method by multiple folds. In Table 5 the performance improvement of the MSWS-EXI method from the SCWS method is shown to range from approximately 300% (the video sequence *silent*) to approximately 735% (the video sequence *tennis*). The average improvement for all video sequences is approximately 450%. Even if compression is not applied to the proposed extension the MSWS method can still outperform the SCWS method across all video sequences.

If the acceptable packet loss rate is set at 2% the approximated playback buffer size required for each transmission method for all video sequences is shown in Figure 26 and Table 6.

Table 6. Approximate playback buffer size required by all transmission methods for all video sequences using a 2.8Mbps bandwidth.

Video sequence	Approximate playback buffer size (s)			
	HTTP	SCWS	MSWS	MSWS-EXI
akiyo	4.61	24.44	12.97	5.08
coastguard	2.79	17.58	16.93	4.77
container	3.67	23.92	18.06	5.87
foreman	3.85	18.01	11.39	3.93
hall_monitor	3.33	21.89	19.72	5.80
mother_daughter	3.66	15.98	12.57	3.52
pamphlet	5.33	20.30	18.02	6.52
silent	5.40	19.99	9.89	6.82
stefan	5.10	21.17	16.18	6.50
tennis	3.28	22.99	17.77	3.59

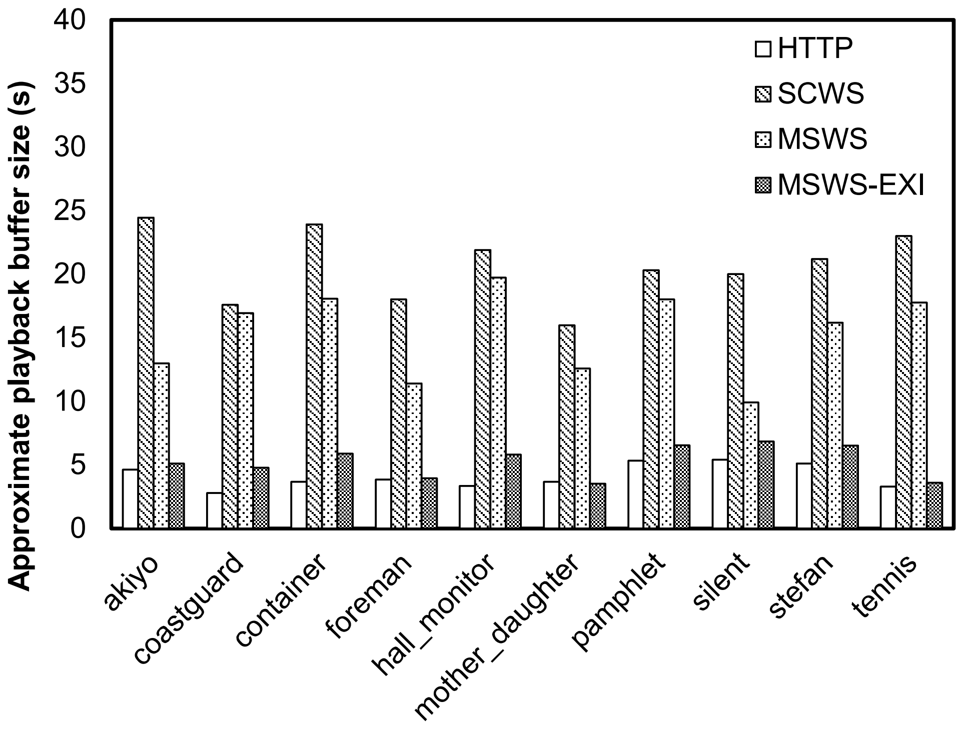


Figure 26. Approximate playback buffer size required by all transmission methods for all video sequences using a 2.8Mbps bandwidth.

As mentioned before, the playback buffer size is calculated by adding the average RTP packet delay value and twice the standard deviation. From Table 6 it can be seen that the SCWS method requires around 20 seconds of playback buffer. The MSWS method improves the playback buffer size by half in some cases. And it improves on average 75% of the playback buffer size of the SCWS method. However the performance of the MSWS method varies a lot compared to the SCWS method. Upon using compression in the MSWS-EXI the proposed extension further improves the playback buffer size to an average of 25% of the playback buffer size of the SCWS method. In addition, the performance of the MSWS-EXI method is more consistent than the MSWS method. These results are based on a 2.8Mbps bandwidth, which is the lowest value simulated in the experiments. In Table 7 and Figure 27 the performances of the transmission methods using different available bandwidth for the video sequence *akiyo* are shown.

Table 7. The average RTP packet delay for each transmission method against different available bandwidth for the video sequence akiyo.

Transmission method	Bandwidth (Mbps)																	
	2.8	2.9	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	4.0	4.1	4.2	4.3	4.4	4.5
HTTP	1.12	0.88	0.56	0.38	0.36	0.25	0.23	0.20	0.18	0.15	0.13	0.12	0.12	0.12	0.11	0.11	0.11	0.11
SCWS	8.36	5.04	2.06	1.15	0.91	0.71	0.39	0.40	0.29	0.27	0.24	0.22	0.21	0.20	0.18	0.19	0.18	0.18
MSWS	4.00	2.76	1.36	1.03	0.63	0.51	0.60	0.26	0.22	0.20	0.19	0.15	0.14	0.11	0.11	0.10	0.10	0.10
MSWS-EXI	1.28	0.96	0.66	0.46	0.32	0.31	0.25	0.22	0.19	0.15	0.14	0.12	0.12	0.11	0.11	0.11	0.10	0.10

