

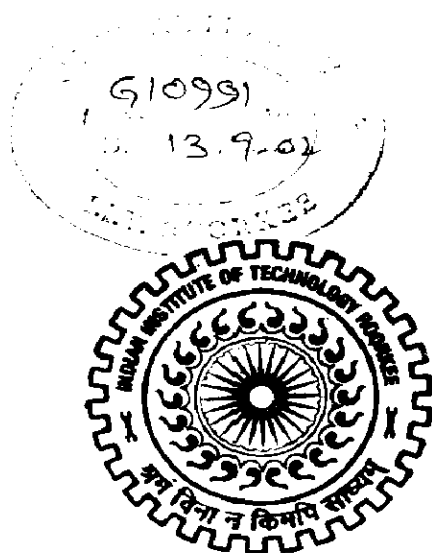
# MULTIMEDIA MESSAGING SERVICE

## A DISSERTATION

*Submitted in partial fulfilment of the  
requirements for the award of the degree  
of*  
MASTER OF COMPUTER APPLICATIONS

By

**HIMANSHU SINGH**



DEPARTMENT OF MATHEMATICS  
INDIAN INSTITUTE OF TECHNOLOGY ROORKEE  
ROORKEE-247 667 (INDIA)

MAY, 2002

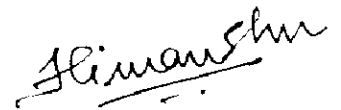
## CANDIDATE'S DECLARATION

I hereby declare that the work which is being presented in this report entitled "**Multimedia Messaging Service**" in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications (M.C.A.) in the Department of Mathematics, Indian Institute of Technology, Roorkee, is an authentic record of my own work carried out by me for a period of five months from January 2002 to May 2002 under the supervision and guidance of **Dr. Sunita Gakkhar**, Associate Professor, Department of Mathematics, I.I.T.-Roorkee and **Mr. Kanwaldeep Singh**, Technical Leader, Hughes Software Systems.

I have not submitted the matter embodied in this project work for the award of any other degree.

Date: 31/05/2002

Place: ROORKEE

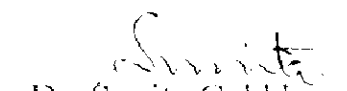


Himanshu Singh  
MCA 3<sup>rd</sup> year  
I.I.T. - Roorkee

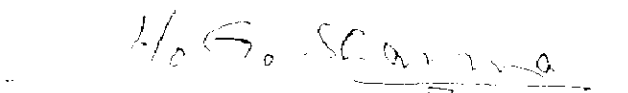
This is to certify that the above statements are correct to the best of my knowledge and belief.

Date: 31/05/2002

Place:



Dr. Sunita Gakkhar  
Associate Professor,  
Department of Mathematics,  
I.I.T. - Roorkee.



Professor & Head  
Department of Mathematics  
I.I.T. Roorkee-247 667

**TO WHOMSOEVER IT MAY CONCERN**

This is to certify that the project titled **“Multimedia Messaging Service”** being submitted by **Mr. Himanshu Singh**, a student of M.C.A. at Indian Institute of Technology, Roorkee, in partial fulfilment of the award of the degree of Master of Computer Applications, has been completed under my supervision at Hughes Software Systems, Gurgaon, during the period from January 2002 to May 2002.

This work has not been submitted to any other Institution or University for the award of any degree to the best of my knowledge.

*Kanwaldeep Singh*  
28/05/2002

**Kanwaldeep Singh**  
**Technical Leader**  
**Hughes Software Systems**  
**Gurgaon**

*[Faint, illegible text]*

## ACKNOWLEDGEMENT

My sincere thanks to Dr. Sunita Gakkhar, Associate Professor, Department of Mathematics, IIT- Roorkee, for her guidance, encouragement and constructive criticism throughout my dissertation.

It is my privilege to express my gratitude to my supervisor Mr. Kanwaldeep Singh, Technical Leader, Hughes Software Systems whose guidance has always been there to help me tackle my problems. His suggestions and encouragement throughout the project helped me in learning the various aspects of developing a project and completing the project in the stipulated time.

I would also like to thank all the members of the PPG team at Hughes for helping and inspiring me throughout the development of this project.

Himanshu Singh  
MCA III Year  
IIT, Roorkee

## Hughes Software Systems – A Profile

### Corporate History

Hughes Software Systems Limited (HSS) was incorporated in India on 30th December 1991 with Hughes Network Systems (HNS), a unit of Hughes Electronics Corporation (HUGHES), USA as its principal shareholder.

HUGHES, a world-leading provider of digital television entertainment, broadband services, satellite-based private business networks, and video and data broadcasting, reported revenues of \$7.3 billion in 2000.

HNS, a unit of HUGHES, is the world's largest provider of broadband satellite network solutions for businesses and consumers, with over 400,000 systems installed in more than 85 countries.

With the opening of the Indian economy in the early nineties, HUGHES was attracted by the long term potential of the emerging software services market and chose to set up a subsidiary in India to supplement HNS' R&D efforts.

HSS began its operations in New Delhi with a team of about 20 professionals and was initially focused on developing software solutions in the areas of VSAT-based networks for voice and data, cellular wireless telephony, packet switching and multi-protocol routing. Within three years, HSS grew to 240 professionals and in 1995, shifted its own campus to Electronic City, Gurgaon, a New Delhi suburb. Today, over 1600 professionals work on cutting edge technologies at six state-of-the-art facilities near New Delhi.

After having gained considerable expertise in the communications software business, HSS expanded its business and entered the market licensing products and offering its services to customers worldwide. Till date, the Company has delivered services and products to over 150 customers worldwide and it is now a leading technology supplier in the convergent market place.

In 1998, HSS set up a center in Bangalore to focus on development of software for Internet and Internet Protocol based solutions and applications. The Bangalore development center, housed in two state-of-the-art facilities can accommodate 250 professionals.

HSS added another first to its credit in 1999 when it went public, with India's first book built IPO. HSS also initiated its Voice over Packet Business to transform the communications market. HSS expanded operations to the USA in the same year by opening several offices. Continuing with its expansion activities, HSS opened marketing offices in U.K and Germany in 2000 and in Finland in the year 2001.

HSS has always been at the cutting edge of technology. This credo of excellence also extends to its employee initiatives. . HSS won the coveted CIO Award for usage of Information Technology in Human Resource Management. The award was in recognition of HSS' initiatives in using IT to create and drive its people practices. HSS was also nominated as the emerging company of the year by Economic Times and was voted as the top 10 most respected companies in India in a Business World survey in 2001. The ranking catapults HSS to the league of a few select companies in India

## Technology Focus

Hughes Software Systems (HSS), the #1 Communications Software company in India, offers a full spectrum of communications related software services, products and solutions. With over 40 customers spread over the Americas, Europe and Asia, HSS focuses on providing solutions to Telecom/ Datacom Equipment Manufacturers, System Integrators and Communication Services Providers.

The companies using solutions from HSS include ADC, Aheadcom, Alcatel, Brix Networks, Cisco, Coppercom, Embedded Wireless, Ericsson, IP Unity, IpVerse, Johnson Control, Lucent, Longboard, Megisto, Nokia, NEC, Radisys, Santerra, Shoreline, Snowshore, Spatial Wireless, Sylantron Shanghai Bell, SK telecom, Tektronics, Telspec, Think Engine, Tollbridge, Toshiba, Westwave, Winphoria, Zhone to name a few.

Continued significant investments in R&D have positioned HSS at the forefront of emerging communication technologies. The focus areas are Communication Protocols, Wireless Networks, Telecom/ Data Networks, Next Generation Networks, Intelligent Networks, Network Management, Internet and E-commerce.

## Software Products

HSS' products serve two major markets segments. The Original Equipment Manufacturer (OEM) offerings include Protocol Stacks, Signaling Interworking Functions, Gateways, SS7 ADF and Intelligent Peripherals. The TELCO offerings include Mediation Device Systems, Short Message Service Center (SMSC) and an Electronic Bill Presentment and Payment solution. These products are backed by porting, consulting and custom engineering services to help customers integrate and build solutions. HSS also offers comprehensive after sales support service to its customers.

HSS added another first to its credit in 1999 when it went public, with India's first book built IPO. HSS also initiated its Voice over Packet Business to transform the communications market. HSS expanded operations to the USA in the same year by opening several offices. Continuing with its expansion activities, HSS opened marketing offices in U.K and Germany in 2000 and in Finland in the year 2001.

HSS has always been at the cutting edge of technology. This credo of excellence also extends to its employee initiatives. . HSS won the coveted CIO Award for usage of Information Technology in Human Resource Management. The award was in recognition of HSS' initiatives in using IT to create and drive its people practices. HSS was also nominated as the emerging company of the year by Economic Times and was voted as the top 10 most respected companies in India in a Business World survey in 2001. The ranking catapults HSS to the league of a few select companies in India

### Technology Focus

Hughes Software Systems (HSS), the #1 Communications Software company in India, offers a full spectrum of communications related software services, products and solutions. With over 40 customers spread over the Americas, Europe and Asia, HSS focuses on providing solutions to Telecom/ Datacom Equipment Manufacturers, System Integrators and Communication Services Providers.

The companies using solutions from HSS include ADC, Aheadcom, Alcatel, Brix Networks, Cisco, Coppercom, Embedded Wireless, Ericsson, IP Unity, IpVerse, Johnson Control, Lucent, Longboard, Megisto, Nokia, NEC, Radisys, Santerra, Shoreline, Snowshore, Spatial Wireless, Sylantron Shanghai Bell, SK telecom, Tektronics, Telspec, Think Engine, Tollbridge, Toshiba, Westwave, Winphoria, Zhone to name a few.

Continued significant investments in R&D have positioned HSS at the forefront of emerging communication technologies. The focus areas are Communication Protocols, Wireless Networks, Telecom/ Data Networks, Next Generation Networks, Intelligent Networks, Network Management, Internet and E-commerce.

### Software Products

HSS' products serve two major markets segments. The Original Equipment Manufacturer (OEM) offerings include Protocol Stacks, Signaling Interworking Functions, Gateways, SS7 ADF and Intelligent Peripherals. The TELCO offerings include Mediation Device Systems, Short Message Service Center (SMSC) and an Electronic Bill Presentment and Payment solution. These products are backed by porting, consulting and custom engineering services to help customers integrate and build solutions. HSS also offers comprehensive after sales support service to its customers.

## Software Services

### *Encompassing all aspects of communication systems*

With a collective experience of over 5000 person-years in implementing communication software projects for its customers worldwide, HSS has successfully demonstrated high value addition and has enabled its customers to meet challenging time-to-profit objectives. HSS gets involved with its customers from the early stages of System definition, Specifications and System Design, and extends its involvement in conducting field trials and post-delivery support.

### *Dedicated Development Facility (DDF)*

A DDF can also be set up to work as an extended development center for the customer in India. This is useful when customers are looking for flexibility and dedicated resources. HSS is committed to preserving the customers' Intellectual Property Rights and adopting their software engineering processes. Based on its strong expertise in all communication technologies and applications, HSS is involved in the design and development of Switching Systems, Mobile Satellite Communication Systems, Cellular Infrastructure, Access Networks, Enterprise Networking Solutions, Network Management Solutions, VoIP Solutions and E-Commerce and internet based systems.

## Expert Areas

HSS communication software services consist of customized software development and systems design consulting services. The company undertakes full life-cycle software development, consisting of problem definition, system design, detail engineering, implementation and testing, post-delivery support and total project management. In depth domain expertise in a wide range of communication technologies and a large inventory of pre-engineered building blocks are two key intellectual assets of the company, that provide an unparalleled advantage to customers in cutting down product development time. The company's expertise lies in areas that include

- Network Management
- Digital Signaling and Media Processing
- Broadband Solutions
- Switching Technologies
- Wireless Solutions



# Abbreviations

<b>CGI</b>	Common Gateway Interface
<b>ESMTP</b>	Extended Simple Mail Transfer Protocol
<b>HTTP</b>	HyperText Transport Protocol
<b>IANA</b>	Internet Assigned Numbers Authority
<b>IMAP</b>	Internet Message Access Protocol
<b>ISDN</b>	Integrated Services Digital Network
<b>IPv4</b>	Internet Protocol version 4
<b>IPv6</b>	Internet Protocol version 6
<b>MIME</b>	Multipurpose Internet Mail Extensions
<b>MM</b>	Multimedia Message
<b>MMS</b>	Multimedia Messaging Service
<b>MS</b>	Mobile Station
<b>MSISDN</b>	Mobile Station International Subscriber Directory Number
<b>OTA</b>	Over The Air
<b>PDU</b>	Protocol Data Unit

<b>PLMN</b>	Public Land Mobile Network
<b>POP</b>	Post Office Protocol
<b>RFC</b>	Request For Comments
<b>SMIL</b>	Synchronized Multimedia Integration Language
<b>S/MIME</b>	Secure/Multipurpose Internet Mail Extensions
<b>SMS</b>	Short Message Service
<b>SMTP</b>	Simple Mail Transfer Protocol
<b>URI</b>	Uniform Resource Identifier
<b>WAP</b>	Wireless Application Protocol
<b>WIM</b>	WAP Identity Module
<b>WINA</b>	WAP Interim Naming Authority
<b>WML</b>	Wireless Markup Language
<b>WSP</b>	Wireless Session Protocol

## Definitions

### **Email Server**

A generic class of servers that nominally hosts email services that operate using the SMTP, POP and/or IMAP protocols.

