

DEVELOPING A CROSS PLATFORM IMS CLIENT  
USING THE JAIN SIP APPLET PHONE

Submitted in fulfilment  
of the requirements of the degree of  
MASTER OF SCIENCE  
of Rhodes University

Walter Tawanda Muswera

*Grahamstown, South Africa*  
December 2014

## **Abstract**

Since the introduction of the IP Multimedia Subsystem (IMS) by the Third Generation Partnership Project (3GPP) in 2002, a lot of research has been conducted aimed at designing and implementing IMS capable clients and network elements.

Though considerable work has been done in the development of IMS clients, there is no single, free and open source IMS client that provides researchers with all the required functionality needed to test the applications they are developing. For example, several open and closed source SIP/IMS clients are used within the Rhodes University Convergence Research Group (RUCRG) to test applications under development, as a result of the fact that the various SIP/IMS clients support different subsets of SIP/IMS features.

The lack of a single client and the subsequent use of various clients comes with several problems. Researchers have to know how to deploy, configure, use and at times adapt the various clients to suit their needs. This can be very time consuming and, in fact, contradicts the IMS philosophy (the IMS was proposed to support rapid service creation).

This thesis outlines the development of a Java-based, IMS compliant client called RUCRG IMS client, that uses the JAIN SIP Applet Phone (JSAP) as its foundation. JSAP, which originally offered only basic voice calling and instant messaging (IM) capabilities, was modified to be IMS compliant and support video calls, IM and presence using XML Configuration Access Protocol (XCAP).

## **Acknowledgements**

First and foremost I offer my sincerest gratitude to my supervisor, Professor Alfredo Terzoli, who has supported me throughout my thesis with his patience and knowledge whilst granting me the space to grow and work in my own way. Without his encouragement and effort, this thesis would not have been completed or written. One simply could not wish for a better or friendlier supervisor.

Secondly, I would like to acknowledge the financial support of Telkom SA, Tellabs, Easttel, Genband, Bright Ideas 39 and THRIP through the Telkom Centre of Excellence in the Department of Computer Science at Rhodes University.

Finally, I thank my family, friends and the Rhodes University Convergence Research Group (RUCRG), who were undoubtedly the fulcrum that kept me from falling. Without you guys, the journey would have not been worthwhile! Your love, support and prayers have done so much for me that no words in this world can ever describe. I love you guys and I am truly blessed to have you in my life.

## Related Publications

The work that appears in this thesis has been presented in the following conference papers:

1. Walter Muswera, and Alfredo Terzoli. Development of an IMS Compliant, Cross Platform Client Using the JAIN SIP Applet Phone. In Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2010.
2. Walter Muswera, and Alfredo Terzoli. Developing a Cross Platform IMS Client using the JAIN SIP Applet Phone. In Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2011.
3. Walter Muswera, and Alfredo Terzoli. RUCRG IMS Client: Design and Implementation of Presence and XCAP. In Southern Africa Telecommunication Networks and Applications Conference (SATNAC), 2012.

# Abbreviations, Acronyms, and Terms

3GPP - 3rd Generation Partnership Project

AAA - Authentication, authorization and accounting

AH - Authorization header

AKA - Authentication and key agreement

AUTN - Authentication token

AVP - Audio video profile

CK - Cipher key

CSCF - Call session control function

HSS - Home subscriber server

HTTP - Hypertext Transfer Protocol

I-CSCF - Interrogating CSCF

IETF - Internet Engineering Task Force

IK - Integrity key

IM - Instant messaging

IMPI - IMS private user identity

IMPU - IMS public user identity

IMS - IP Multimedia Subsystem

IP - Internet Protocol

IPsec - Internet Protocol Security

J2SE - Java 2 Standard Edition

MD5 - Message Digest 5

P-CSCF - Proxy CSCF

PDA - Personal digital assistant

PRACK - Provisional acknowledgement

PSTN - Public switched telephone network

RAND - Random challenge

RTP - Real Time Transport Protocol

S-CSCF - Serving CSCF

SA - Security association

SDP - Session Description Protocol

SIP - Session Initiation Protocol

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UMTS - Universal Mobile Telecommunications System