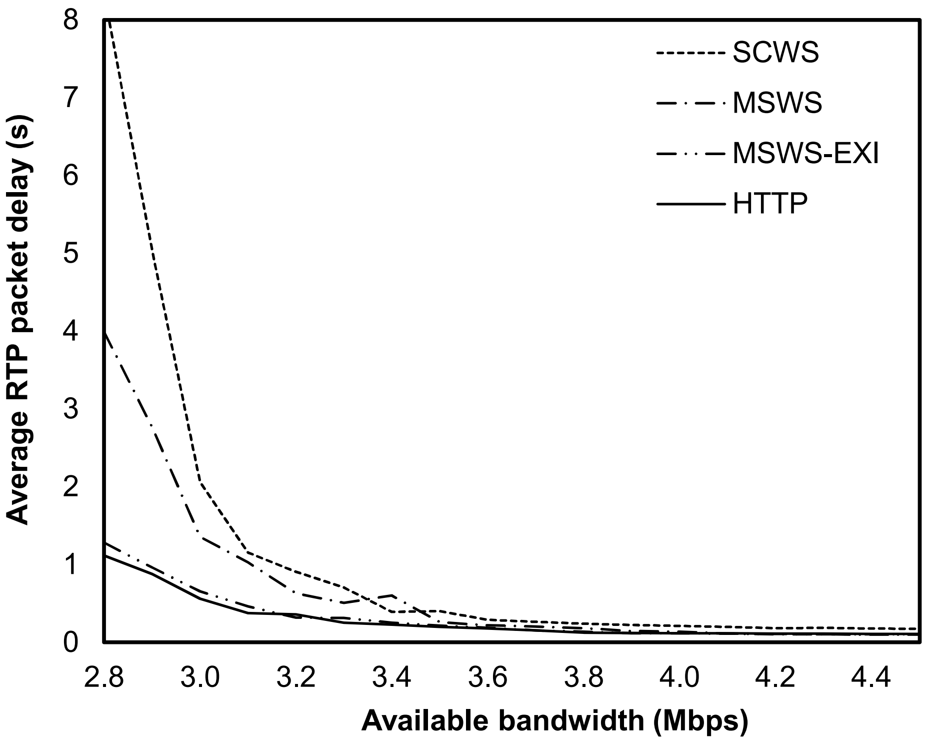
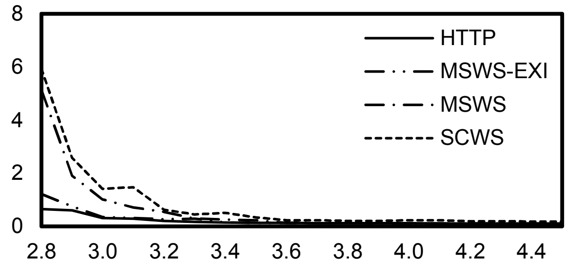
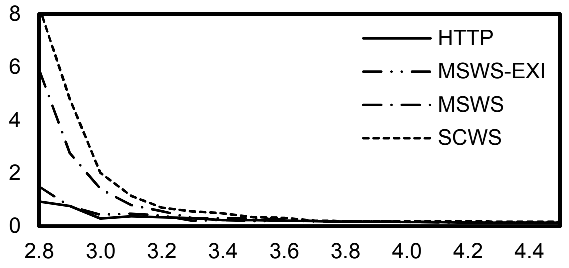


Figure 27. The average RTP packet delay for each transmission method against different available bandwidth for the video sequence akiyo.

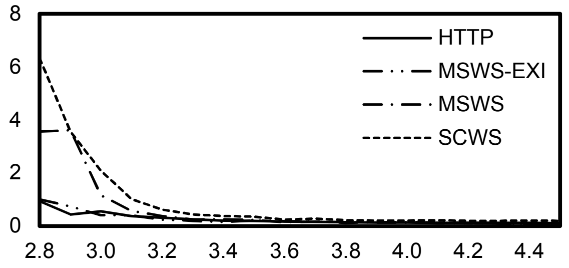
When the video sequence *akiyo* is transmitted at a bandwidth of 2.8Mbps using the transmission methods, the difference between the performance of the SCWS method and the MSWS-EXI method is fairly large. When the available bandwidth increases their performances get closer together. This is understandable as there are less network congestions when the bandwidth increases. However, from Table 7 it can be seen that the average RTP packet delay of the SCWS method can only get to as close as approximately 180% of the other methods. This is because only the SCWS method has the overhead of setting up the Web service connections. Compared to this the MSWS method and the MSWS-EXI method do not need a separate Web service connection for each packet. Therefore, the MSWS method has a performance close to the HTTP method starting from a bandwidth of 4.0Mbps whereas the MSWS-EXI method performs even better by having a similar performance to the HTTP method starting from a bandwidth of 3.4Mbps.