### **Multimedia Messaging Service (MMS)**

A system application by which a WAP client is able to provide a messaging operation with a variety of media types.

### **MMS Client**

The MMS service endpoint located on the WAP client device.

### **MMS Encapsulation**

The definition of the protocol data units, the fields and their encodings necessary to send and receive multimedia messages including multimedia objects.

### **MMS Terminal**

A mobile station (MS, terminal) that implements the MMS Client to provide the MMS service.

### **MMS Originating Terminal**

The MMS Terminal which sends a multimedia message.

### **MMS Recipient Terminal**

The MMS Terminal which receives a multimedia message.

### **MMS Proxy-Relay**

A server which provides access to various messaging systems. It may operate as a WAP origin server in which case it may be able to utilize features of the WAP system.

### **MMS Server**

A server that provides storage services and operational support for the MMS service.

### **Transaction**

One or more message exchanges that collectively are considered logically separate from other message exchanges.

### **WAP Origin Server**

A server that can deliver appropriate content upon request from a WAP client.

## Abstract

Short Messaging Service (SMS) has brought about a revolution in mobile telephony, and has gone a long way in popularizing cellular phones. The use of SMS is not restricted to passing a message from one cellphone to another. The users of the service have been using their creativity to frame new phrases, coin new acronyms, and come up with new emoticons. In fact SMS is often called 'the new language' of today's generation. But SMS, being a text-only service does restrict the users' creativity. Multimedia Messaging Service (MMS), on the other hand, would allow the user to compose messages with rich content. Users can include text, images, audio and video clips into their messages. The freedom of expression and creativity that this service offers is expected to bring about another revolution in the world of mobile messaging. In fact it is said that this revolution would have a similar impact on the cellphone industry as the transition from DOS to Windows had on the PC industry.

This document starts with explaining the technical details of Multimedia Messaging Service and later on describes a prototype of the service developed during the dissertation. The prototype implements a subset of the functionalities specified by the WAP forum. The idea was to develop a basic working model of MMS without making the design too complex. The prototype consists of the two essential components of the MMS System - an MMS Client and an MMS Proxy Relay.

The prototype was developed on a Sun Ultra-250 machine running Sun Solaris Operating System. The development and debugging tool used was Sun Workshop. Exceed from Hummingbird Communications was used as the X-Server to display and use Sun machine's Graphical Desktop on the local Windows 2000 based PC. In the final phases of development, Purify from Rational Software was used to find and remove memory leaks from the system.



# Table of Contents

<b>INTRODUCTION</b>	<b>4</b>
.1 Overview	4
.2 MMS Messaging Framework	4
1.2.1 Example use case	6
.3 MMS Client/MMS-Proxy-Relay Interface	7
.4 MMS Internet Email Interworking	7
1.4.1 Sending Messages To Internet Email Servers	8
1.4.2 Receiving Messages Sent From Internet Email Systems	8
1.4.3 Retrieving Messages From Internet Email Servers	8
.5 MMS Addressing	8
1.5.1 Internet Addressing	8
1.5.2 Wireless network addressing	8
<b>MMS CLIENT TRANSACTIONS</b>	<b>10</b>
.1 Overview	10
.2 Introduction to MMS Transaction Model	11
.3 MMS Client Transactions	13
2.3.1 MMS Client sending Message to MMS Proxy Relay	13
2.3.2 MMS Proxy-Relay Sending Notification to MMS Client	14
2.3.3 MMS Client Fetching Message from MMS Proxy-Relay	15
2.3.4 MMS Proxy-Relay Sending Delivery Report to MMS Client	17
2.3.5 Read Reports	18
<b>MMS ENCAPSULATION PROTOCOL</b>	<b>19</b>
.1 Overview	19
.2 Message Structure Overview	19
.3 MMS Protocol Data Units and Fields	21
3.3.1 Sending of Multimedia Message	21
3.3.2 Multimedia Message Notification	26
3.3.3 Retrieval Of Multimedia Message	29
3.3.4 Delivery Acknowledgement	31
3.3.5 Delivery Reporting	31
3.3.6 Read Reporting	32
.4 Error Considerations	33
3.4.1 Interoperability Considerations with Version Numbering	33

•

3.4.2	Interoperability between MMS Versions with the Same Major version number	33
3.4.3	Interoperability between MMS Versions with Different Major Version Numbers	33
<b>3.5</b>	<b>Binary Encoding of Protocol Data Units</b>	<b>34</b>
3.5.1	Basic Rules	34
3.5.2	Encoding Rules	35
3.5.3	Header Encoding	37
3.5.4	Assigned Numbers	41
<b>3.6</b>	<b>MMS Addressing Model</b>	<b>42</b>
<b>4</b>	<b>DESIGN OF THE MMS PROTOTYPE</b>	<b>44</b>
<b>4.1</b>	<b>Functionality Implemented</b>	<b>44</b>
4.1.1	Transactions Supported	44
4.1.2	Message Headers supported	44
<b>4.2</b>	<b>Architecture of the MMS Client</b>	<b>46</b>
4.2.1	Functional Decomposition	46
4.2.2	The UI	46
4.2.3	The HTTP Server	47
4.2.4	The MMS Client CGI Script	47
4.2.5	The MMS Client Driver	47
4.2.6	The MMS Client Transactions Stack	47
4.2.7	The MIME Encoder	48
4.2.8	The HTTP Client	48
<b>4.3</b>	<b>Internal interfaces</b>	<b>48</b>
4.3.1	The HTTP Server-MMS Client CGI Script Interface	48
4.3.2	The MMS Client CGI Script-MMS Client Driver Interface	48
4.3.3	The MMS Client Driver-User Interface	48
4.3.4	The MMS Client Driver-UI Interface	49
4.3.5	The MMS Client Driver-MMS CTS Interface	49
4.3.6	The MMS CTS-MIME Encoder Interface	49
4.3.7	The MMS CTS-HTTP Client Interface	50
<b>4.4</b>	<b>Architecture of the MMS Proxy Relay</b>	<b>51</b>
4.4.1	Functional Decomposition	51
4.4.2	The Message Handler	51
4.4.3	The HTTP Server	52
4.4.4	The SMTP Client	53
4.4.5	The Message Retriever Script	53
4.4.6	The MPR Script	53
4.4.7	The PUSH Client	53
4.4.8	The HTTP Client	54
<b>4.5</b>	<b>Internal Interfaces</b>	<b>55</b>
4.5.1	Message Handler-SMTP Client Interface	55
4.5.2	Message Handler-Push Client Interface	55
4.5.3	Push Client-HTTP Client Interface	55
4.5.4	Message Handler-Message Retriever Script Interface	55
4.5.5	Message Handler-MPR Script	55
4.5.6	Message Retriever Script-HTTP Server Interface	56
4.5.7	MPR Script-HTTP Server Interface	56



<b>5</b>	<b>CONCLUSION</b>	<b>57</b>
	<b>REFERENCES</b>	<b>58</b>
	<b>APPENDIX A: PUSH ARCHITECTURE OVERVIEW</b>	<b>59</b>
<b>A.1</b>	<b>The Push Framework</b>	<b>59</b>
<b>A.2</b>	<b>The Push Proxy Gateway</b>	<b>60</b>

# 1 Introduction

---

## 1.1 Overview

The Multimedia Messaging Service (MMS), as its name implies, is intended to provide a rich set of content to subscribers in a messaging context. It supports both sending and receiving of such messages by properly enabled client devices. An example of such a message is shown in Figure 1-1 below.

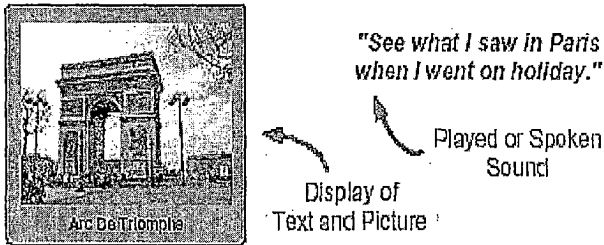


Figure 1-1 Example Message with Multimedia Content

The Multimedia messaging service is viewed as a non-real-time delivery system. This is comparable to many messaging systems in use today. Prime examples include traditional email available on the Internet and wireless messaging systems such as paging or SMS. These services provide a store-and-forward usage paradigm and it is expected that the MMS will be able to interoperate with such systems. Real-time messaging also exists in various forms. For example, instant messaging available from various vendors or various chat services (e.g. text, voice) are becoming popular. Such services are not currently supported with the MMS system but may be considered for future releases.

## 1.2 MMS Messaging Framework

A key feature of MMS is the ability to support messaging activities with other available messaging systems. This is shown in Figure 1-2 below which shows an abstract view of an MMS network diagram. It is expected that specific MMS networks may have one or more such connections as well as include specific messaging services not directly represented (e.g. fax or voice mail systems).

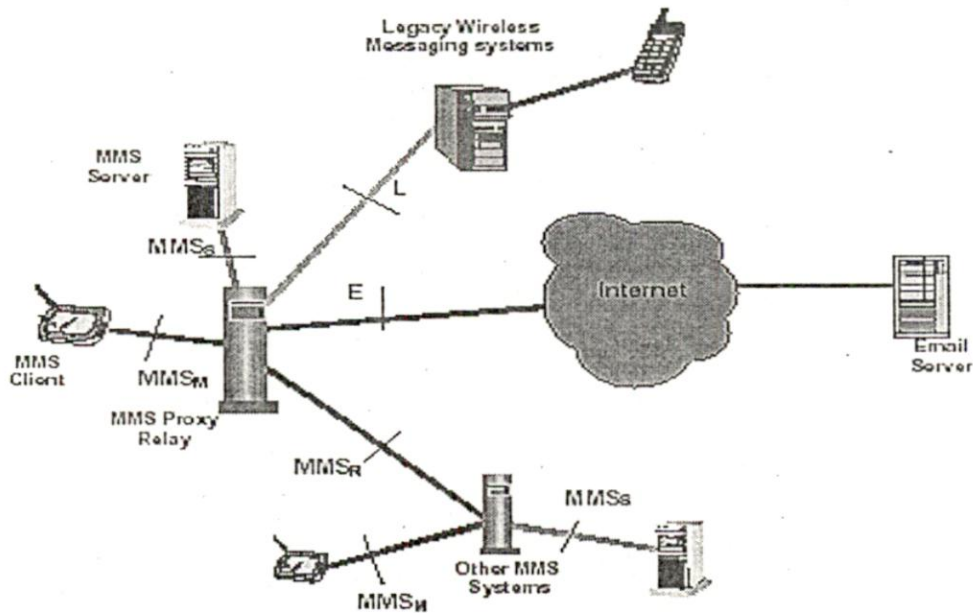


Figure 1-2 MMS Network Representation

The system elements shown in Figure 1-2 can be summarised as follows:

- **MMS Client** – This is the system element that interacts with the user. It is expected to be implemented as an application on the user’s wireless device.
- **MMS Proxy-Relay** – This is the system element that the MMS Client interacts with. It provides access to the components that provide message storage services, and it is responsible for messaging activities with other available messaging systems. Some implementations may combine this component with the MMS Server.
- **MMS Server** – This system element provides storage services for MM messages. Some implementations may combine this component with the MMS Proxy -Relay.
- **Email Server** – This system element provides traditional Internet email services. It supports the SMTP protocol to send messages as well as POP and/or IMAP protocols to retrieve messages.
- **Legacy Wireless Messaging Systems** – This system element represents various systems that currently exist in support of wireless messaging systems. This would include paging and SMS systems that provide messaging to a large number of subscribers.

The interfaces shown in the diagram are described as follows:

- **MMS<sub>m</sub>** – the interface defined between the MMS Client and the MMS Proxy -Relay.
- **MMS<sub>S</sub>** - the interface defined between the MMS Server and the MMS Proxy -Relay. This interface may be transcendental when the MMS Server and MMS Proxy -Relay are combined into a single component. The need to define this interface has not yet been established.
- **MMS<sub>R</sub>** - the interface defined between MMS Proxy -Relays of separate MMS Systems, see section 9. Currently, there is no specification that defines this interface. This interface may be standardised in the future by the WAP forum or by some other standardisation body.
- **E** - the standard email interface used between the MMS Proxy -Relay and internet-based email systems utilising SMTP, POP and IMAP transport protocols. This interface may be standardised in the future by the WAP forum or by some other standardisation body.
- **L** - the interfaces used between the MMS Proxy -Relay and legacy wireless messaging systems. As there are various such systems, this is viewed as being a set of interfaces. This interface may be standardised in the future by the WAP forum or by some other standardisation body.

### 1.2.1 Example use case

The following example information flow for a use case is provided to further illustrate the functions and roles of the various system elements in the MMS framework. The example given here concerns end-to-end MMS messaging between terminals.

1. User activates MMS Client (assumed to be available on terminal).
2. User selects or enters MM target address(es).
3. User composes/edits MM message to be sent.
4. User requests that MM message is sent.
5. MMS Client submits the message to its associated MMS Proxy -Relay via the MMSM interface.
6. MMS Proxy-Relay resolves the MM target address(es).
7. MMS Proxy-Relay routes forward the MM to each target MMS Proxy -Relay via the MMSR interface.
8. The MM is stored by the MMS Server associated with the target MMS Proxy -Relay.
9. Target MMS Proxy -Relay sends a notification to target MMS Client via the MMSM interface.
10. Target MMS Client retrieves the MM from the MMS Server.
11. Target MMS Client notifies target user of new MM available.
12. Target user requests rendering of received MM message.
13. Target MMS Client renders MM message on target user's terminal.