URI - Uniform resource identifier

URL - Uniform resource locator

VoIP - Voice over Internet Protocol

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Background . . . . .	1
1.2	Problem Statement . . . . .	2
1.3	Objectives of the Research . . . . .	3
1.4	Scope . . . . .	3
1.5	Thesis Organisation . . . . .	4
<b>2</b>	<b>Protocols and Technologies</b>	<b>5</b>
2.1	Session Initiation Protocol . . . . .	5
2.1.1	SIP Background and History . . . . .	6
2.1.2	SIP Protocol Operation . . . . .	6
2.1.2.1	SIP Requests . . . . .	6
2.1.2.2	SIP Responses . . . . .	7
2.1.3	SIP Functions . . . . .	8
2.1.4	SIP Entities . . . . .	8
2.1.4.1	User Agents . . . . .	9
2.1.4.2	Registrars . . . . .	9
2.1.4.3	Proxy Servers . . . . .	9
2.1.5	Types of Services Enabled by SIP . . . . .	10
2.1.5.1	Basic Session Management Services . . . . .	10
2.1.5.2	Presence . . . . .	11
2.2	SIP and Other Protocols . . . . .	12

---

2.2.1	Session Description . . . . .	13
2.2.2	The Media Plane . . . . .	16
2.3	The IP Multimedia Subsystem . . . . .	18
2.3.1	IMS Overview . . . . .	18
2.3.2	IMS Architecture . . . . .	18
2.3.3	Elements of the IMS Architecture . . . . .	21
2.3.3.1	IMS Terminals . . . . .	22
2.3.3.2	SIP Servers . . . . .	22
2.3.3.3	User Databases . . . . .	25
2.3.4	UE Procedures . . . . .	25
2.3.4.1	Local P-CSCF Discovery . . . . .	26
2.3.4.2	Application Level SIP Registration . . . . .	26
2.3.4.3	Service Route Discovery During Registration . . . . .	27
2.3.5	IMS Concepts . . . . .	27
2.3.5.1	IMS Identities . . . . .	27
2.3.5.2	IMS Security . . . . .	28
2.3.6	Private SIP Extensions for 3GPP IMS . . . . .	29
2.3.6.1	IMS Services . . . . .	30
2.4	Summary . . . . .	30
<b>3</b>	<b>Existing IMS Clients and JSAP</b>	<b>31</b>
3.1	IMS Clients . . . . .	31
3.1.1	IMS Communicator . . . . .	32
3.1.2	UCT IMS Client . . . . .	33
3.1.3	Mercurio IMS Client . . . . .	33
3.2	JAIN SIP Applet Phone (JSAP) . . . . .	33
3.2.1	JSAP Architecture . . . . .	34
3.2.2	Limitations . . . . .	36
3.3	Summary . . . . .	37



---

<b>4</b>	<b>Development Tools</b>	<b>38</b>
4.1	Application Programming Interfaces (APIs)	38
4.1.1	Media APIs	39
4.1.1.1	FMJ	40
4.1.1.2	FFMPEG	40
4.1.1.3	JFFMPEG	41
4.1.1.4	FFMPEG-Java	41
4.1.1.5	LTI-CIVIL	41
4.1.1.6	Gstreamer	42
4.1.1.7	Gstreamer-Java	42
4.1.2	Signalling APIs	43
4.1.2.1	JAIN SIP API	43
4.1.2.2	JAIN SDP API	45
4.1.3	Mobicents XCAP API	45
4.2	Other Tools Used for Developing the Client	45
4.2.1	JDK 1.6	45
4.2.2	Netbeans 6.8	46
4.3	Summary	46
<b>5</b>	<b>Enhancing Signalling and Media Support in JSAP</b>	<b>47</b>
5.1	Preliminary Analysis	47
5.1.1	Unhandled Server Responses	48
5.1.2	Password re-prompting	48
5.1.3	Cancelling Early Sessions	49
5.1.4	Re-registration	51
5.1.5	De-registration	51
5.2	Enhancements	52
5.2.1	Configuration	52
5.2.2	Reliability of Provisional Responses	55