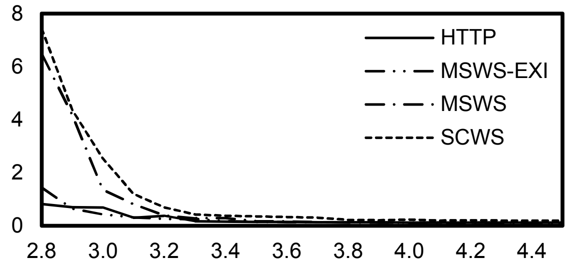
a) coastguard



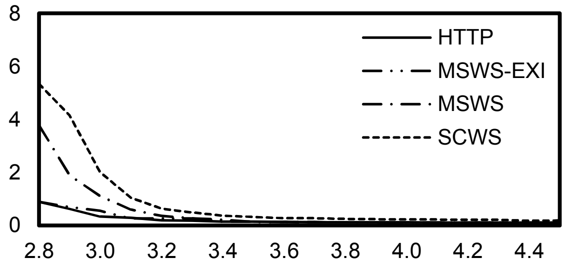
b) container



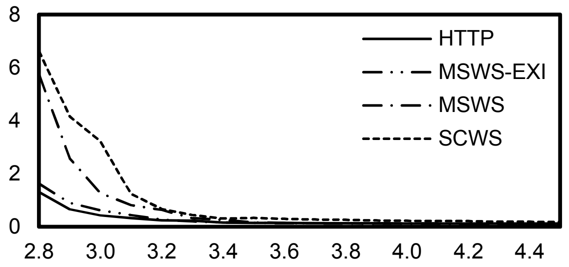
c) foreman



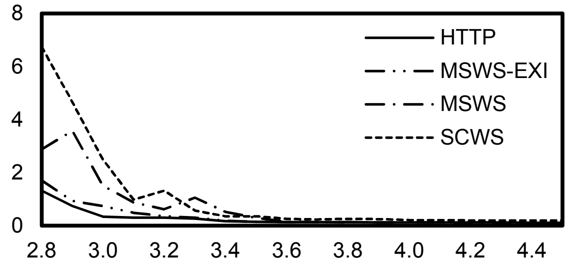
d) hall_monitor



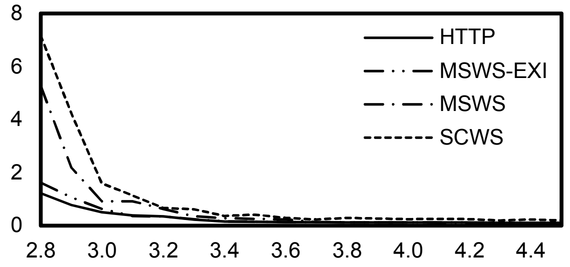
e) mother_daughter



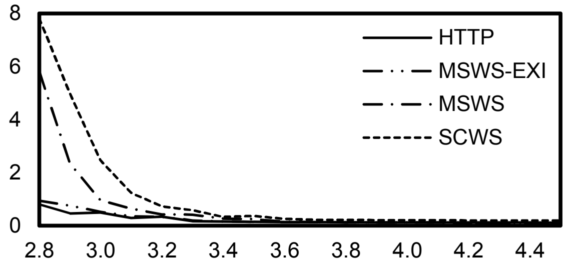
f) pamphlet



g) silent



h) stefan



i) tennis

Figure 28. Average RTP packet delay of each transmission method against different bandwidth values for all video sequences except akiyo. The axes are the same as those shown in Figure 27.

Figure 28 shows the average RTP packet delay of each transmission method against the increasing available bandwidth of each video sequence. As can be seen from the figure, the performances of the transmission methods are similar to that of the video sequence *akiyo*. Considering only the performances of the MSWS-EXI method and SCWS method, Table 8 shows the ratio between the RTP delay values of the MSWS-EXI method and the SCWS method. From the table the average ratio steadily increases from 19% at a bandwidth of 2.8Mbps to 57% at a bandwidth of 3.8Mbps. At a higher available bandwidth the average ratio increases very slowly. This is because without network congestion the connection setup has become the dominant overhead for the SCWS method.

In summary, based on the RTP packet delay results, the proposed multimedia streaming Web services architecture using binary XML is shown to hugely improve the streaming performance of the method based on the current Web services standards. Moreover, the performance of the proposed extension is very close to that of the HTTP transmission method. In the next section the focus is on the performance of the systems with respect to the produced video quality.

Table 8. The ratio between the RTP delay values of the MSWS-EXI method and the SCWS method for each video sequence, expressed as a percentage.

Video sequence	Bandwidth (Mbps)																	
	2.8	2.9	3.0	3.1	3.2	3.3	3.4	3.5	3.6	3.7	3.8	3.9	4.0	4.1	4.2	4.3	4.4	4.5
akiyo	15%	19%	32%	40%	35%	45%	64%	54%	65%	57%	58%	53%	55%	56%	59%	56%	59%	58%
coastguard	20%	29%	24%	21%	42%	64%	29%	40%	56%	53%	57%	57%	48%	48%	56%	56%	56%	56%
container	18%	16%	21%	41%	56%	35%	47%	56%	65%	88%	87%	85%	83%	82%	78%	76%	75%	75%
foreman	16%	21%	20%	39%	38%	44%	43%	58%	73%	59%	69%	65%	64%	53%	58%	58%	54%	57%
hall_monitor	19%	15%	16%	26%	37%	60%	45%	41%	41%	42%	56%	58%	53%	58%	55%	58%	58%	58%
mother_daughter	17%	17%	27%	25%	41%	54%	38%	43%	45%	43%	45%	46%	47%	47%	47%	47%	56%	51%
pamphlet	25%	22%	19%	36%	41%	47%	54%	45%	45%	46%	46%	51%	52%	52%	53%	56%	57%	59%
silent	25%	20%	29%	50%	26%	52%	52%	41%	54%	57%	49%	50%	57%	58%	58%	57%	57%	57%
stefan	22%	25%	39%	32%	50%	38%	44%	34%	47%	55%	43%	44%	49%	44%	47%	56%	47%	56%
tennis	12%	15%	21%	28%	44%	34%	47%	38%	52%	57%	57%	57%	57%	57%	57%	57%	56%	56%
Average:	**19%**	**20%**	**25%**	**34%**	**41%**	**47%**	**46%**	**45%**	**54%**	**56%**	**57%**	**57%**	**56%**	**55%**	**57%**	**58%**	**57%**	**58%**

6.2.2 PSNR of Video Sequences

In this section the PSNR values of each video sequence are compared for each of the investigated transmission methods.

Table 9. Frame loss percentage of all transmission methods for all video sequences using a 2.8Mbps bandwidth.

Video sequence	Frame loss percentage			
	HTTP	SCWS	MSWS	MSWS-EXI
akiyo	0.00%	44.47%	26.83%	0.00%
coastguard	0.00%	39.40%	37.00%	5.47%
container	0.00%	51.30%	38.70%	12.47%
foreman	0.00%	36.73%	24.57%	0.00%
hall_monitor	0.00%	46.07%	39.83%	12.13%
mother_daughter	0.00%	35.27%	29.87%	0.13%
pamphlet	0.00%	36.33%	33.97%	4.27%
silent	0.00%	34.20%	12.53%	4.47%
stefan	0.00%	39.97%	31.97%	6.40%
tennis	0.00%	49.67%	40.20%	0.00%

As explained at the start of section 6.2 the PSNR values are produced using the maximum delay values of the corresponding HTTP method of each video sequence as the size of the playback buffer. Therefore the percentage of dropped packets of the HTTP method of each video sequence is always 0% because all RTP packets arrive within the size of the playback buffer. The percentage of frame loss, because of dropped packets, is shown in Table 9. The frame loss percentage for the SCWS method and the MSWS method is enormous because their transfer overhead sizes are bigger than the HTTP method. Thus their packet delay values are also much bigger than the HTTP method. Since there is no dropped packet for the HTTP method the PSNR values reflect the quality of the video after compression. The purpose of the PSNR values of the HTTP method is then to serve as a reference of the best possible quality of the received video in the simulation. The PSNR values are not correlated among different video sequences. Therefore the comparison of PSNR is focused on a particular video sequence when different transmission methods are used. Apart from the average PSNR value the standard deviation of the PSNR shows the fluctuation of the video quality across the duration of the video sequence.