Note that steps 10 and 11 could occur in reverse order depending on MMS Client implementation, that is, an MM retrieval policy could cause the MMS Client to retrieve an MM only when so allowed by the user.

### 1.3 MMS Client/MMS-Proxy-Relay Interface

As shown in Figure 1-2, the MMS Client interacts with the MMS Proxy - Relay. This operation is consistent with the WAP model where the MMS Proxy - Relay operates as an Origin Server (Pull Operations) or as a Push Initiator (Push Operations). The relationship between the MMS Client and MMS Proxy -Relay is shown in Figure 1-3 below. The messages that transit between the two components are normally transferred using a wireless transport such as WSP between the WSP Client and the WAP Gateway, and then transit over HTTP from the WAP Gateway to the MMS Proxy -Relay.

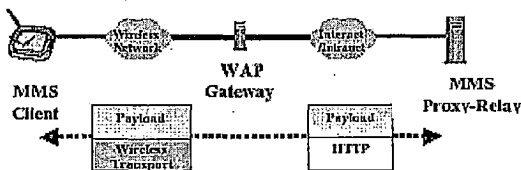


Figure 1-3 MMS Client/MMS Proxy Relay Link

This network representation includes a few items that need to be described. The MMS Proxy -Relay is the network entity that interacts with the user mailbox and is responsible for initiating the notification process to the MMS Client. The WAP Gateway provides standard WAP services needed to implement MMS, these include: HTTP methods; PUSH services; and OTA security. The MMS system is guided by activities between the MMS Client and MMS Proxy -Relay. These activities are described in the WAP MMS Client Transaction document [2] and the MMS Encapsulation document [3].

The network view also shows a payload that is carried by the wireless transport and HTTP. This payload is described in the WAP MMS Message Encapsulation document [3]. It is expected that this data will be transported in its entirety between the MMS Proxy -Relay and the User's Terminal.

### 1.4 MMS Internet Email Interworking

One of the important links on the Network Diagram is the connection of the MMS Proxy Relay to Email Servers connected via the Internet. This connectivity works in both directions.

### 1.4.1 Sending Messages To Internet Email Servers

For sent messages, the MMS Proxy -Relay will submit the message to the addressed host using the SMTP protocol. The message payload will be converted to standard Internet MIME format to permit the various media components to be carried consistently into the Internet environment. The MMS specific header fields will be converted into appropriate headers by prepending an 'X-Mms-' to the header name. This will permit MMS aware systems to understand the fields while not being problematic for non-MMS aware systems.

### 1.4.2 Receiving Messages Sent From Internet Email Systems

Received messages will be similarly converted. The MIME part of the message will be converted to the MMS format. Similarly, any headers found with a prefix of 'X-Mms-' can be converted back to the associated MMS header.

### 1.4.3 Retrieving Messages From Internet Email Servers

It will be important for MMS Clients to be able to retrieve messages that are stored on Internet Email servers. This is normally done through the use of the POP or IMAP protocols. Such retrievals are performed by the MMS Proxy-Relay (this is one of the proxy roles), which will then convert the data into an appropriate MMS format.

## 1.5 MMS Addressing

An important aspect of messaging systems is the ability to address the users in a way that can be efficient for the system as well as meaningful for the senders of messages. This balance is difficult to achieve.

### 1.5.1 Internet Addressing

In the Internet world, where bandwidth is not a primary consideration, addresses are normally expressed in the email address paradigm. In this scheme, addresses look like *user@system* where the system specification may be a domain name or a fully qualified host address. In general, this scheme provides users the ability to have a complete and unique address in an unbounded text string.

### 1.5.2 Wireless network addressing

In the wireless world, where bandwidth efficiency is critical, short address lengths and ease of user entry on limited keypads are the hallmarks of the various systems. For example, in GSM networks, a user's address is based upon the MSISDN number utilized by the device. Similarly, in many paging systems, users are assigned PINs that would permit a caller to deposit a message.

The MMS addressing model, as defined in [3], makes such a more direct or efficient addressing scheme available to MMS subscribers and services. This is seen as

particularly important for interoperability with legacy systems such as the above mentioned, and e.g. for mobile-to-mobile operation.

As message traffic has increased to wireless systems from the wireline world, most such systems have deployed servers that provide external entities the opportunity to address their email to the wireless subscribers directly. Many such systems utilize an *ID@carrier* ( e.g. [9811206998@essarcellphone.com](mailto:9811206998@essarcellphone.com)) approach to facilitate using these addresses for access from email systems.

MMS employs an extensible addressing scheme that permits a variety of addressing paradigms to be supported. More specific details on addressing can be found in the MMS encapsulation specification [3].

## 2 MMS Client Transactions

### 2.1 Overview

The Multimedia Messaging Service is described in terms of actions taken by the MMS Client and its service partner, the MMS Proxy-Relay, a device that operates as a WAP Origin Server for this specialised service. This chapter defines the operational flow of the messages that transit between the MMS Client and the MMS Proxy-Relay. The format of the specific messages is described in the next chapter “*MMS Encapsulation Protocol*”.

The MMS client transactions described in this chapter take place on the interface labeled  $MMS_M$  in the Figure 1-2. The following figure presents an amplified view of the  $MMS_M$  link. It is built on top of the WAP architecture. In its role as an application, MMS provides for the delivery and services related to messaging and the data schemes that will permit presentation methods that provide for the multimedia user experience. These presentation methods are separate from MMS.

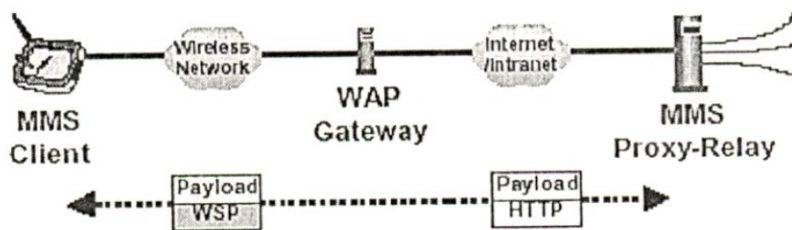


Figure 2-1 Implementation of MMSM Interface

This figure includes a few items that need to be described. The MMS Proxy-Relay is the network entity that interacts with the user mailbox and is responsible for initiating the notification process to the MMS Client. The WAP Gateway provides standard WAP services needed to implement MMS, these include: WSP invocation of HTTP methods, see [4]; WAP PUSH services, see [5]; OTA security; and, Capability Negotiations, see [11]. The above figure also shows a payload that is carried by WSP and HTTP. This payload is described in the MMS Message Encapsulation [3] document. It is expected that this data will be transported in its entirety between the MMS Proxy-Relay and the MMS Client.

This description does not address issues related to the movement or acquisition of MM messages beyond the MMS Proxy-Relay as these are outside the scope of the  $MMS_M$  link.



## 2.2 Introduction to MMS Transaction Model

The MMS service is realized by the invocation of transactions between the MMS Client and the MMS Proxy-Relay. These transactions include information and affect state changes on these devices. This section introduces example transaction flows and later sections describe each individual, logically separate transaction in more detail.

The general transaction flows on  $MMS_M$  for sending and retrieving MM messages do not depend on what type of client the MM message is sent to or received from. The other endpoint for the MM message may be another MMS Client served by the same or another MMS Proxy-Relay, it may be a client on a legacy wireless messaging system, or it may be an e-mail server.

The following three figures provide general views of the  $MMS_M$  transactions needed for:

- 1) an MMS Client to send an MM message and receive back a resulting delivery notice;
- 2) an MMS Client to perform immediate retrieval of a new MM message; and,
- 3) MMS Client to perform delayed retrieval of a new MM message. The arrow labels in the following figures indicate the MMS messages (also known as MMS PDUs) exchanged during transactions. These messages are defined in detail in [3].

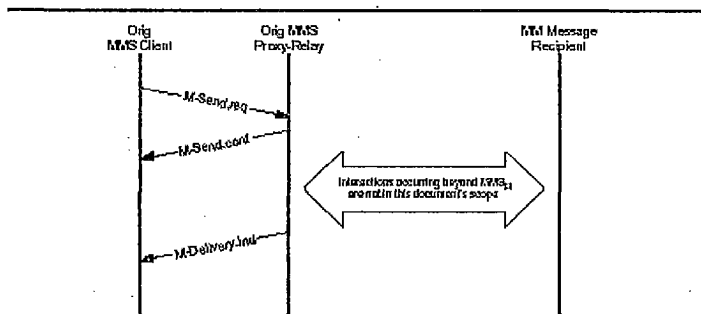


Figure 2-2 Example  $MMS_M$  Transaction Flow - Sending

A receiving MMS Client is said to perform immediate retrieval of a new MM message when it retrieves the data from the MMS Proxy-Relay before acknowledging the message notification.

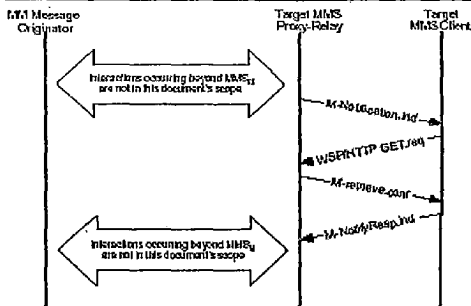


Figure 2-2 Example MMS<sub>M</sub> Transaction Flow – Immediate Retrieval

A receiving MMS Client is said to perform delayed retrieval of a new MM message when it first acknowledges the notification and at some later point retrieves the message from the MMS Proxy -Relay.

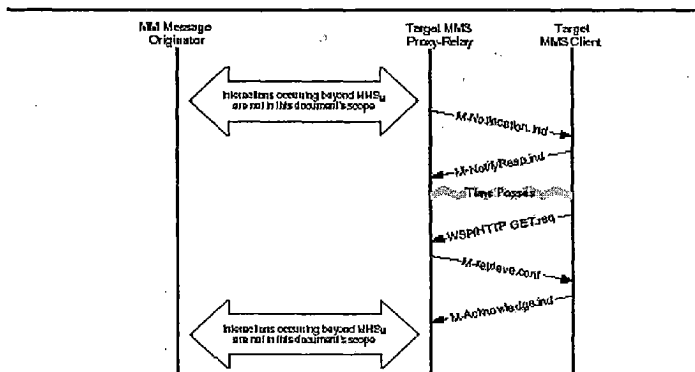


Figure 2-3 Example MMSM Transaction Flow – Delayed Retrieval

If both endpoints for the MM message exchange are MMS Clients, the MMS<sub>M</sub> interface is involved both when the originating MMS Client sends the MM message to the originating MMS Proxy -Relay and when the target MMS Client retrieves the MM message from the target MMS Proxy -Relay. The following figure shows an example where both endpoints are MMS Clients and delayed retrieval is used.

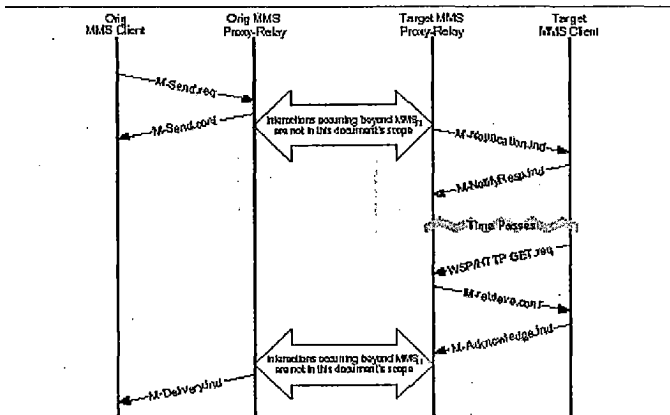


Figure 2-4 Example MMSM Transaction Flow – Delayed Retrieval

As can be seen in these examples, several message exchanges occur on MMS<sub>M</sub>. These message exchanges can be considered to form the following logically separate transactions:

- MMS Client Sending Message to MMS Proxy -Relay
- MMS Proxy-Relay Sending Notification to MMS Client
- MMS Client Fetching Message from MMS Proxy -Relay
- MMS Proxy-Relay Sending Delivery Report to MMS Client

## 2.3 MMS Client Transactions

The PDUs and information elements referred to in the following comply with the definitions in [3].

### 2.3.1 MMS Client sending Message to MMS Proxy Relay

The process for a client to send a message is built on top of the M-Send transaction. It provides the mechanism for the MMS Client to submit an MM message to the MMS Proxy -Relay and to get back information in response. The following figure gives an example of this transaction.

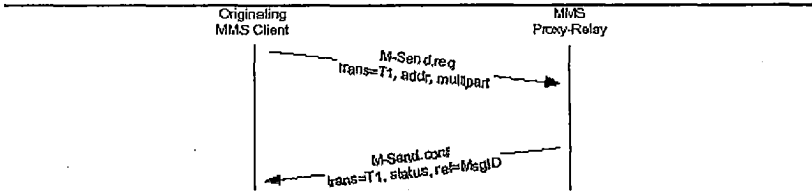


Figure 2-5 Example M-Send Transaction Flow

The MMS Client that wishes to send an MM message invokes a WSP/HTTP POST operation with the M-Send.req message embedded as the content body. This message is submitted using a URI that addresses the MMS Proxy-Relay that supports the specific MMS Client.