---

5.2.2.1	UAC Behaviour . . . . .	55
5.2.2.2	UAS Behaviour . . . . .	56
5.2.3	Session Description Handling . . . . .	59
5.2.3.1	Dynamic Payload Coding and Decoding . . . . .	59
5.2.3.2	Sending the SDP Offer with Preferred Codec . . . . .	61
5.2.3.3	Receiving the SDP Offer and Selecting Preferred Codec . . . . .	62
5.2.3.4	Sending the SDP Answer . . . . .	63
5.2.3.5	Receiving the SDP Answer . . . . .	63
5.2.4	Media Plane . . . . .	64
5.2.4.1	Gstreamer Concepts . . . . .	65
5.2.4.2	Implementation of the Media Plane . . . . .	66
5.3	Summary . . . . .	69
<b>6</b>	<b>Improving Presence Support</b>	<b>70</b>
6.1	Presence . . . . .	70
6.2	JSAP+ Presence . . . . .	71
6.3	Network Storage of User Information . . . . .	72
6.3.1	Choice of Technology . . . . .	73
6.3.2	Integration of XCAP support . . . . .	73
6.3.2.1	Authentication and Authorisation . . . . .	73
6.3.2.2	Client Operations . . . . .	74
6.3.3	Presence Lists . . . . .	77
6.4	JSAP++ Architecture . . . . .	79
6.5	Summary . . . . .	81
<b>7</b>	<b>Adding IMS Compliance</b>	<b>82</b>
7.1	Development Process . . . . .	82
7.2	IMS Registration . . . . .	83
7.3	IMS Session Establishment . . . . .	91

---

7.3.1	UAC Behaviour . . . . .	93
7.3.2	UAS Behaviour . . . . .	95
7.4	SDP Codec Negotiation in IMS . . . . .	99
7.5	Session Cancellation . . . . .	103
7.6	Session Ending . . . . .	104
7.7	Putting Things Together . . . . .	105
7.8	Summary . . . . .	108
<b>8</b>	<b>RUCRG IMS Client Testing</b>	<b>109</b>
8.1	Testing . . . . .	109
8.1.1	Conformance Testing . . . . .	109
8.1.2	Interoperability Testing . . . . .	110
8.1.3	Interoperability with Conformance Monitoring . . . . .	110
8.2	Testing Requirements . . . . .	111
8.2.1	Hardware Requirements . . . . .	111
8.2.2	Software Requirements . . . . .	111
8.2.3	Testbed Specifications . . . . .	112
8.3	Test Setup . . . . .	113
8.4	System Testing . . . . .	114
8.4.1	Endpoint Registration with a Registrar . . . . .	115
8.4.1.1	Entities Involved . . . . .	115
8.4.1.2	Test Purpose . . . . .	115
8.4.1.3	Preconditions . . . . .	115
8.4.1.4	Results . . . . .	115
8.4.2	Point-to-point Audio/Visual call using Proxy/IMS . . . . .	120
8.4.2.1	Entities Involved . . . . .	120
8.4.2.2	Test Purpose . . . . .	120
8.4.2.3	Preconditions . . . . .	120
8.4.2.4	Results . . . . .	120

---

8.4.3	Presence . . . . .	128
8.4.3.1	Entities Involved . . . . .	128
8.4.3.2	Test Purpose . . . . .	128
8.4.3.3	Preconditions . . . . .	128
8.4.3.4	Results . . . . .	129
8.5	Summary . . . . .	131
<b>9</b>	<b>Conclusion</b>	<b>132</b>
9.1	Synopsis . . . . .	132
9.2	Discussion . . . . .	135
9.2.1	Achieved Goals . . . . .	135
9.2.2	Challenges . . . . .	136
9.2.3	Limitations . . . . .	136
9.3	Future Work . . . . .	136
9.4	Summary . . . . .	137
	<b>Appendix A Accompanying CD-ROM</b>	<b>144</b>
	<b>Appendix B Deployment Guide</b>	<b>145</b>