Table 10 shows the average PSNR values and standard deviations for each video sequence and their transmission method when the available bandwidth is 2.8Mbps. It can be seen that the PSNR values and standard deviation of the MSWS-EXI method are very close to or the same as that of the HTTP method in all cases. That means the video quality produced by the MSWS-EXI can closely match the reference quality. This is because the frame loss percentage for the MSWS-EXI method is very close to the one for the HTTP method, as shown in Table 9. As a result, only minor distortion in the frame content occurs in the received video sequences.

It can be seen in Table 9 that the frame loss percentage for the MSWS method and SCWS method is much higher than the MSWS-EXI method. It is difficult to reconstruct a video sequence when the frame loss percentage is high. Therefore, as can be seen from Table 10, the MSWS method and SCWS method generally give lower PSNR values than the MSWS-EXI method. And because of a lower average PSNR value the MSWS method and the SCWS method have a higher standard deviation, which means fluctuation of quality across frames in the received video sequence.

Table 10. Average PSNR (in dB) and standard deviation of PSNR (in dB) of the video sequences and their transmission methods using a 2.8Mbps available bandwidth.

Video sequence	Transmission method	Average PSNR	Std. dev. of PSNR	Transmission method	Average PSNR	Std. dev. of PSNR
akiyo	HTTP	44.99	1.16	MSWS	37.48	12.45
	SCWS	32.00	14.51	MSWS-EXI	44.99	1.16
coastguard	HTTP	30.66	0.78	MSWS	24.73	7.75
	SCWS	24.43	7.52	MSWS-EXI	29.82	3.34
container	HTTP	37.78	0.98	MSWS	30.68	9.05
	SCWS	28.29	9.30	MSWS-EXI	35.46	5.91
foreman	HTTP	35.16	1.30	MSWS	30.01	9.12
	SCWS	26.95	10.97	MSWS-EXI	35.16	1.30
hall_monitor	HTTP	38.15	0.73	MSWS	31.61	8.02
	SCWS	31.67	7.19	MSWS-EXI	36.31	4.85
mother_daughter	HTTP	41.80	0.88	MSWS	35.65	9.55
	SCWS	34.46	9.93	MSWS-EXI	41.73	1.13
pamphlet	HTTP	41.86	1.08	MSWS	33.58	11.70
	SCWS	32.56	12.19	MSWS-EXI	40.71	5.23
silent	HTTP	37.12	0.75	MSWS	34.67	6.41
	SCWS	30.24	9.39	MSWS-EXI	36.32	3.65
stefan	HTTP	28.57	1.11	MSWS	24.17	6.58
	SCWS	23.08	6.83	MSWS-EXI	27.63	3.73
tennis	HTTP	32.22	1.26	MSWS	24.83	8.89
	SCWS	23.14	9.03	MSWS-EXI	32.22	1.26

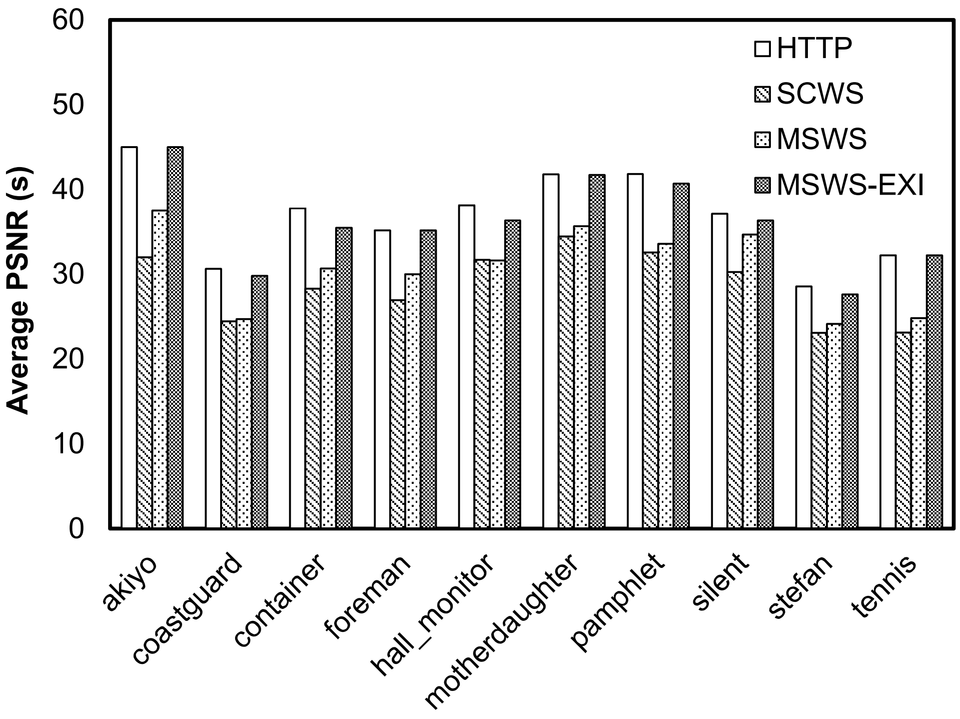


Figure 29. The average PSNR of each video sequence using different transmission methods with a 2.8Mbps available bandwidth.