The MMS Client should compose a transaction ID for the submitted message. This transaction ID is used by the MMS Client and MMS Proxy -Relay to provide linkage between the originated M-Send.req and the response M-Send.conf messages. The value used for the transaction ID is determined by the MMS Client and no interpretation is expected by the MMS Proxy -Relay.

Upon receipt of the M-Send.req message, the MMS Proxy-Relay should respond to the WSP/HTTP POST with a response that includes the M-Send.conf message in its body. This response message provided a status code for the requested operation. If the MMS Proxy-Relay is willing to accept the request to send the message, the status should be 'accepted' and the message includes a message-ID that may be used for following activities that need to refer to the specific sent message (e.g. delivery reports).

### 2.3.2 MMS Proxy-Relay Sending Notification to MMS Client

To inform an MMS Client that an MM message is available and for it to return back information, a set of asynchronous messages, M-Notification.ind and M-NotifyResp.ind, are utilized. This provides the mechanism for the MMS Proxy -Relay to notify the MMS Client with certain factors about the new MM. This will let the MMS Client retrieve the MM.

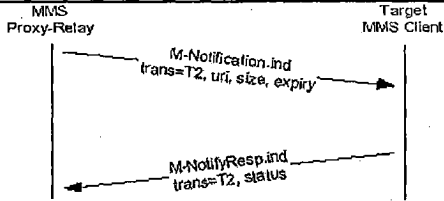


Figure 2-6 Example MMS Notification of MM message to Target Client

The MMS Proxy-Relay should utilise the M-Notification.ind message when it needs to inform the MMS Client that a message is available for delivery. The M-Notification.ind message should be sent by the MMS Proxy-Relay to the MMS Client using the WAP PUSH framework [PUSHARCH]. The M-Notification.ind message should be sent as the message body of a [PUSHMSG]. The X-Wap-Application-Id message header of that push message MUST be set to 'x-wap-application:mms.ua' if the absoluteURI form of the app-id syntax is used, and MUST be set to '4' if the app-assigned-code form of the app-id syntax is used.

The information conveyed should include an RFC2396 compliant URI that will be used to actually retrieve the message in a subsequent operation by the MMS Client. Additional information about the message (e.g. message size, expiry) may be used by the MMS Client to determine its behaviour. For example, the MMS Client may delay the retrieval of the message until after a user confirmation if it exceeds a size threshold. The MMS Proxy -Relay should compose a transaction ID for the notification message. This transaction ID is used by the MMS Client and MMS Proxy -Relay to provide linkage between the originated M-Notification.ind and the response M-NotifyResp.ind messages. The value used for the transaction ID is determined by the MMS Proxy-Relay and no interpretation is expected by the MMS Client.

Upon receipt of the M-Notification.ind message, the MMS Client should respond by invoking a WSP/HTTP POST operation with an M-NotifyResp.ind message embedded as the content body. This message is submitted using a URI that addresses the MMS Proxy -Relay that supports the specific MMS Client. The MMS Client SHOULD ignore the associated WSP/HTTP POST response from the MMS Proxy -Relay. The M-NotifyResp.ind response message should provide a message retrieval status code. The status 'retrieved' should be used only if the MMS Client has successfully retrieved the MM message prior to sending the NotifyResp.ind response message.

### 2.3.3 MMS Client Fetching Message from MMS Proxy-Relay

The operation for retrieval of the MM message by the MMS Client from the MMS Proxy -Relay is built upon the normal WSP/HTTP GET functionality. Therefore, no new operation is actually defined. The message type for the message returned from the MMS

Proxy -Relay to the MMS Client is M-retrieve.conf. Delivery of the MM message may be either before or after the M-NotifyResp.ind message, depending on immediate retrieval or delayed retrieval of MM message respectively. The MMS Proxy -Relay may therefore decide to request an acknowledgement from the MMS Client to confirm successful retrieval in case of delayed retrieval. These variations are shown in Figure 2-7 and Figure 2-8 respectively.

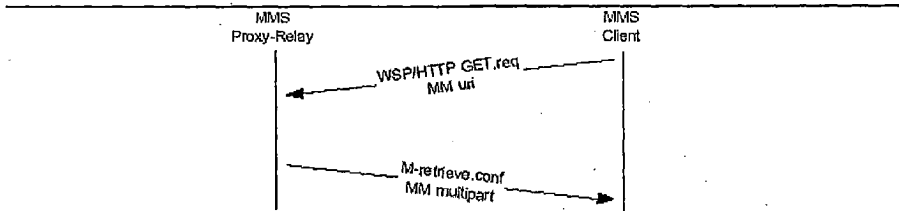


Figure 2-7 MMS Retrieval Transaction without Acknowledgement

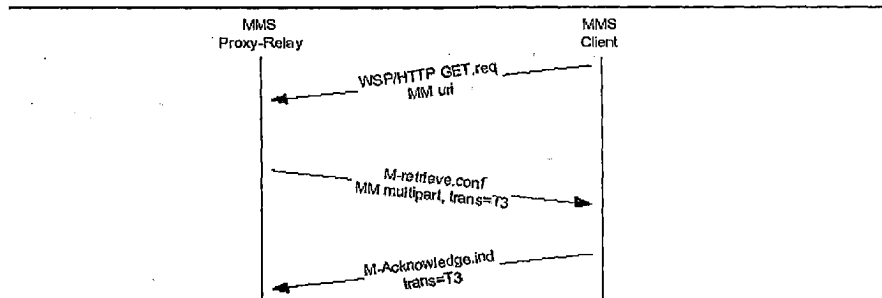


Figure 2-8 MMS Retrieval Transaction with Acknowledgement

The MMS Client should initiate the retrieval activity by utilizing the URI that was delivered to it in the M-Notification.ind message using the connection oriented mode of the normal WSP/HTTP GET method operation.

When setting up the WSP session and when sending the GET request the MMS Client should convey the capabilities of the terminal and of the MMS Client.

The response message M-retrieve.conf, if successful, contains the MM message. This MM message should include MMS headers providing additional information.

Depending on the MMS Proxy -Relay needs, the M-retrieve.conf response that it provides may request an acknowledgement to be generated by the MMS Client. The MMS Proxy -Relay may make this request based on whether or not it needs to provide a delivery notice back to the originator of the MM message. Alternatively, it may make that

request based upon an expectation that it would then be able to delete the message from its own store. This decision is not a part of this transaction.

The MMS Proxy Relay should make this request for acknowledgement by including a transaction ID in the M-retrieve.conf message. This transaction ID is used by the MMS Client and MMS Proxy-Relay to provide linkage between the originated M-retrieve.conf and the response M-acknowledge.ind messages. The value used for the transaction ID is determined by the MMS Proxy -Relay and no interpretation is expected by the MMS Client.

If an acknowledgement is requested, the MMS Client should respond by invoking a WSP/HTTP POST operation with an M-Acknowledge.ind message embedded as the content body. This message is submitted using a URI that addresses the MMS Proxy -Relay that supports the specific MMS Client. The MMS Client should ignore the associated WSP/HTTP POST response from the MMS Proxy -Relay. The M-Acknowledge.ind message confirms successful MM message retrieval to the MMS Proxy Relay.

### 2.3.4 MMS Proxy-Relay Sending Delivery Report to MMS Client

To permit the originating MMS Client to know when a message delivery has occurred the M-Delivery.ind message has been defined to provide that information. The M-Delivery.ind message originates at the MMS Proxy-Relay providing information to the MMS Client about the message that was delivered. There is no associated response or acknowledgment message. The following Figure 2- shows an example of this message.

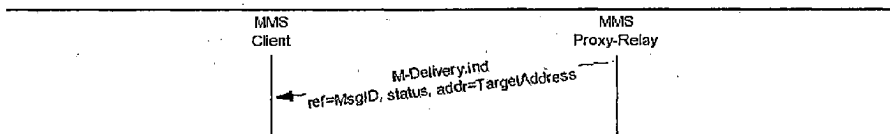


Figure 2-9 Example Delivery Report Transaction

The M-Delivery.ind message should be sent by the MMS Proxy -Relay to the MMS Client using the WAP PUSH framework [5]. The M-Delivery Delivery.ind .message should be sent as the message body of a PUSH message as described in [6]. The X-Wap-Application-Id message header of that push message MUST be set to 'x-wapapplication:mms.ua' if the absoluteURI form of the app-id syntax is used, and MUST be set to '4' if the appassigned-code form of the app-id syntax is used.

The M-Delivery.ind message conveys information about the status of a particular message delivery that was performed. The message is identified by the Message ID that was generated when the original message was posted. It also provides addressing information of the originally targeted entity.

If an MM message was addressed to multiple entities, multiple M-Delivery.ind messages SHOULD be expected to be returned, one for each addressed entity.

### 2.3.5 Read Reports

When the originating MMS Client requests the Read-Reply in a multimedia message, the receiving MMS Client may send a read message back to it. This message is sent and delivered using the normal mechanisms as described in this section.

To permit a user to determine that a message is a read reply, a few fields can be used to provide that information:

- The subject field should be copied from the original, prepending a 'Read:' to the text.
- The Message-ID of the original message is available and should be included in the message body.
- The body of the message may provide information about the read action or status.

The following is an example of a read reply message. It is in response to a message that user A had sent to user B:

```
From: B
To: A
Sent: Friday, January 21, 2000 1:50 PM
Subject: Read: My Message
Your message

      To: B
      Subject: My Message
      Message-ID: <200002211806.MAA26265@mail1.domain.com>
      Sent: 1/21/2000 1:29 PM

was read on 1/21/2000 1:50 PM.
```

**Figure 2-10 Example for Read Report**

If supported by a receiving MMS Client, the read reply message is sent to the MMS Proxy -Relay when an MM message has been read that had been flagged with the Read-Reply flag. The message should be sent using the normal M-Send operation as it is just another message origination. As such, it should be delivered using the normal delivery methods. Due to the nature of the message, the Message Message-Class field should have the value 'Auto', the Read Read-Reply flag must not be set, and the Delivery Delivery-Report flag must not be set in a read-reply message.

The MMS Client receiving a read message will see it as a new message. The interpretation as a read-reply is done by context. In cases where the original message had multiple addresses, the MMS Client should expect that multiple read-reply messages will be returned.



## 3 MMS Encapsulation Protocol

---

### 3.1 Overview

This chapter describes the content and encodings of the protocol data units (PDUs) for the multimedia messaging service.

In multimedia messaging service the WAP WSP/HTTP is used to transfer multimedia messages between the terminal (MS) and the MMS Proxy -Relay. The WSP session management and the related capability negotiation mechanisms as well as security functions are out of the scope of this document.

There are basically eight types of PDUs used in MMS transactions:

- Send message to MMS Proxy -Relay(M-Send.req, M-Send.conf)
- Fetch message from MMS Proxy -Relay (WSP/HTTP GET.req, M -Retrieve.conf)
- MMS Notification about new message (M-Notification.ind, M-NotifyResp.ind)
- Delivery Report about sent message (M-Delivery.ind)
- Acknowledgement of message delivery (M-Acknowledge.req)

Logically the PDU consists of headers and a multipart body. The multipart body is present only as a sent multimedia message and a successfully fetched message. Some of the headers originate from standard RFC 2822 headers and others are specific to the multimedia messaging.

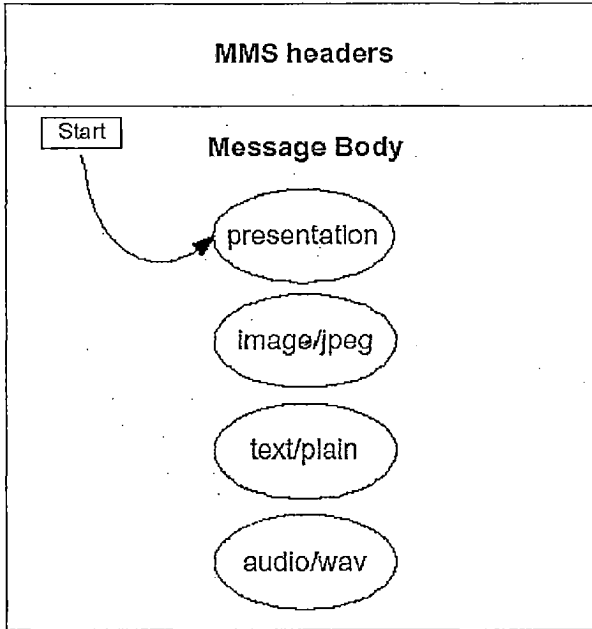
According to WSP definitions, comma separated lists of header field values are coded as multiple headers with identical name. If the headers are converted from binary encoding to textual format, several header fields with the same name are combined into a comma separated list, and vice versa. The order of the header fields is preserved.

The textual format of the headers is that defined in RFC2822 and RFC 2616. Binary encoding is similar to WSP header encoding [4]. In this specification, values for header fields and parameter names are assigned.