# List of Figures

2.1	SIP User Agent	9
2.2	Presence Operation	12
2.3	IMS Functional Planes (Adapted from Oguejiofor <i>et al</i> [43])	20
2.4	<P, S and I> - CSCF working together	25
3.1	JSAP Derived Architecture	35
5.1	Testing Information Flow (Adapted from Luo [36])	48
5.2	Dialog Creation	50
5.3	Client Configuration Life Cycle	54
5.4	Session setup Using Reliability Mechanisms	58
5.5	JSAP Gstreamer Video Pipelines	67
6.1	JSAP+ Presence Handling	71
6.2	RUCRG XCAP Client Interface	74
6.3	Presence implementation using RLS with XCAP	78
6.4	JSAP++ Presence implementation with XCAP	79
6.5	JSAP++ Architecture	80
7.1	Authentication Screen	86
7.2	IMS Registration (Non-roaming Case)	88
7.3	SIP Registration	89
7.4	De-registration menu	90

---

7.5	Ready for calls . . . . .	92
7.6	Invite Screen . . . . .	93
7.7	Incoming Call Screen . . . . .	96
7.8	Session setup Call Flows . . . . .	98
7.9	Points when CANCEL is enabled for RUCRG IMS client . . . . .	103
7.10	RUCRG IMS client CANCEL sequence diagram . . . . .	104
7.11	RUCRG IMS client BYE sequence diagram . . . . .	105
7.12	IMS Client Display Before Registration . . . . .	106
7.13	IMS UAC Main Flow Diagram . . . . .	107
8.1	RUCRG IMS Client Position in RUCRG Testbed . . . . .	113
8.2	Interoperability Testing with Conformance Monitoring (Adapted from ETSI [17]) . . . . .	114
8.3	Test Arrangement for SIP Registration . . . . .	116
8.4	Test Arrangement for IMS Registration . . . . .	118
8.5	Test Arrangement for SIP Session Setup . . . . .	120
8.6	Test Arrangement for IMS Session Setup . . . . .	121
8.7	Test Arrangement for SIP Media Test Case . . . . .	123
8.8	Test Arrangement for IMS Media Test Case . . . . .	123
8.9	Test Arrangement for Presence Test Case . . . . .	128

# Chapter 1

## Introduction

Research groups such as the RUCRG (Rhodes University Convergence Research Group) involved in the development of SIP/IMS applications require an IMS compliant client for application testing. However, there is a lack of a single, free and open source IMS compliant client that provides researchers with all the functionality needed to test SIP/IMS applications. This chapter sets the scene to explain what has been done in this research to overcome this problem. This chapter also outlines in brief the background and objectives of this research.

### 1.1 Background

In recent years, there has been a growing interest in multimedia communication services offered via the Internet and other telecommunications platforms. Internet users who used to merely surf the web or send emails are now using services such as instant messaging (IM), presence, on-line gaming as well as voice and video over IP (VVoIP). This growth (Internet usage), has been driven by the capability of the Internet to provide several new services seamlessly to users at any time. Furthermore, the growth can be attributed to the availability of protocols and standard APIs that are openly available to service developers [11].

With Internet users accessing services from a range of end user terminals (with varying capabilities), the challenge of integrating voice and data services in fixed and mobile environments has become more complex. Service integration has thus become an important aspect when building IP multimedia communication services.

The IP Multimedia Subsystem (IMS) is a service delivery framework specified by the 3rd Generation Partnership Project (3GPP) and other standards development organisations

[41]. It defines a unifying architecture for IP based services over both packet switched (PS) and circuit switched (CS) networks. The IMS enables the convergence of different wireless and fixed access technologies for the creation, delivery and consumption of multimedia services [41, 43]. This means that the IMS enables users to access services from a range of end user terminals. Additionally, the IMS supports service integration through standardised reference points (interfaces and protocols) which not only makes service creation faster and easier but also makes Internet technologies such as Web services available to end user devices with varying technologies. IMS can therefore be viewed as a catalyst for convergence, a platform through which new communication applications are delivered as well as an enabler for service driven development.