The PSNR values are also shown in Figure 29. It can be seen that the SCWS method has lower average PSNR values than its counterparts for all video sequences. However, the differences in PSNR are not relatively the same across different video sequences. This can be explained by the calculation of the PSNR. When there is a missing frame in the transmitted video the missing frame is typically replaced by the previous frame. In the case of a video sequence without much movement, such as the video sequence *hall_monitor*, some of these missing frames can still achieve a very high PSNR value because they are essentially the same as the previous frame in the video. In essence, different video content can affect the calculated PSNR values. Nevertheless as shown in Figure 29 the MSWS-EXI method can produce a video with a better PSNR value than the SCWS method in most of the cases.

In addition to the comparison of average PSNR values a mean opinion score (MOS) can be calculated using a mapping between the PSNR values and a 5-point scale [74]. MOS reflects the scale of subjective opinion test of user perceived audio/video quality [75]. Although the experiments carried out in this thesis do not have subjective ratings based on user opinions the mapping from PSNR to MOS gives an approximation of the user perceived quality of the transmitted video as shown in Table 11.

Table 11. The perceived quality rating of the Mean Opinion Score (MOS) and the mapping between PSNR (in dB) and MOS.

MOS	Perceived Quality	PSNR
5	Excellent	> 37
4	Good	31 – 37
3	Fair	25 – 31
2	Poor	20 – 25
1	Bad	< 20

The mapping between PSNR and MOS is applied in a frame-by-frame basis. That means, first, a MOS is calculated for each frame of a video sequence. Then the overall MOS value of the video sequence is calculated by averaging the MOS values of all its video frames. Figure 30 shows the MOS values of all video sequences for all transmission methods in a 2.8Mbps available bandwidth.

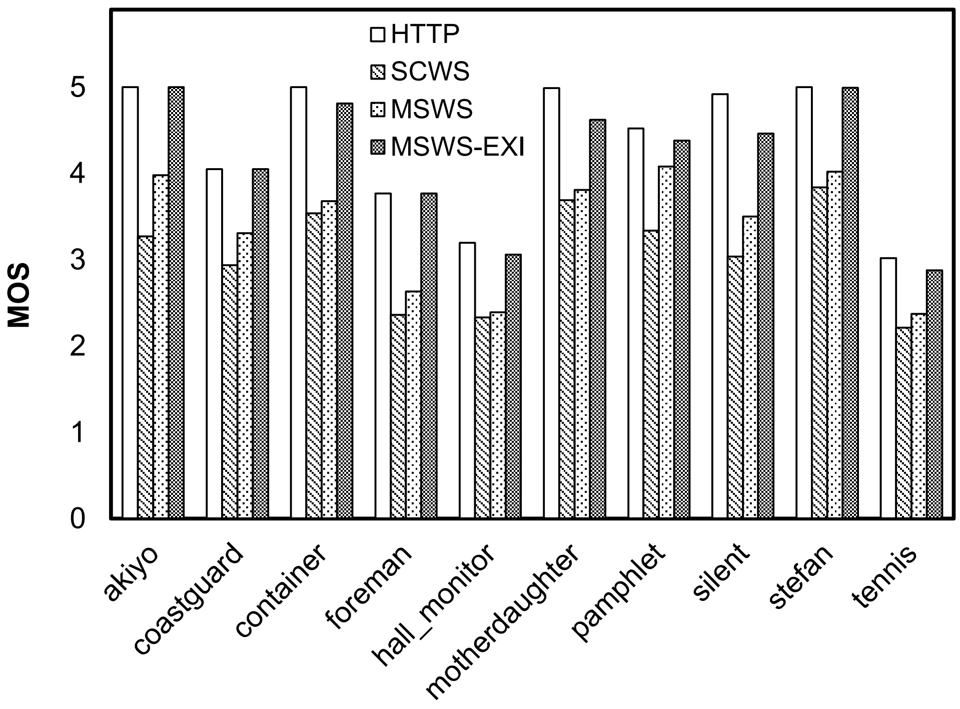


Figure 30. The MOS of each video sequence using different transmission methods with a 2.8Mbps available bandwidth.

As can be seen from Figure 30 both the HTTP method and the MSWS-EXI method have an excellent quality for most of the video sequences. Similar to the PSNR results the SCWS method consistently has the lowest MOS values. This is due to the fact that some of the video frames in the SCWS method cannot be reconstructed and are therefore replaced by a previous frame. Because of this the overall quality of the resultant video sequence is low. Looking across the video sequences the MSWS-EXI method typically has one level higher of MOS, for example, from fair to good or from good to excellent, than the SCWS method. In terms of user experience this is an excellent improvement using the proposed extension in the Web services architecture.