### 3.2 Message Structure Overview

The multimedia messaging PDUs consists of MMS headers and a message body. The message body may contain any content type, including preassigned content types defined in [4]. The MIME multipart RFC2045-RFC2047 is used in email systems and are therefore compatible. The content type of the PDUs is application/vnd.wap.mms -message. The WSP content type application/vnd.wap.multipart.related content type provides a good example how multimedia content and presentation information can be encapsulated to a single message. Figure 3-1 depicts the conceptual model and example of the encapsulation.

## application/vnd.wap.mms-message



**Figure 3-1 Model of MMS data Encapsulation**

The MMS-headers contain MMS-specific information of the PDU. This information contains mainly information how to transfer the multimedia message from originating terminal to the recipient terminal. In the multimedia messaging use case, the message body consists of multipart/related structure including multimedia objects, each in separate part, as well as optional presentation part. The order of the parts has no significance. The presentation part contains instructions how the multimedia content should be rendered to the display and speakers etc, on the terminal. There may be multiple presentation parts, but one of them must be the root part. In case of multipart/related, the root part is pointed from the Start parameter.

If the presentation part does not exist, it is up to the implementation of the terminal how the multimedia content is presented. Examples of the presentation techniques are SMIL and WML. The message body is used only when the multimedia message is sent or retrieved. All other PDUs contain only the mms-headers part. The message can contain various multimedia parts. Figure 3-1 shows just one possibility.

### 3.3 MMS Protocol Data Units and Fields

The header fields for sending, notification, retrieving, reporting and acknowledging of a multimedia message are described in the Tables 1-7. The names of the fields that do not originate from RFC2822 are preceded by X-Mms-. The MMS Protocol Data Units may contain additional Header fields such as found in standard RFC2822 headers which are not explicitly referenced in this document.

#### 3.3.1 Sending of Multimedia Message

The sending of the multimedia message consists of two messages: M-Send.req and M-Send.conf. The transaction identifier is created and used by the sending client and it is unique within the send transaction only.

##### 3.3.1.1 Send Request

This chapter describes messages sent by the MS to the MMSProxy -Relay, and those headers generated by the sender's MMS Proxy -Relay and added to the headers generated by the client. These headers are used to generate the MMS notification to the recipient, and are delivered with the message body parts to the recipient at retrieval.

In addition to the following tokens described in the table below, it is also possible to provide header extendability using WSP mechanism of encoding of a new unassigned header field name.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-send-req	Mandatory. Specifies the transaction type.
X-Mms-Transaction-ID	Transaction-id-value	Mandatory. A unique identifier for the message. This transaction ID identifies the M-Send.req and the corresponding reply only.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to this specification, the version is 1.0

Date	Date-value	Optional. Arrival time of the message at MMS Proxy - Relay. MMS Proxy Relay will generate this field when not supplied by terminal.
From	From-value	Mandatory. Address of the message sender. This field MUST be present in a message delivered to a recipient. The sending client MUST send either its address or insert-anaddress token. In case of token, the MMS Proxy -Relay MUST insert the correct address of the sender.
To	To-value	Optional. Any number of address fields allowed.
Cc	Cc-value	Optional. Any number of address fields allowed.
Bcc	Bcc-value	Optional. Any number of address fields allowed.
Subject	Subject-value	Optional. Subject of the message.
X-Mms-Message-Class	Message-class-value	Optional. Class of the message. Value Auto indicates a message that is automatically generated

		<p>by the client. If the Message-Class is Auto, the originating terminal should NOT request Delivery-Report or Read-Report.</p> <p>If field is not present, the receiver interprets the message as personal.</p>
X-Mms-Expiry	Expiry-value	<p>Optional, default: maximum.</p> <p>Length of time the message will be stored in MMS Proxy -Relay or time to delete the message. The field has two formats, either absolute or interval.</p>
X-Mms-Delivery-Time	Delivery-time-value	<p>Optional: default: immediate.</p> <p>Time of desired delivery. Indicates the earliest possible Delivery of the message to the recipient. The field has two formats, either absolute or interval.</p>
X-Mms-Priority	Priority-value	<p>Optional. Default: Normal.</p> <p>Priority of the message for the recipient.</p>
X-Mms-Sender-Visibility	Sender-visibility-value	<p>Optional. Default: show address/phone number of the sender to the recipient unless the sender has a secret number/address.</p> <p>Hide = don't show any address. Show = show even secret address.</p>
X-Mms-Delivery-Report	Delivery-report-value	<p>Optional. Default determined when service is ordered.</p>

		Specifies whether the user wants a delivery report from each recipient. When Message-Class is Auto, the field should always be present and the value should be No.
X-Mms-Read-Reply	Read-reply-value	Optional. Specifies whether the user wants a read report from each recipient as a new message. When message-Class is Auto, the field should always be present and the value should be No.
Content-Type	Content-type-value	Mandatory. The content type of the message.

**Table 1 Headers of the M-Send.req message**

Application-specific headers in M-Send.req provide technology that allows the use of application-specific extensions for multimedia messaging service which allows, e.g., the use of additional RFC2822 headers.

The message body follows the headers.

When the content type application/vnd.wap.multipart.related is used and if the Start parameter in the related structure is present, it must point to the presentation part of the multimedia message. If the Start parameter is not present, the presentation part, if present at all, MUST be the first part in the multipart structure.

### 3.3.1.2 Send confirmation

When the MMS Proxy –Relay has received the Send request, it sends a response message back to the MS indicating the status of the operation. The response message contains a the mms -headers only.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-send-conf	Mandatory. Identifies the message type
X-Mms-Transaction-ID	Transaction-id-value Response-text -value	Mandatory. This transaction ID identifies the M-Send.conf and the corresponding request only.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0
X-Mms-Response-Status	Response-status-value	Mandatory. MMS specific status.
X-Mms-Response-Text	Response-text -value	Optional. Description which qualifies the response status value.
Message-ID	Message-ID-value	Optional. This is a unique reference assigned to message. This ID should always be present when the MMS Proxy – Relay accepted the message.  The ID enables a client to match delivery reports with previously sent

	messages.
--	-----------

**Table 2 M-Send.conf message**

The MMS Proxy-Relay must always assign a message ID to the message when successfully received for delivery. The message ID shall be globally unique according to the needs of the MMS Proxy-Relay that receives the multimedia message for delivery.

### 3.3.2 Multimedia Message Notification

MMS Notifications inform the MS about the contents a received message. The MMS Notification message consists only of MMS headers. No other parts are present. The purpose of the notification is to allow the client to automatically fetch a MM from the location indicated in the notification.

The transaction identifier is created by the MMS Proxy-Relay and it is unique up to the following M-NotifyResp only.

If the MMS Client requests deferred delivery with M-NotifyResp, the MMS Proxy-Relay may create a new transaction identifier.

<b>Name</b>	<b>Content</b>	<b>Comments</b>
X-Mms-Message-Type	Message-type-value = m-notification-ind	Mandatory. Specifies the transaction type.
X-Mms-Transaction-ID	Transaction-id-value	Mandatory. Identifies the notification and the subsequent transaction that is closed by the following M-NotifyResp.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0.
From	From-value	Optional. Address of the sender. If hiding the address of the sender from the recipient



		is supported, the MMS Proxy Relay will not add this field to a message header.
Subject	Subject-value	Optional. Subject of the message.
X-Mms-Message-Class	Message-class-value	Mandatory. Class of the message.
X-Mms-Message-Size	Message-size-value	Mandatory. Full size of message in octets.
X-Mms-Expiry	Expiry-value	Mandatory. Length of time the message will be available. The field has only one format, interval.
X-Mms-Content-Location	Content-location-value	Mandatory. This field defines the location of the message.

Table 3 M-Notification.ind message

The standard URI format should be used to represent the content location, for example:

`http://mmsc/msgrtvr?msgid`

The confirmation of the notification is presented in Table 4. The purpose of the confirmation is to acknowledge the transaction to the MMS Proxy -Relay.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-notifyresp-Ind.	Mandatory. Identifies the message type
X-Mms-Transaction-ID	Transaction-id-value	Mandatory. Identifies the transaction started by M-Notification.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0
X-Mms-Status	Status-value	Mandatory. Message status. The status Retrieved should be used only after successful retrieval of multimedia message.
X-Mms-Report-Allowed	Report-allowed-value	Optional. Default: Yes. Sending of delivery report allowed to the user or not.

Table 4 M-NotifyResp.hnd message

### 3.3.3 Retrieval Of Multimedia Message

A client should retrieve messages by sending a WSP/HTTP GET request to the MMS Proxy -Relay containing a URI to the received message.

When successful, the response to the retrieve request will contain headers and the body of the incoming message.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-retrieve-conf	Mandatory. Specifies the message type.
X-Mms-Transaction-ID	Transaction-id-value	Optional. Identifies either the transaction that has been started by M-Notification without M-NotifResp or new transaction when deferred delivery was requested. The new transaction ID is optional.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0.
Message-ID	Message-ID-value	Optional. This is an unique reference assigned to message. This ID should always be present when the originator client Requested a read reply. The ID enables a client to match read reports with previously sent messages.
Date	Date-value	Mandatory. Sending date and time.
From	From-value	Optional. Address of the sender. If

		hiding the address of the sender from the recipient is supported, the MMS Proxy -Relay will not add this field to a message header.
To	To-value	Optional. Address of the recipient. Any number of address fields allowed.
Cc	Cc-value	Optional. Any number of address fields allowed.
Subject	Subject-value	Optional. Message subject
X-Mms-Message-Class	Message-class-value	Optional. Message class. If field is not present, the receiver Inteprets the message as personal.
X-Mms-Priority	Priority-value	Optional. Default: Normal Priority of the message.
X-Mms-Delivery- Report	Delivery-report-value	Optional. Default: No. Specifies whether the user wants a <i>delivery report</i> from each recipient.
X-Mms-Read-Reply	Read-reply-value	Optional. Default: No. Specifies whether the user wants a read report from each Recipient as a new message.
Content-Type	Content-type-value	Mandatory. The content type of the message.

Table 5 M-Retreive.conf message

Application-specific headers in M-Retrieve.conf provide technology that allows the use of application-specific extensions for multimedia messaging service which allows, e.g., the use of additional RFC2822 headers.

The message body follows the headers.

When the content type application/vnd.wap.multipart.related is used and if the Start parameter in the related structure is present, the client should expect it to point to the presentation part of the multimedia message.

### 3.3.4 Delivery Acknowledgement

An MMS Acknowledge message confirms the delivery of the message from the receiving terminal to the MMS Proxy - Relay.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-acknowledge-ind	Mandatory. Identifies the transaction type.
X-Mms-Transaction-ID	Transaction-id-value	Mandatory. This is the transaction number that originates from immediately previous M-Retrieve operation.
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0.
X-Mms-Report-Allowed	Report-allowed-value	Optional. Default: Yes. Sending of delivery report allowed to the user.

Table 6 M-Acknowledge.ind message

### 3.3.5 Delivery Reporting

A MMS Delivery Report must be sent from the MMS Proxy -Relay to the originating MS when the originator has requested a delivery report and the recipient has not explicitly requested for denial of the report. As for example, the recipient can request for denial of the Delivery Report by using the X-Mms-Report-Allowed field of M-Acknowledge.ind

or M-NotifyResp.ind message. There will be a separate delivery report from each recipient. There is no response message to the delivery report.

Name	Content	Comments
X-Mms-Message-Type	Message-type-value = m-delivery-ind.	Mandatory. Identifies the PDU type
X-Mms-MMS-Version	MMS-version-value	Mandatory. The MMS version number. According to the current specification, the version is 1.0
Message-ID	Message-ID-value	Mandatory. Identifier of the message. From Send request, connects delivery report to sent message in MS.
To	To-value	Mandatory. Needed for reporting in case of point-to-multipoint message.
Date	Date-value	Mandatory. Date and time the message was handled (fetched, expired, etc.) by the recipient or MMS Proxy-Relay.
X-Mms-Status	Status-value	Mandatory. The status of the message.

**Table 7 M-Delivery.ind message**

### 3.3.6 Read Reporting

When the originating terminal requested the Read-Reply in the multimedia message, the recipient terminal may send a new multimedia message back to the originating terminal when the user has read the multimedia message. The content of the multimedia message is a terminal implementation issue. The read-reply multimedia message **MUST** have the Message-Class as Auto in the message.

The MMS Proxy-Relay must deliver the read-reply message as ordinary multimedia message.

When the originating terminal receives the Read-Reply, it should not create delivery report or read-reply message.

## 3.4 Error Considerations

### 3.4.1 Interoperability Considerations with Version Numbering

The MMS version number is divided into two parts: major version number and minor version number. MMS versions with only minor version number differences should provide full backward compatibility. MMS versions with major version number differences should not provide backward compatibility.

### 3.4.2 Interoperability between MMS Versions with the Same Major version number

The following rules should be followed between different MMS versions having the same major version number but different minor version number.

When a terminal or proxy -relay receives a PDU containing a particular minor version number it may respond with a PDU containing a different minor version number.

Unless a specific behavior has been defined, the receiving terminal or proxy -relay should ignore all unrecognised fields and recognised fields with unrecognised values.

The receiving proxy -relay should respond to any unknown PDU with M-Send.conf with status value 'Errorunsupported-message'.

The receiving terminal should respond to any unknown PDU with M-NotifyResp.ind with status value 'Unrecognised'.

### 3.4.3 Interoperability between MMS Versions with Different Major Version Numbers

The following rules should be followed between specifications with different major version numbers.