## 1.2 Problem Statement

The RUCRG in the Department of Computer Science is mainly concerned with current trends in the move towards converged service platforms for next generation networks (NGNs) and the Internet. Research in these areas covers service orchestration, policy frameworks for service development, development of tool-kits for services such as IP television (IPTV), Location Based Services (LBS) and Video on Demand (VoD) using open standards. These applications are built on platforms such as the Mobicents Application Server, FOKUS IMS Core, Kamailio and Asterisk. As the interest in the IMS grows, applications being developed within the RUCRG use IMS as a deployment platform because it enables the deployment and/or delivery of integrated services using open standards [11]. Several free, open and closed source SIP/IMS clients are currently being used by the RUCRG developers for application testing. The use of various clients comes with several challenges. The major challenges being that application developers have to learn to install, configure, use and extend the various clients to suit their needs. Additionally, most of the available clients support only a subset of the functions that are required, which poses further challenges during testing. For example, some clients:

- Can only be used on specific platforms.
- Support a limited range of video and audio codecs.
- Do not support network storage of user data (such as resource lists).
- Cannot be extended because they are closed source and proprietary.
- Are difficult to debug, test and extend due to the structure of their code.



- Have existing bugs that have not been fixed in a long time or have been discontinued.

Among these clients is the JSAP [29] which one of RUCRG members helped to develop. Researchers in the RUCRG used the JSAP extensively to test SIP applications and have a deep understanding of the client code. Unfortunately, the client only supports SIP applications. The RUCRG decided to upgrade the JSAP to be IMS compliant and create a single client that researchers (RUCRG) can easily adapt to suit their needs as they develop new services.

### 1.3 Objectives of the Research

The main objective of this thesis was to upgrade the JSAP to be IMS compliant so that it can be used for both SIP and IMS application testing. The goal was to produce a client that provides native IMS functionality, supports re-usability of client code, enables service composition/aggregation, and allows easy modification by the RUCRG researchers. These goals were identified by working closely with the RUCRG researchers, gathering a comprehensive list of requirements and incrementally adding functionality to the client. This client was developed to be compliant with 3GPP, European Telecommunications Standards Institute (ETSI), Telecoms and Internet converged Services and Protocols for Advanced Network (TISPAN) and the Internet Engineering Task Force (IETF) recommendations and specifications. Care was taken not to limit the use of the client to the IMS platform thus making the client backward compatible with legacy SIP servers and applications. Lastly, the client had to be free in terms of cost (use of freely available libraries in development) as well as open source.

### 1.4 Scope

The study focuses on upgrading the JSAP to be IMS compliant but not to develop particular services.

This study does not take into account the use of the IP multimedia Services Identity Module (ISIM) as in the case of mobile IMS clients because this PC based IMS client does not use ISIM modules.

The client will support audio and video using the following codecs only: PCMU, GSM, G722, G723, DVI4\_8000, DVI4\_16000, PCMA, G728, G729, JPEG, H261, H263 and H263+.

re-INVITE will not be implemented so there is no session management, that is, renegotiation of session parameters. This feature will be added in future.

Finally, there is no support for the encryption of communication between the clients and the IMS network. Clients use the initial IMS authentication to establish a trusted connection.

## 1.5 Thesis Organisation

The remainder of this thesis is organised into eight chapters:

Chapter 2 - Protocols and Technologies: Provides an overview of literature related to SIP and the IMS. The chapter shows the relationship between SIP and the IMS, specifically highlighting how SIP plays a major role in the IMS.

Chapter 3 - Existing IMS Clients and JSAP: Assesses some of the existing IMS clients that are used in the RUCRG to show their limitations and put into perspective why a new client was needed. The chapter also provides an overview of the JSAP and its architecture, focusing mainly on the work that was carried out to understand its structure.

Chapter 4 - Development Tools: The chapter discusses the software and tools used for developing the client.