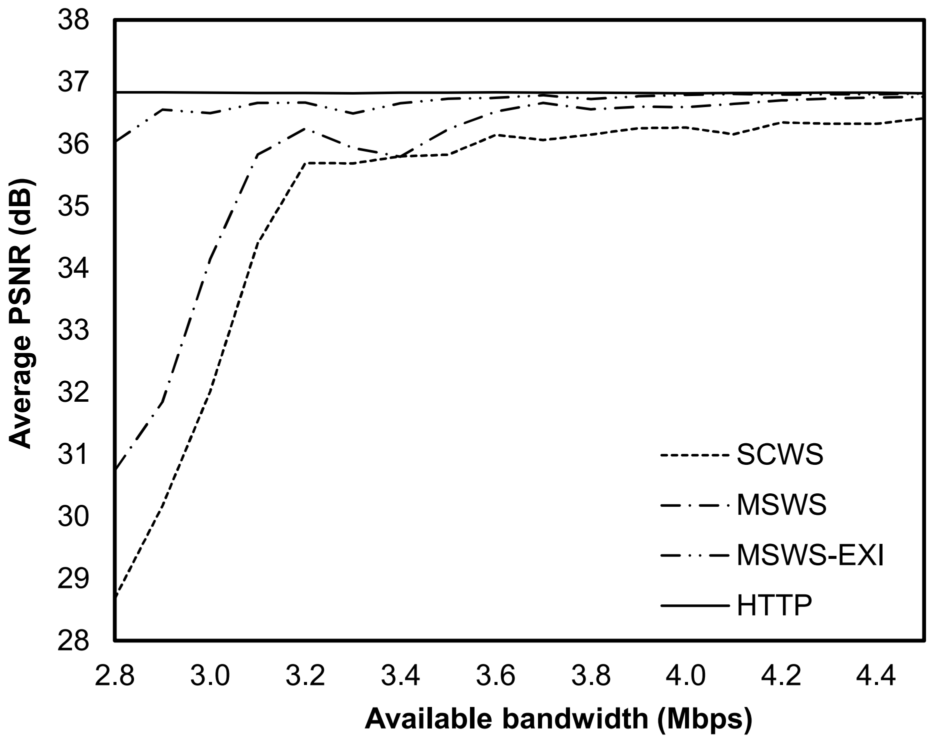


Figure 31. The average PSNR of all video sequences for each transmission method against different levels of available bandwidth.

For relatively higher values of available bandwidth the average PSNR was predicted to increase for all transmission methods, except the HTTP method. The HTTP method has the same PSNR value across different bandwidth values because, in the experiment, the playback buffer is sized according to its maximum RTP packet delay. For the rest of the transmission methods the average PSNR increases as the number of dropped packets decreases, which was the observed behaviour. This is shown in Figure 31 that the average PSNR values for all video sequences when different transmission methods are used against different available bandwidth. It can be observed that the PSNR of the HTTP method is constant at around 37dB. The performance of the MSWS-EXI method fluctuates just under 37dB. This is expected as the RTP packet delay is not significantly different to the HTTP method and the MSWS-EXI method, as can be seen from Table 7. The SCWS method and the MSWS method both improve drastically when the available bandwidth increases. Although all methods converge towards 37dB at the maximum available bandwidth the PSNR of the MSWS-EXI method is still much better than the SCWS method.

CHAPTER 7

CONCLUSION

This thesis has presented a novel and efficient extension to the Web services architecture for multimedia Web services. First a mechanism for publishing and discovering multimedia Web services is described. A metadata query service is created to help facilitate the description of the underlying multimedia content for a particular Web service. The transmission of multimedia streams is the result of two new streaming data MEPs and the implementations of the MEPs using HTTP SOAP bindings. The HTTP SOAP bindings use the MIME Multipart/Related structure to output the multimedia packets. Each packet is then transmitted as a package of a SOAP message and its binary data. The new MEPs and SOAP bindings incur additional transfer overhead in the transmission of multimedia streaming data compared to a simple HTTP transmission method. To reduce the effect of this overhead, the application of various SOAP compression techniques for the packaging of SOAP messages are investigated extensively. By comparing the performance of the compression techniques, using one of the binary XML schemes is the most efficient among all compression techniques. Experimental results of the proposed system show that the Web service transmission using binary XML schemes has a performance comparable to an HTTP multimedia streaming transmission scheme.

7.1 Future Work

7.1.1 Multimedia Web Service Composition

Using the proposed multimedia Web service framework large scale multimedia applications can be developed by the composition of a set of distributed multimedia Web services. The decision of the service selection and composition can be based on a set of QoS constraints such as minimizing the delay in the service path. Qian et al. [77] used delay and reliability as the measures for the optimization of the service path in the service composition graph. However, it did not consider a minimum bandwidth requirement, which is typically required by real-time multimedia applications. Approaches in network path selection algorithms such as [76] and [78] have considered a combination of delay and bandwidth constraints. If similar approaches can be adopted in service composition, multimedia applications with minimum bandwidth requirement can be developed under the multimedia Web service framework.

7.1.2 QoS Aware Multimedia Web Services

At the moment the multimedia data is transmitted using the original quality of the data. The multimedia Web service would be more efficient if the quality of the transmission is adaptable and can be controlled by some QoS parameters. For example, a SOAP request can be bundled with the required QoS parameter in the form of Composite Capability/Preference Profiles (CC/PP) [56]. Given the description in the CC/PP, a multimedia stream in an appropriate quality would be returned by the QoS-aware multimedia Web service. It would be ideal if multimedia quality can be adjusted at the start and on the fly.

BIBLIOGRAPHY AND REFERENCES

[1] T. Erl, “Service-Oriented Architecture: Concepts, Technology & Design,” Prentice Hall PTR, 2005

[2] K. Nahrstedt and W.-T. Balke, “A taxonomy for multimedia service composition,” in *Proc. the 12th ACM Int. Conf. Multimedia (MULTIMEDIA '04)*, pp. 88–95, 2004.

[3] "YouTube APIs and Tools," Google, http://code.google.com/apis/youtube/, 2011.

[4] YouTube, http://www.youtube.com

[5] J. R. Smith and P. Schirling, “Metadata standards roundup,” *Multimedia, IEEE*, vol.13, no.2, pp. 84–88, April-June 2006.

[6] B.S. Manjunath, P. Salembier, and T. Sikora, Eds., *Introduction to MPEG-7: Multimedia Content Description Interface*, Chichester, West Sussex, U.K.: Wiley, 2002.

[7] J. Wilkinson and B. Devlin, “The material exchange format (mxf) and its application,” *SMPTE journal*, vol. 111, no. 9, pp. 378–384, 2002.

[8] “Dublin Core Metadata Element Set, Version 1.1.,” Dublin Core Metadata Initiative, http://dublincore.org/documents/dces, 2010.

[9] T. Yu and K.J. Lin, “QCWS: an implementation of QoS-capable multimedia Web services,” *Multimedia Tools and Applications*, vol. 30, no. 2, pp. 165–187, Aug. 2006.

[10] J. Zhang, L. Zhang, F. Quek and J. Chung, “A service-oriented multimedia componentization model,” *International Journal of Web Services Research*, vol. 2, no. 1, pp. 54-76, Jan-Mar 2005.