The receiving proxy -relay should respond to any PDU having major version number which it does not support with MMS 1.0 M-Send.conf containing status value 'Error-unsupported-message'.

The receiving terminal should respond to any PDU having major version number which it does not support with MMS 1.0 M-NotifyResp.ind containing status value 'Unrecognised'.

If the receiving terminal or proxy -relay supports multiple major versions including the version number of the received PDU, it must respond to the received PDU with a PDU from the same major version.

All major MMS versions must support MMS 1.0 M-Send.conf and MMS 1.0 M-NotifyResp.ind.

### 3.5 Binary Encoding of Protocol Data Units

The basic encoding mechanism for multimedia messages originates from WSP specification [4], because this is very tight encoding intended to optimize amount of data transmitted over the air.

The encoded MMS messages are stored to the Data field of the Post, Reply and Push PDUs [4]. Thus, the MMS messages are not encoded into WSP headers using WSP codepage technique.

If user-defined headers are used, the mechanism described in the next section (Application-header) MUST be used.

In the encoding of the fields, the order of the fields is not significant, except that Message-Type, Transaction-ID and MMS-Version MUST be at the beginning of the message headers, in that order, and the content type MUST be the last header, followed by message body.

The definitions for non-terminals not found in this document MUST follow the definitions in [4].

Note: The term "non-terminal" comes from the same context as described in RFC2234.

In the encoding of the message body, the binary encoding specified in [4] should be used whenever available. Otherwise, text encoding is used.

#### 3.5.1 Basic Rules

The following rules are used through this document to describe the basic parsing constructs. The rules for Token, TEXT and OCTET have the same definition as per [7]. The mechanisms specified in this document are described in augmented BNF similar to that used by [7].

The notation <Octet N> is used to represent a single octet with the value N in the decimal system. The notation <Any octet M-N> is used for a single octet with the value in the range from M to N, inclusive.

Text-string = [Quote] \*TEXT End-of-string  
; If the first character in the TEXT is in the range of 128-255, a Quote character  
; must precede it.  
; Otherwise the Quote character must be omitted. The Quote is not part of the contents.

Token-text = Token End-of-string

Quoted-string = <Octet 34> \*TEXT End-of-string  
; The TEXT encodes an RFC2616 Quoted-string with the enclosing quotation-marks "<" removed



Extension-media = \*TEXT End-of-string  
; This encoding is used for media values, which have no well-known binary encoding

Short-integer = OCTET  
; Integers in range 0-127 shall be encoded as a one octet value with the most significant  
; bit set to one (1xxx xxxx) and with the value in the remaining least significant bits.

Long-integer = Short-length Multi-octet-integer  
; The Short-length indicates the length of the Multi-octet-integer

Multi-octet-integer = 1\*30 OCTET  
; The content octets shall be an unsigned integer value with the most significant octet  
; encoded first (big-endian representation).  
; The minimum number of octets must be used to encode the value.

Uintvar-integer = 1\*5 OCTET

Constrained-encoding = Extension-Media | Short-integer  
; This encoding is used for token values, which have no well-known binary encoding, or  
; when the assigned number of the well-known encoding is small enough to fit into  
; Short-integer.

Quote = <Octet 127>

End-of-string = <Octet 0>

The following rules are used to encode length indicators.

Value-length = Short-length | (Length-quote Length)  
; Value length is used to indicate the length of the value to follow

Short-length = <Any octet 0-30>

Length-quote = <Octet 31>

Length = Uintvar-integer

### 3.5.2 Encoding Rules

The following rules are used to encode headers in an MMS message:

Header = MMS-header | Application-header

MMS-header = MMS-field-name MMS-value

Application-header = Token-text Application-specific-value

Token-text = Text -string

MMS-field-name = Short-integer

Application-specific-value = Text -string

MMS-value =

Bcc-value |  
Cc-value |  
Content-location-value |  
Content-type-value |  
Date-value |  
Delivery-report-value |  
Delivery-time-value |  
Delta-seconds-value |  
Expiry-value |  
From-value |  
Message-class-value |  
Message-ID-value |  
Message-type-value |  
Message-size-value |  
MMS-version-value |  
Priority-value |  
Read-reply-value |  
Report-allowed-value |  
Response-status-value |  
Response-text -value |  
Sender-visibility-value |  
Status-value |  
Subject-value |  
To-value |  
Transaction-id-value

### 3.5.3 Header Encoding

#### 3.5.3.1 *Bcc field*

Bcc-value = Encoded-string-value

Addressing model is discussed in a later section.

#### 3.5.3.2 *Cc field*

Cc-value = Encoded-string-value

Addressing model is discussed in a later section.

#### 3.5.3.3 *Content-Location field*

Content-location-value = Uri-value

Uri-value = Text-string

URI value SHOULD be encoded per [RFC2616].

#### 3.5.3.4 *Content-Type field*

The Content-Type field is encoded as Content-type-value defined in [4]. Preassigned content-types can be found in [4]. The use of start-parameter in case of multipart/related is define in RFC2387 and SHOULD be encoded according to [4].

#### 3.5.3.5 *Date field*

Date-value = Long-integer

In seconds from 1970-01-01, 00:00:00 GMT.

#### 3.5.3.6 *Delivery-Report field*

Delivery-report-value = Yes | No

Yes = <Octet 128>

No = <Octet 129>

#### 3.5.3.7 *Delivery-Time field*

Delivery-time-value = Value-length (Absolute-token Date-value | Relative-token Delta-seconds-value)

Absolute-token = <Octet 128>

Relative-token = <Octet 129>

#### 3.5.3.8 *Delta-seconds-value*

Delta-seconds-value = Long-integer

### **3.5.3.9 Encoded-string-value**

Encoded-string-value = Text -string | Value-length Char-set Text -string

The Char-set values are registered by IANA .

### **3.5.3.10 Expiry field**

Expiry-value = Value-length (Absolute-token Date-value | Relative-token Delta-seconds-value)

Absolute-token = <Octet 128>

Relative-token = <Octet 129>

### **3.5.3.11 From field**

From-value = Value-length (Address-present-token Encoded-string-value | Insert-address-token )

Address-present-token = <Octet 128>

Insert-address-token = <Octet 129>

Addressing model is discussed in a later section.

### **3.5.3.12 Message-Class field**

Message-class-value = Class-identifier | Text-string

Class-identifier = Personal | Advertisement | Informational | Auto

Personal = <Octet 128>

Advertisement = <Octet 129>

Informational = <Octet 130>

Auto = <Octet 131>

The token-text is an extension method to the message class.

### **3.5.3.13 Message-ID field**

Message-ID-value = Text -string

Encoded as in email address as per [RFC822]. The characters "<" and ">" are not included.

### **3.5.3.14 Message-Type field**

Message-type-value = m-send-req | m-send-conf | m-notification-ind | m-notifyresp-ind | m-retrieve-conf | macknowledge-ind | m-delivery-ind

m-send-req = <Octet 128>

m-send-conf = <Octet 129>

m-notification-ind = <Octet 130>

m-notifyresp-ind = <Octet 131>

m-retrieve-conf = <Octet 132>

m-acknowledge-ind = <Octet 133>

m-delivery-ind = <Octet 134>

Unknown message types will be discarded.

### **3.5.3.15 Message-Size field**

Message-size-value = Long-integer

Message size is in bytes.

### **3.5.3.16 MMS-Version field**

MMS-version-value = Short-integer

The three most significant bits of the Short-integer are interpreted to encode a major version number in the range 1-7, and the four least significant bits contain a minor version number in the range 0-14. If there is only a major version number, this is encoded by placing the value 15 in the four least significant bits [4].

### **3.5.3.17 Priority field**

Priority-value = Low | Normal | High

Low = <Octet 128>

Normal = <Octet 129>

High = <Octet 130>

### **3.5.3.18 Read-Reply field**

Read-reply-value = Yes | No

Yes = <Octet 128>

No = <Octet 129>

### **3.5.3.19 Report-Allowed field**

Report-allowed-value = Yes | No

Yes = <Octet 128>

No = <Octet 129>

### 3.5.3.20 *Response-Status field*

Response-status-value =

Ok |  
Error-unspecified |  
Error-service-denied |  
Error-message-format-corrupt |  
Error-sending-address-unresolved |  
Error-message-not-found |  
Error-network-problem |  
Error-content-not-accepted |  
Error-unsupported-message

Ok = <Octet 128>  
Error-unspecified = <Octet 129>  
Error-service-denied = <Octet 130>  
Error-message-format-corrupt = <Octet 131>  
Error-sending-address-unresolved = <Octet 132>  
Error-message-not-found = <Octet 133>  
Error-network-problem = <Octet 134>  
Error-content-not-accepted = <Octet 135>  
Error-unsupported-message = <Octet 136>

Any other values should NOT be used. They are reserved for future use. The value Error-unsupported-message is reserved for version management purpose only.

### 3.5.3.21 *Response-Text field*

Response-text -value = Encoded-string-value

### 3.5.3.22 *Sender-Visibility field*

Sender-visibility-value = Hide | Show

Hide = <Octet 128>

Show = <Octet 129>

### 3.5.3.23 Status field

Status-value = Expired | Retrieved | Rejected | Deferred | Unrecognised

Expired = <Octet 128>

Retrieved = <Octet 129>

Rejected = <Octet 130>

Deferred = <Octet 131>

Unrecognised = <Octet 132>

The value Unrecognized is reserved for version management purpose only.

### 3.5.3.24 Subject field

Subject-value = Encoded-string-value

### 3.5.3.25 To field

To-value = Encoded-string-value

Addressing model is discussed in a later section.

### 3.5.3.26 Transaction-Id field

Transaction-id-value = Text-string

## 3.5.4 Assigned Numbers

The Table 8 contains the field name assignments.

Name	Assigned Number
Bcc	0x01
Cc	0x02
Content-Location	0x03
Content-Type	0x04
Date	0x05
Delivery-Report	0x06
Delivery-Time	0x07
Expiry	0x08
From	0x09
Message-Class	0x0A
Message-ID	0x0B

Message-Type	0x0C
MMS-Version	0x0D
Message-Size	0x0E
Priority	0x0F
Read-Reply	0x10
Report-Allowed	0x11
Response-Status	0x12
Response-Text	0x13
Sender-Visibility	0x14
Status	0x15
Subject	0x16
To	0x17
Transaction-Id	0x18

Table 8 Fieldname Assignments

### 3.6 MMS Addressing Model

The MMS addressing model contains two addresses: the address of the MMS Proxy -Relay and the address of the recipient user and terminal. The address of the MMS Proxy -Relay shall be the URI of MMS.Proxy -Relay given by the MMS service provider. Thus, the URI needs to be configurable in the terminal.

A notation for the address of the recipient user in the terminal needs to be defined. The addressing model allows only single user in the terminal, thus combining the address of the terminal and the user. WAP Push Drafting Committee has solved this issue by using ABNF RFC2234 notation for defining the address type in the WAP Push Proxy Gateway specification. The text below is copied from the PPG specification and edited for usage in this document.

The external representation of addresses processed by the MMS Proxy -Relay is defined using ABNF. The format is compatible with Internet e-mail addresses [9]. The MMS Proxy -Relay MUST be able to parse this address format, and it MUST be able to determine whether it supports the specified address type or not.

```
address = ( e-mail / device-address )
e-mail = "Joe User <joe@user.org>" ; corresponding syntax defined in
;RFC822 per header field

device-address = ( global-phone-number "/"TYPE=PLMN" )
                / ( ipv4 "/"TYPE=IPv4" )
                / ( ipv6 "/"TYPE=IPv6" )
```



/ ( escaped-value "/TYPE=" address-type )

address-type = 1\*address-char

; A network bearer address type identifier registered with WINA

address-char = ( ALPHA / DIGIT / "\_" )

value = 1\*( %x20-2E / %x30-3C / %x3E-7E )

escaped-value = 1\*( safe-char )

; the actual value escaped to use only safe characters by replacing

; any unsafe-octet with its hex-escape

safe-char = ALPHA / DIGIT / "+" / "-" / "." / "%" / "\_"

unsafe-octet = %x00-2A / %x2C / %x2F / %x3A-40 / %x5B-60 / %x7B-FF

hex-escape = "%" 2HEXDIG ; value of octet as hexadecimal value

global-phone-number = ["+"] 1\*( DIGIT , written-sep )

written-sep ={"-"/"."}

ipv4 = 1\*3DIGIT 3( "." 1\*3DIGIT ) ; IPv4 address value

ipv6 = 4HEXDIG 7( ":" 4HEXDIG ) ; IPv6 address per RFC 2373

Each value of a user-defined-identifier is a sequence of arbitrary octets. They can be safely embedded in this address syntax only by escaping potentially offending values. The conversion to escaped-value is done by replacing each instance of unsafe-octet by a hex-escape which encodes the numeric value of the octet.

Some examples of the mechanism:

To: 0401234567/TYPE=PLMN

To: +358501234567/TYPE=PLMN

To: Joe User <joe@user.org>

To: FEDC:BA98:7654:3210:FEDC:BA98:7654:3210/TYPE=IPv6

To: 195.153.199.30/TYPE=IPv4

Addresses using the /TYPE format SHOULD NOT contain anything else than what is specified in the examples. E-mail addresses can use the field as it is allowed by RFC822 specification.