Chapter 5 - Enhancing Signalling and Media Support in JSAP: Discusses the process of incorporating advanced SIP request and response messages into JSAP in preparation for IMS support. The chapter also discusses how Gstreamer was integrated into the client to handle media.

Chapter 6 - Improving Presence Support: Discusses how XCAP support was incorporated into the JSAP client to support presence and buddy list uploads/downloads.

Chapter 7 - Adding IMS Compliance: Discusses how IMS support was added to the client (registration and session setup) to work seamlessly with the existing SIP implementation.

Chapter 8 - RUCRG IMS Client Testing: Presents how testing was performed to validate the new IMS capable client. The chapter also details the experimental setup (what was being tested and how it was done) and the results of the tests that were carried out.

Chapter 9 - Conclusion: Assesses whether the client meets the objectives that were set out for the project, and proposes possible extensions that can be made to the client to provide richer IMS services.

# Chapter 2

## Protocols and Technologies

This chapter will provide an overview of the key protocols and technologies underpinning the IMS client developed in this thesis. Specifically, the discussion will focus on the protocols that were used in the development of the JSAP such as Session Initiation Protocol (SIP), Session Description Protocol (SDP) and Real-time Transport Protocol (RTP). Furthermore, an explanation on why SIP plays such a crucial role in the Internet communications space will be provided. This will lead us to examine the IP Multimedia Subsystem (IMS) and its features.

### 2.1 Session Initiation Protocol

A large number of applications that are developed within the Rhodes University Convergence Research Group (RUCRG) are examples of multimedia communication services delivered over the Internet. These applications depend on establishment and termination of sessions between servers and clients. Most, if not all, of these sessions are established using SIP.

SIP is defined by the Internet Engineering Task Force (IETF) in RFC 3261 [54] as an application layer signalling protocol for initiating, modifying, or terminating communication and collaborative sessions over Internet Protocol (IP) networks [11]. SIP facilitates communication between different users by providing the means for endpoints (clients) to discover one another and to negotiate variables for the session they would like to share. In other words, SIP helps to find the best way for users to communicate given their preferences and the capabilities of the devices they have at their disposal.

### 2.1.1 SIP Background and History

SIP originated in late 1996 as a component of the IETF multicast backbone (Mbone - an experimental multicast network on top of the public Internet). SIP was adopted as an IETF proposed standard in 1999 and published under RFC 2543 [27]. In 2002, SIP was published under RFC 3261 [54] after being enhanced with new features and a better design to incorporate interoperability functions. RFC 3261 is currently the core SIP specification as defined by IETF and is backward compatible with RFC 2543. SIP has become widely used for VVoIP services. As will be shown later in this chapter, SIP is at the heart of the IMS network architecture [39].

### 2.1.2 SIP Protocol Operation

SIP is based on the Web protocol Hypertext Transfer Protocol (HTTP) and like HTTP, SIP has a client/server architecture. SIP is therefore a request/response protocol. Requests are generated by SIP clients while SIP servers receive requests and return responses. Before we could upgrade the JSAP, there was need to verify that the already implemented SIP functions worked according to the SIP specifications. The following sections discuss how the various SIP functions work.

#### 2.1.2.1 SIP Requests

The core IETF SIP specification defines six types of SIP requests, each with a different purpose. Every SIP request contains a field called a method, which denotes its purpose. Table 2.1 shows the methods defined by the IETF for each of the core SIP requests.

Table 2.1: RFC 3261 SIP Request Methods

Method	Description
ACK	Confirms that the client has received a final response to a request.
BYE	Used by a client to tell the server that it wishes to release the call.
CANCEL	Cancels a pending request with the same <i>Call-ID</i> , <i>To</i> , <i>From</i> and Call sequence number ( <i>Cseq</i> ) header field values.
INVITE	Indicates that the user or service is being invited to participate in a session. The message body may contain a description of the session.
OPTIONS	Queries the capabilities of the other side.
REGISTER	Used by a client to register the address listed in the <i>To</i> header field with a SIP server.

In addition to the original six SIP requests, other request methods are defined by the IETF as the extensions to the RFC 3261 base SIP specification. These methods are shown in Table 2.2.