[11] J. Zhang and J. Chung, “An open framework supporting multimedia web services,” *Multimedia Tools and Applications*, vol. 30, no. 2, Aug 2006, pp. 149-164.

[12] S. Decneut, F. Hendrickx, S. V. Assche and L. Nachtergaele, “Targeting heterogeneous multimedia environments with Web Services,” in *Proc. IEEE Int. Conf. Web Services 2004*, pp. 682-689.

[13] I. Brunkhorst, S. Tonnies and W.T. Balke, “Multimedia content provisioning using service oriented architectures,” in *Proc. IEEE Int. Conf. Web Services 2008*, pp. 262–269.

[14] S. Tonnies, B. Kohncke, P. Hennig and W.T. Balke, “A service oriented architecture for personalized rich media delivery,” in *Proc. IEEE Int. Conf. Services Computing 2009*, pp. 340–347.

[15] M. Ruth, F. Lin and S. Tu, “Adapting single-request/multiple-response messaging to Web services,” in *Proc. 29th Int. Conf. Computer Software and Applications 2005*, pp. 287–292.

[16] “OASIS Web Services Notification (WSN) TC,” OASIS, http://www.oasis-open.org/committees/wsn/

[17] D. Davis et al., “Web Services Eventing (WS-Eventing),” W3C, http://www.w3.org/TR/ws-eventing/, 2011.

[18] K. Chiu, M. Govindaraju, and R. Bramley, “Investigating the limits of SOAP performance for scientific computing,” *Proc. IEEE Int. Sym. High Performance Distributed Computing 2002*, pp. 246–254, Jul 2002.

[19] C. Kohlhoff and R. Steele, “Evaluating SOAP for high performance business applications: real-time trading systems,” *Proc. WWW’03*, 2003.

[20] S. Heinzl et al., “A scalable service-oriented architecture for multimedia analysis, synthesis and consumption,” *International Journal of Web and Grid Services*, vol. 5, no. 3, pp. 219–260, Sep 2009.

[21] S. Heinzl et al., “Flex-SwA: flexible exchange of binary data based on SOAP messages with attachments,” *Int. Conf. Web Services 2006*, pp. 3–10, Sep 2006.

[22] S. Oh, H. Bulut, A. Uyar, W. Wu and G. Fox, “Optimized communication using the SOAP infoset for mobile multimedia collaboration applications,” *Proc. Int. Sym. Collaborative Technologies and Systems 2005*, pp. 32–39, May 2005.

[23] International Telecommunication Union, “X.892: Information technology – Generic applications of ASN.1: Fast Web Services,” ITU, http://www.itu.int/rec/T-REC-X.892/, 2005.

[24] O. Dubuisson, “ASN.1 - Communication between heterogeneous systems,” Morgan Kaufmann Editor, Oct 2000.

[25] D. Booth et al., “Web Services Architecture,” W3C, http://www.w3.org/TR/ws-arch/, 2004

[26] C. M. MacKenzie et al., “OASIS Reference Model for Service Oriented Architecture V 1.0,” OASIS, http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, 2006

[27] N. Mitra and Y. Lafon, “SOAP Version 1.2 Part 0: Primer (second edition),” W3C, http://www.w3.org/TR/soap12-part0/, 2007

[28] J. Klensin, “Simple Mail Transfer Protocol,” IETF, http://www.ietf.org/rfc/rfc2821.txt, 2001

[29] R. Fielding et al., “Hypertext Transfer Protocol -- HTTP/1.1,” IETF, http://www.ietf.org/rfc/rfc2616.txt, 1999

[30] M. Gudgin et al., “SOAP version 1.2 part 1: Messaging framework (second edition),” W3C, http://www.w3.org/TR/soap12-part1/, 2007.

[31] M. Gudgin et al., “SOAP version 1.2 part 2: Adjuncts (second edition),” W3C, http://www.w3.org/TR/soap12-part2/, 2007.

[32] P. V. Biron and A. Malhotra, “XML Schema Part 2: Datatypes (second edition),” W3C, http://www.w3.org/TR/xmlschema-2/, 2004

[33] J. J. Barton et al., “SOAP Messages with Attachments,” W3C, http://www.w3.org/TR/SOAP-attachments/, 2000

[34] J. Cowan and R. Tobin, “XML Information Set (second edition),” W3C, http://www.w3.org/TR/xml-infoset/, 2004

[35] E. Levinson, “The MIME Multipart/Related Content-type,” IETF, http://www.ietf.org/rfc/rfc2387.txt, 1998

[36] H. F. Nielsen, H. Sanders, R. Butek and S. Nash, “Direct Internet Message Encapsulation (DIME),” IETF, http://xml.coverpages.org/draft-nielsen-dime-01.txt, 2002

[37] M. Gudgin et al., “SOAP Message Transmission Optimization Mechanism,” W3C, http://www.w3.org/TR/soap12-mtom/, 2005

[38] M. Gudgin et al., “XML-binary Optimized Packaging,” W3C, http://www.w3.org/TR/xop10/, 2005

[39] R. Chinnici et al., “Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language,” W3C, http://www.w3.org/TR/wsdl20/, 2007

[40] L. Clement et al.,“UDDI Version 3.0.2,” OASIS, http://uddi.org/pubs/uddi_v3.htm, 2004

[41] H. S. Thompson, “XML Schema Part 1: Structures (second edition),” W3C, http://www.w3.org/TR/xmlschema-1/, 2004

[42] M. Gudgin, M. Hadley and T. Rogers, “Web Services Addressing 1.0 – Core,” W3C, http://www.w3.org/TR/ws-addr-core/, 2006.