The terminal MUST support at least one of the addressing methods. The addressing model may be expanded later to cover other formats of addresses, such as URI-based addressing as discussed in RFC2396.

## 4 Design of the MMS Prototype

---

The MMS Prototype developed in this project consisted of two components: an MMS Proxy Relay Prototype and an MMS Client Simulator. To keep the design simple, only a subset of the features described in the specifications were implemented.

The functionality supported by the prototype is summarized in the next section.

### 4.1 Functionality Implemented

#### 4.1.1 Transactions Supported

- The MMS Client sending message to the MMS Proxy Relay.
- The MMS Proxy Relay sending back the response to this message, indicating the status of the message and the message-id allotted.
- The MPR sending a notification to the MMS Client.
- The MMS Client sending a response to the Notification.
- The MMS Client fetching a message from the MPR.
- The MMS Client sending an acknowledgement after fetching the message.
- The MPR sending a Delivery Report to MMS Client.

#### 4.1.2 Message Headers supported

##### 4.1.2.1 *M-Send.req message*

- X-Mms-Message-Type
- X-Mms-Transaction-ID
- X-Mms-MMS-Version
- Date
- From
- To
- Cc
- Bcc
- Subject
- X-Mms-Expiry
- X-Mms-Delivery-Time
- X-Mms-Delivery-Report
- Content-Type

##### 4.1.2.2 *M-Send.conf message*

- X-Mms-Message-Type
- X-Mms-Transaction-ID
- X-Mms-MMS-Version
- X-Mms-Response-Status
- Message-ID

#### **4.1.2.3 *M-Notification.ind message***

- X-Mms-Message-Type
- X-Mms-Transaction-ID
- X-Mms-MMS-Version
- X-Mms-Status
- X-Mms-Report-Allowed

#### **4.1.2.4 *M-Retrieve.conf message***

- X-Mms-Message-Type
- X-Mms-Transaction-ID
- X-Mms-MMS-Version
- Message-ID
- Date
- From
- To
- Cc
- Subject
- X-Mms-Delivery-Report
- Content-Type

#### **4.1.2.5 *M-Acknowledge.ind message***

- X-Mms-Message-Type
- X-Mms-Transaction-ID
- X-Mms-MMS-Version
- X-Mms-Report-Allowed

#### **4.1.2.6 *M-Delivery.ind message***

- X-Mms-Message-Type
- X-Mms-MMS-Version
- Message-ID
- To
- Date
- X-Mms-Status

## 4.2 Architecture of the MMS Client

### 4.2.1 Functional Decomposition

The Figure 4-1 shows the functional decomposition of the MMS Client. The MMS Client consists of seven components – User Interface, MMS Client Driver, MMS Client Transactions Stack, HTTP Client, HTTP Server and MMS Client CGI Script. A description of each of these components is given below.

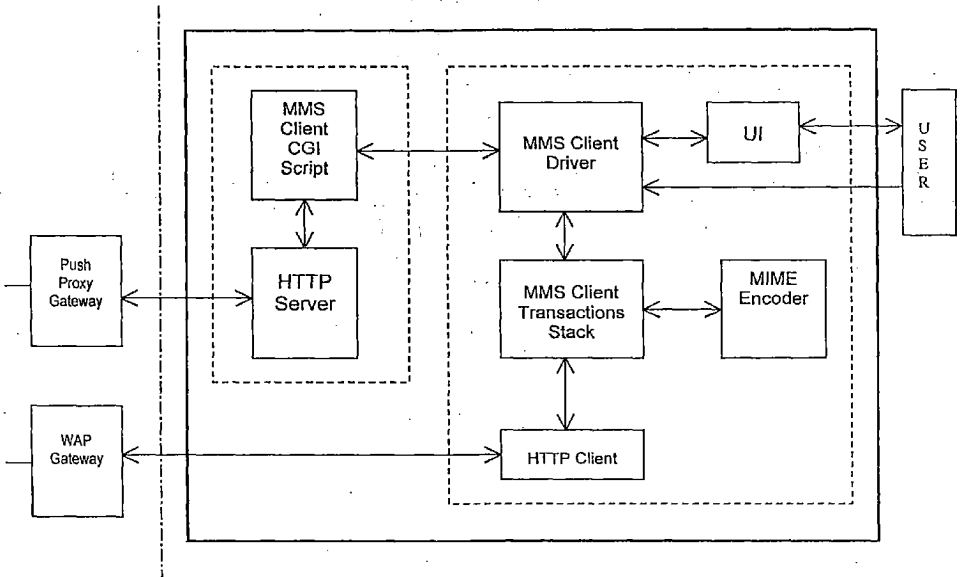


Figure 4-1 Functional Decomposition of the MMS Client

### 4.2.2 The UI

The User Interface does the actual interaction with the MMS user. It provides the user with facilities like composing and sending new messages, fetching and viewing received messages. The messages sent earlier (and stored in a sent-message store) can also be viewed along with their status code indicating the delivery status of the message. The delivery status is determined from the Delivery Report received for the message.

### 4.2.3 The HTTP Server

The HTTP server accepts HTTP POST requests from MMS clients, invokes the MMS Client CGI Script and passes the data received in the body of the POST request to the Script. It is also responsible for sending the response to the HTTP POST request. The HTTP server and the MMS Client CGI Script together provide the framework for receiving the PUSH messages from the MPR.

### 4.2.4 The MMS Client CGI Script

The MMS Client CGI Script receives the data from the HTTP Server, which is an MMS Notification or Delivery Report PDU PUSHed by the MPR and passes it on to the MMS Client Driver. It then receives any responses from the MMS Client Driver, adds headers indicating the response to the PUSH submission and sends it back to the HTTP server.

### 4.2.5 The MMS Client Driver

The MMS Client Driver waits for inputs from the user and the MMS Client CGI Script. If it receives input from the user, it analyses the input and invokes the User Interface module functions that handle the interaction with the user. On the other hand if a PUSH message is received from the MMS Client CGI Script, it invokes the MMS Client Transactions module and passes the message to it.

### 4.2.6 The MMS Client Transactions Stack

The MMS CTS comes into action in the following transactions:

- **Message Submission** : The MMS CTS module adds MMS headers to the composed message received from the MMS Client Driver and forms an MMS PDU. It then encodes the message as given in section 7 of [3] and then sends the message using the HTTP Client.
- **Message Retrieval** : When the user opts to fetch a new message, the MMS CTS fetches the message using the HTTP Client, decodes it, and then appends it to the message store.
- **Handling PUSH messages:** When a PUSH message arrives via the HTTP Server, the MMS CTS module parses and analyses it to find out the type of the PDU contained in the message. If the message is a Notification PDU, the relevant information is extracted from the PDU and appended to the message store. If the received PDU is a Delivery Report PDU, the status code of the messages stored in the sent-message store is updated as indicated in the Delivery Report. Also, a text message indicating the event is generated by the module which will be displayed to the user.

#### 4.2.7 The MIME Encoder

The MIME Encoder receives the content to be encoded from the MMS CTS, uses the BASE64 encoding scheme to encode the content into a MIME message and returns an encoded message back to the MMS CTS. Multiple multimedia files are encoded into a single 'multipart' MIME message as explained in [10].

#### 4.2.8 The HTTP Client

The HTTP Client is used by the MMS Client Driver during the Submission and Retrieval of messages. When submitting a new message, the HTTP Client is used to generate an HTTP POST request. The message is sent in the body of the POST request. The URI used in the POST request is the URI of the MPR with which the MMS Client is registered. When retrieving a message, the HTTP Client is used to generate an HTTP GET request to fetch the message. The URI used in this case is that which was indicated in the notification for the message.

### 4.3 Internal interfaces

#### 4.3.1 The HTTP Server-MMS Client CGI Script Interface

The HTTP Server invokes the MMS Client CGI Script and writes the MMS PDU received from the PPG on the standard input of the CGI Script. The MMS Client Script reads the PDU from its standard input and passes it to the MMS Client Driver. The CGI Script then reads the response received from the MMS Client Driver, generates a response to the PUSH submission as per section 7.4.2 of [6] and writes it on its standard output, which is read by the HTTP Server. This response is used by the HTTP Server to generate the response to the HTTP POST request.

#### 4.3.2 The MMS Client CGI Script-MMS Client Driver Interface

The communication channel between the MMS Client is a datagram based Unix Domain Socket. The MMS Client Script passes the PUSH messages containing MMS PDUs received from the HTTP Server to the MMS Client Driver. And the MMS Client Driver passes the response to the PUSH submissions back to the MMS Client CGI Script. The MMS PDUs passed using the PUSH framework are the M-Notification.ind PDU and the M-Delivery.ind PDU, the structure of which is as discussed in [3].

#### 4.3.3 The MMS Client Driver-User Interface

The MMS Client Driver interacts directly with the user through its standard input stream. Any key pressed by the user is read in by the Client Driver module and passed on to the user interface module for processing.

#### 4.3.4 The MMS Client Driver-UI Interface

The MMS Client Driver invokes the UI module functions and the communication between the two is through the function parameters. Based on the user keystrokes different functions of the UI module are called to carry out the requested activity. The different functions in the UI module are for Menu Display, Message Composition, Message Retrieval and PUSH message indication.

- **Menu Display** : The menu option chosen by the user is passed to the Menu Display function so that it can update the menu display if required.
- **Message Composition:** The MMS message composed by the user is passed back to the MMS Client Driver by the message composer function.
- **Message Retrieval:** The message-id of the message to be retrieved is passed back to the MMS Client Driver.
- **PUSH indication** : A text message explaining the PUSH message received is sent as a parameter to a PUSH indication display function.

#### 4.3.5 The MMS Client Driver-MMS CTS Interface

The MMS CTS module is invoked by the MMS Client Driver. The communication between the two is through function parameters. The interface comes into play in the following scenarios:

##### *4.3.5.1 Arrival of a PUSH message*

In this case the MMS Client Driver passes the MMS PDU received from the PUSH to the MMS CTS for classification and further action.

##### *4.3.5.2 New Message Submission*

When a new message is to be sent, the MIME encoded message is passed to the MMS CTS for encapsulation, encoding and forwarding.

##### *4.3.5.3 Message Retrieval*

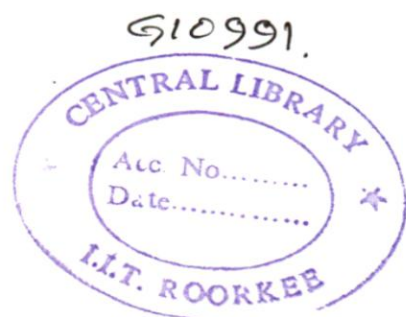
When the user has opted to fetch a new message, the message-id selected by the user is passed to the MMS CTS by the MMS Client Driver.

#### 4.3.6 The MMS CTS-MIME Encoder Interface

The MMS CTS invokes the MIME Encoder and passes the message body and the file names of the multimedia attachments to it as function parameters. The MIME Encoder returns a pointer to the encoded message back to the MMS CTS.

#### 4.3.7 The MMS CTS-HTTP Client Interface

The MMS CTS invokes the HTTP Client and passes the HTTP method name (GET or POST), MMS PDU and the URI of the MPR to the HTTP Client as function parameters. Any responses obtained from the HTTP Server are also passed back to the MMS CTS through parameters.





## 4.4 Architecture of the MMS Proxy Relay

### 4.4.1 Functional Decomposition

The figure 3-1 shows the functional decomposition of the MPR. The MPR consists of seven modules - Message Handler, SMTP Client, PUSH Client, Message Retrieval CGI Script, MPR CGI Script, HTTP Client and HTTP Server. A description of the functions of each of these systems is given below.

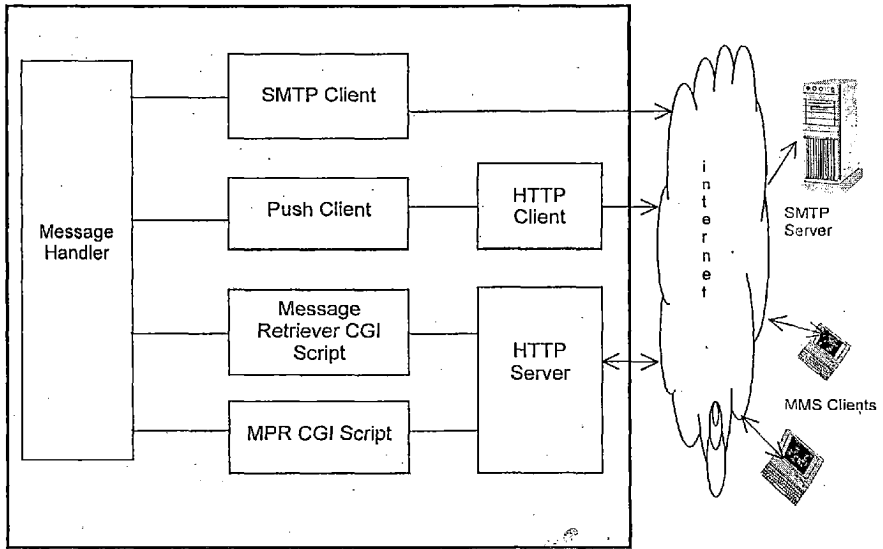


Figure 4-2 Functional Decomposition of the MMS Proxy Relay

### 4.4.2 The Message Handler

The Message Handler handles the storage, scheduling, delivery and delivery-reporting of MMS messages.