[43] J. Postel, “User Datagram Protocol,” IETF, http://www.ietf.org/rfc/rfc0768.txt, 1980

[44] J. Postel, “Internet Protocol,” IETF, http://www.ietf.org/rfc/rfc0791.txt, 1981

[45] J. Postel, “Transmission Control Protocol,” IETF, http://www.ietf.org/rfc/rfc0793.txt, 1981

[46] H. Schulzrinne et al., “RTP: A Transport Protocol for Real-Time Applications,” IETF, http://www.ietf.org/rfc/rfc3550.txt, 2003

[47] H. Schulzrinne and S. Casner, “RTP Profile for Audio and Video Conferences with Minimal Control,” IETF, http://www.ietf.org/rfc/rfc3551.txt, 2003

[48] J. Cheney, “Compressing XML with multiplexed hierarchical PPM models,” in *Proc. Conf. Data Compression 2001*, pp. 163–172.

[49] H. Liefke and D. Suciu, “XMill: an efficient compressor for XML data,” *SIGMOD Rec.*, vol. 29, no. 2, pp. 153–164, Jun. 2000.

[50] S. Sakr, “XML compression techniques: A survey and comparison,” *Computer and System Science*, vol. 75, no. 5, pp. 303–322, Aug. 2009.

[51] C. Werner, C. Buschmann and S. Fischer, “Compressing SOAP messages by using differential encoding,” in *Proc. IEEE Int. Conf. Web Services 2004*, pp. 540–547.

[52] M.C. Rosu, “A-SOAP: Adaptive SOAP message processing and compression,” in *Proc. IEEE Int. Conf. Web Services 2007*, pp. 200–207.

[53] International Telecommunication Union, “X.891: Information technology – Generic applications of ASN.1: Fast infoset,” ITU, http://www.itu.int/rec/T-REC-X.891/, 2005.

[54] J. Schneider and T. Kamiya, “Efficient XML interchange (EXI) format 1.0,” W3C, http://www.w3.org/TR/exi/, 2011.

[55] M. Brambilla et al., “Managing asynchronous Web services interactions,” in *Proc. IEEE Int. Conf. Web Services 2004*, pp. 80–87.

[56] G. Klyne et al., “Composite Capability/Preference Profiles (CC/PP): Structure and Vocabularies 1.0,” W3C, http://www.w3.org/TR/CCPP-struct-vocab/, 2007.

[57] J. H. Gailey, “Sending Files, Attachments, and SOAP Messages Via Direct Internet Message Encapsulation,” *MSDN Magazine*, Microsoft Inc., Dec 2002.

[58] A. Vetro, “MPEG-21 digital item adaptation: enabling universal multimedia access,” *Multimedia*, *IEEE* , vol.11, no.1, pp. 84–87, Jan-Mar 2004.

[59] H. Schulzrinne, A. Rao and R. Lanphier, “Real Time Streaming Protocol (RTSP),” IETF, http://www.ietf.org/rfc/rfc2326.txt, 1998.

[60] G. Lam and D. Rossiter, “Streaming multimedia delivery in Web services based e-learning platforms,” in *Proc. IEEE Int. Conf. Advanced Learning Technologies 2007*, pp. 706–710.

[61] G. Lam and D. Rossiter, “A SOAP-based streaming content delivery framework for multimedia Web services,” in *Proc. IEEE Asia-Pacific Conf. Services Computing 2008*, pp. 1097–1102.

[62] F. Yergeau, “UTF-8, a transformation format of ISO 10646,” http://tools.ietf.org/html/rfc3629, 2003.

[63] C. Bournez, “Efficient XML Interchange Evaluation,” W3C, http://www.w3.org/TR/exi-evaluation/, 2009.

[64] C. Krasic, K. Li and J. Walpole, “The case for streaming multimedia with TCP,” in *Proc. IDMS 2001*, pp. 213-218, 2001.

[65] B. Wang, J. Kurose, P. Shenoy and D. Towsley, “Multimedia streaming via TCP: An analytic performance study,” *ACM Trans. Multimedia Comput. Commun. Appl.*, vol. 4, no. 2, pp. 1–22, May 2008.

[66] S. Heinzl, M. Mathes and B.M. Freisleben, “A Web service communication policy for describing non-standard application requirements,” in *Proc. Int. Sym. Applications and the Internet 2008*, pp. 40–47.

[67] The network simulator – ns-2, http://www.isi.edu/nsnam/ns/

[68] Xiph.org Test Media, http://media.xiph.org/video/derf/

[69] VLC media player, http://www.videolan.org/vlc/

[70] H. Schulzrinne, “RTP tools (version 1.18),” http://www.cs.columbia.edu/irt/software/rtptools/, 2010.

[71] Computer for Interaction and Communications Research Group, University of Naples Federico II, “Network tools and traffic traces,” http://www.grid.unina.it/Traffic/

[72] R. Fielding et al., “Hypertext transfer protocol - HTTP/1.1,” http://www.w3.org/Protocols/rfc2616/rfc2616.html, 1999.

[73] Q. Hyunh-Thu and M. Ghanbari, “Scope of validity of PSNR in image/video quality assessment,” *Electronics Letters*, vol. 44, no. 13, pp. 800-801, June 2008.

[74] J. Klaue, B. Rathke and A. Wolisz, “EvalVid – A Framework for Video Transmission and Quality Evaluation,” *Computer Performance Evaluation, Modelling Techniques and Tools, Lecture Notes in Computer Science*, vol. 2794, pp. 255-272, 2003.

[75] International Telecommunication Union, "P.800 : Methods for subjective determination of transmission quality," ITU, http://www.itu.int/rec/T-REC-P.800-199608-I/, 1996.

[76] J. Chen, G. Chan and V. Li, “Multipath routing for video delivery over bandwidth-limited networks,” *IEEE Journal on Selected Areas in Communications*, vol.22, no.10, pp. 1920- 1932, Dec. 2004.

[77] Z. Qian, S. Zhang, K. Yim and S. Lu, “Service oriented multimedia delivery system in pervasive environments,” *Journal of Universal Computer Science*, vol. 17, no. 6, pp. 961–980, 2011.

[78] S. Misra, G. Xue and D. Yang, “Polynomial time approximations for multi-path routing with bandwidth and delay constraints,” *INFOCOM 2009*, pp.558-566, 19-25 April 2009.