#### 4.4.2.1 Storage

The Message Handler receives the messages from the MPR CGI Script and stores them in a buffer. A Message-ID is assigned to each message that is stored. On receiving the message from the CGI Script it also generates and sends back a confirmation message in the form of an M-Send.conf PDU. Embedded in the PDU is a status code indicating the status of the MMS message and the Message-ID assigned to the message by the Message Handler.

At any time, there will be two types of messages stored with the Message Handler:

- a) Pending messages: These are the messages whose delivery time hasn't yet arrived.
- b) Notified messages: These are the messages whose delivery time has arrived and a notification for them has also been sent to the MMS client. These messages await delivery till the client fetches them.

Thus, a message-state indicator is also stored by the message handler corresponding to every message that it stores.

#### **4.4.2.2 Scheduling**

The Message Handler always stores the messages in a queue, sorted according to their Deliver-After-Time. A timer is run for the message having the most recent Deliver-After-Time. The expiry of the timer indicates that the Delivery time of one or more messages has arrived. All such messages are then removed from the queue and marked ready for delivery. The timer is then restarted for the next most recent message.

#### **4.4.2.3 Delivery**

When some message(s) are ready for delivery the Message Handler checks the receiver address field to see if the message is to be sent to an email Client or an MMS Client. The receiver address for an email client is an email address and that for an MMS Client is of type PLMN as is given in section 2.3 of [2]. For the former case, the message is decoded and converted into an RFC 2822 compliant email message and forwarded to the SMTP Client module for forwarding.

For messages meant for MMS Clients, a notification is first sent to the MMS Client through the Push Client. The MPR maintains a mapping of PLMN addresses to IPv4 addresses in a config file. The IP address of the client is thus fetched from the config file and passed to the PUSH Client along with the Notification PDU. This IP address will be used by the PUSH Client to PUSH the notification to the MMS Client.

The stored message is finally passed on to the client through the Message Retriever CGI Script whenever a fetch request for the same comes-in. The message-id allotted by the Message-Handler is used to match fetch request with the notification sent earlier.

#### **4.4.2.4 Delivery Report**

After a message is delivered to MMS Client(s), a delivery report is sent to the originating MMS Client. The Delivery.ind PDU is generated by the Message Handler and passed on to the PUSH Client along with the IP address of the MMS Client.

### **4.4.3 The HTTP Server**

HTTP is used as the transport for sending/receiving MMS PDUs. The HTTP server is used both for receiving the MMS messages from the MMS Clients and for delivering the MMS messages to the MMS Clients when they request it.

The HTTP server is responsible for accepting HTTP POST requests from MMS clients, invoking the MPR Script and passing the contents of the PDU to the MPR Script. It is

also responsible for sending the response (generated by the script) for the HTTP POST request.

The HTTP server also accepts HTTP GET requests from the MMS Clients which are requests for retrieval of MMS messages. In this case too, the HTTP Server invokes a CGI Script (the Message Retriever) which will retrieve and hand over the MMS Message to the HTTP Server that will then pass it on to the MMS Client.

Apache will be used as the HTTP Server for the development of the MMS System.

#### 4.4.4 The SMTP Client

The SMTP Client is required for sending mails to email clients. It accepts mails from the Message Handler and passes them on to the SMTP Server. The address of the host running the SMTP server is also passed to it by the Mail Handler. To establish an SMTP session with the SMTP server it extracts the requisite information (the sender address, the receiver address etc.) from the mail message.

#### 4.4.5 The Message Retriever Script

The Message Retriever Script is invoked by the HTTP server when the client requests the retrieval of an MMS message. The request carries with itself a Message-ID that will be used to fetch the MMS message. The script then uses this ID to fetch the message from the Message Handler and then passes the message so retrieved to the HTTP server to be sent back to the MMS Client.

#### 4.4.6 The MPR Script

The MPR Script is invoked by the HTTP server when an MMS Client uses the HTTP POST request to send an MMS message. The Script receives the MMS PDU containing the MMS message from the HTTP server and handles the MMS Message to the Message Handler. It then reads the response to this message from the Message Handler and passes this response to the HTTP server. This response is embedded in the body of the response to the HTTP POST and sent back to the MMS Client by the HTTP server.

#### 4.4.7 The PUSH Client

The PUSH Client is used in the following transactions:

- a) Notification : A notification message is sent to the MMS Client indicating to it that a message is available with the MPR and can be fetched. The M-Notification.ind PDU is generated by the message handler and passed to the PUSH Client along with the IP address of the MMS Client. The PUSH Client then uses the HTTP Client to send the notification PDU to the MMS Client using HTTP POST request. Port number 4035 on the MMS client is used for the connection.
- b) Delivery Report : The M-Delivery.ind PDU is generated by the Message Handler and passed to the PUSH Client with the IP address of the MMS Client. The message is

PUSHed to the MMS Client using the HTTP Client as was explained in the case of Notification Transaction above.

#### 4.4.8 The HTTP Client

The HTTP Client is used by the PUSH Client to send the Notification and Delivery-Report PDUs to MMS Clients. The PUSH Client uses the HTTP Client to generate an HTTP POST request to PUSH the PDUs to the MMS Clients.

The libwww library available from the WWW Consortium shall be used as the HTTP Client.

## 4.5 Internal Interfaces

### 4.5.1 Message Handler-SMTP Client Interface

The Message Handler passes the address of the host running the SMTP server and the MMS message as parameters to the SMTP Client. The structure of the message is as described in [3].

### 4.5.2 Message Handler-Push Client Interface

This interface comes into play in the following two transactions:

- a) **Notification:** The Message Handler passes the M-Notification.ind PDU and the IP address of the MMS Client as parameters to the Push Client. The structure of the M-Notification.ind PDU is as described in section 6.2 of [3].
- b) **Delivery Report:** The Message Handler passes the M-Delivery.ind PDU and the IP address of the MMS Client as parameters to the Push Client. The structure of the M-Delivery.ind PDU is as described in section 6.5 of [3].

### 4.5.3 Push Client-HTTP Client Interface

This interface comes into play whenever the PUSH Client PUSHes MMS PDUs to MMS Clients. The Push Client passes the PDU, the IP address of the client and the port number to connect to, to the HTTP Client as parameters.

### 4.5.4 Message Handler-Message Retriever Script Interface

This interface comes into action during the retrieval of messages. The information flow on this interface is through Unix Domain Sockets. The Message Retriever Script passes a Message-id to the Message Handler. The Message Handler in turn passes the headers and the body of the MMS message corresponding to that Message-id back to the Message Retriever Script.

The structure of the headers of this M-Retrieve.conf message is as given in section 6.3 of [3]. The message body follows the headers.

### 4.5.5 Message Handler-MPR Script

This interface comes into play when the MMS Client sends an MMS message. The information flow along this interface is through Unix Domain Sockets. The MPR Script passes the Message to the Message Handler along this interface. This message, called the M-Send.req message, consists of headers and a body. The structure of the headers of the M-Send.req message is as given in section 6.1.1 of [3]. The message body immediately follows the headers.

The Message Handler generates a response PDU on receiving the MMS message and passes it to the MPR Script. The structure of this M-Send.conf PDU is as given in section 6.1.2 of [3].

#### 4.5.6 Message Retriever Script-HTTP Server Interface

The HTTP server, when invoking the Message Servlet, passes the Message-id as a parameter to it. The Message Retriever Script writes the M-Retrieve.conf PDU (fetched from the Message Handler) on its standard output which is read-in by the HTTP Server and sent back to the MMS Client.

#### 4.5.7 MPR Script-HTTP Server Interface

The MPR Script receives the M-Send.req PDU on its standard input. It then passes this PDU to the Message Handler and receives an M-Send.conf PDU from it. The MPR Script then writes this response PDU to its standard output. This confirmation PDU forms the body of the response to the HTTP POST request that invoked the MPR Script.

## 5 Conclusion

---

The final system was tested for conformance with the functionalities included in the Protocol Implementation Conformation Specification (PICS) and was found to be conforming. All the transactions were tested and found to be working fine.

As already mentioned, the system aimed at implementing only a small set of functionalities to keep the design simple and the development less time consuming. Further improvements can be done to the system, by incorporating the features that were earlier left out for the sake of simplicity. The MMS Client in the current system does not include the functionality to render the multimedia content in the messages. It just identifies and isolates it. So, the client could be upgraded to provide rendering capabilities as well. The MMS Proxy Relay could be upgraded by adding Content-Conversion and Client Capability Negotiation capabilities, which would enable the relay to know the capabilities of the MMS Clients and transform the multimedia content into a format that can be better handled by the client.

## References

---

1. **Hypertext Transfer Protocol (HTTP/1.1) - RFC 2616**  
<http://www.ietf.org>
2. **Multipurpose Internet Mail Extensions: Format for Internet Message Bodies – RFC 2045**  
<http://www.ietf.org>
3. **Simple Mail Transfer Protocol - RFC 2821**  
<http://www.ietf.org>
4. **SMTP Message Format - RFC 2822**  
<http://www.ietf.org>
5. **WAP MMS Architecture Overview - WAP-205-MMSArchOverview-20010425-a, 2001**  
<http://www.wapforum.org>
6. **WAP MMS Client Transactions – WAP-206-MMSCTR-20010612-a, 2001**  
<http://www.wapforum.org>
7. **WAP MMS Encapsulation Protocol – WAP-209-MMSEncapsulation-20010601-a, 2001**  
<http://www.wapforum.org>
8. **WAP PUSH Architecture Overview – WAP-250-PushArchOverview-20010703-a, 2001**  
<http://www.wapforum.org>
9. **WAP Wireless Session Protocol – WAP-230-WSP-20010705-a**  
<http://www.wapforum.org>
10. **WAP User Agent Profile Specification – WAP-174-UAPProf-20010102, 2001**  
<http://www.wapforum.org>



# Appendix A: Push Architecture Overview

---

In the "normal" client/server model, a *client* requests a service or information from a *server*, which then responds in transmitting information to the client. This is known as "pull" technology: the client "pulls" information from the server. Browsing the World Wide Web is a typical example of pull technology, where a user enters a URL (the request) that is sent to a server, and the server answers by sending a Web page (the response) to the user.

In contrast to this, there is also "push" technology, which is also based on the client/server model, but where there is no explicit request from the client before the server transmits its content. The WAP Push framework introduces a means to transmit information to a device without a user request.

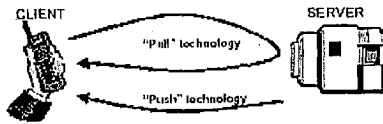


Figure 1

Another way to say this is that whereas "pull" transactions of information are always initiated from the client, "push" transactions are server-initiated.

## A.1 The Push Framework

A push operation in WAP is accomplished by allowing a *Push Initiator* (PI) to transmit *push content* and *delivery instructions* to a *Push Proxy Gateway* (PPG), which then delivers the push content to the WAP client (henceforth referred to as "client" or "terminal") according to the delivery instructions.

The PI is typically an application that runs on an ordinary web server. It communicates with the PPG using the *Push Access Protocol* (PAP). The PPG uses the *Push Over-The-Air* (OTA) *Protocol* to deliver the push content to the client.

Figure 2 illustrates the push framework:



Figure 2

PAP is based on standard Internet protocols; XML is used to express the delivery instructions, and the push content can be any MIME media type. These standards help make WAP Push flexible and extensible.

As mentioned, the PPG is responsible for delivering the push content to the client. In doing so it potentially may need to translate the client address provided by the PI into a format understood by the mobile network, transform the push content to adapt it to the client's capabilities, store the content if the client is currently unavailable, etc. The PPG does more than deliver messages. For example, it may notify the PI about the final outcome of a push submission and optionally handle cancellation, replace, or client capability requests from the PI.

The OTA protocol provides both connectionless and connection-oriented services. While the (mandatory) connectionless service relies upon Wireless Session Protocol (WSP), the (optional) connection-oriented service may be provided in conjunction with both WSP (OTA-WSP) and HTTP (OTA-HTTP). An important part of the OTA protocol is the Session-Initiation Application (SIA), which is further described in section 8.3.

Figure 2 illustrates the PI and the PPG as separate entities, which likely will be the most common configuration. It shall however be noted that the PI and the PPG may be co-located. The latter could, for example, be feasible for PPG operator services, large service providers, or when transport level end-to-end security is needed.

## A.2 The Push Proxy Gateway

The Push Proxy Gateway (PPG) is the entity that does most of the work in the Push framework. Its responsibilities include acting as an access point for content pushes from the Internet to the mobile network, and everything associated therewith (authentication, address resolution, etc).

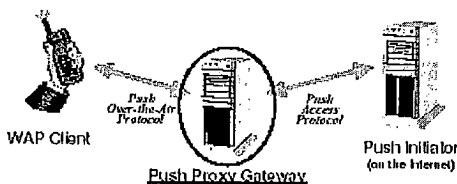


Figure 3

As the PPG is the entry point to a mobile network, it may implement network access-control policies about who is able to gain access to the network, i.e. who is able to push content and who is not, and under which circumstances, etc. It should be noted that both PPG (push) and WAP proxy [WAP] (pull) functionality may be built into a single proxy.

- For a more comprehensive discussion see WAP Push Architecture overview (WAP-250-archOverview-20010703-a).