Table 2.2: Other SIP Request Methods

Method	RFC	Description
INFO	2976	Transfers information during a session.
MESSAGE	3428	Allows the transfer of IM.
NOTIFY	3265	Informs the user about the subscribed event.
SUBSCRIBE	3265	Enables the user to subscribe to certain events.
REFER	3515 and 4488	Enables the sender of the request to instruct the receiver to contact a third party.
PRACK	3262	Provisional Reliable ACK (PRACK) plays the same role as ACK request, but for provisional responses.
UPDATE	3311	Allows a client to update parameters of a session, but has no impact on the state of a Dialog.
PUBLISH	3903	PUBLISH is similar to REGISTER in that it allows a user to create, modify and remove state to another entity which manages this state on behalf of the user.

### 2.1.2.2 SIP Responses

According to RFC 3261 [54], every request must have at least one final response, and may also have a number of provisional responses. Responses include a three digit (numeric) status code and a reason phrase. The latter contains human readable information about the status code. There are two types of SIP responses: provisional/informational and final response. These responses are grouped into status codes, of which there are six values for the first digit as shown in Table 2.3

Table 2.3: RFC 3261 SIP Responses

Response	Description
1xx	Provisional or information responses. They indicate that the request has been received and the recipient is processing the request.
2xx	Success response.
3xx	Redirection responses. The requester needs to take further action to complete the request.
4xx	Client-error responses
5xx	Server-error responses
6xx	Global-failure responses. The request cannot be fulfilled at any server.

### 2.1.3 SIP Functions

Signalling plays a key role in IP multimedia communication services and, as has been earlier pointed out, SIP is typically used as the signalling protocol. SIP basically solves two key aspects in IP multimedia communications:

#### 1. Session setup, modification, and termination

One of the main functions of SIP is the initiation of multimedia sessions. By using SIP, a local user can signal his/her desire to engage in a multimedia session with a remote user. Similarly, the remote user can use SIP to signal his/her acceptance or rejection of the communication [45, 10]. During the session setup, session descriptors are exchanged so that both parties can agree on the crucial parameters for the session. SIP can also be used to modify session parameters of an ongoing session, for instance, if a user is engaged in an audio session and wants to add video to the session. A re-INVITE request is sent in order to add the new media components to the session.

The last SIP function related to session management is session termination. Any of the session participants can use SIP to signal his/her desire to terminate the communication while effectively stopping media transmission and reception [45, 10].

#### 2. Location of users

SIP makes use of elements called proxy servers to help route requests to a user's current location. Proxy servers obtain the user's location when user agents send registrations [54]. SIP clients therefore need to register with a proxy server as this allows their location to be known (this is important for receiving incoming requests). The location is identified by an IP address and a port number. This means that in SIP, registration is used for routing incoming SIP requests but has no role in authorising outgoing requests. Authorisation and authentication are handled in SIP on a request-by-request basis via a challenge/response mechanism [54].

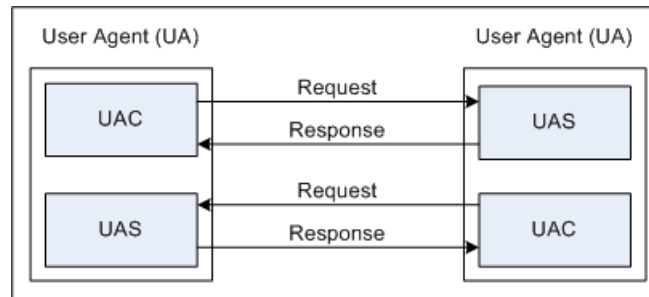
### 2.1.4 SIP Entities

The SIP protocol defines a number of SIP entities as part of the SIP architecture:

### 2.1.4.1 User Agents

A SIP UA comprises two components, a user agent client (UAC) and a user agent server (UAS) as shown in Figure 2.1:

Figure 2.1: SIP User Agent



The UAC is responsible for the generation of new SIP requests and the reception of the associated responses. The UAS is responsible for receiving SIP requests and generating the appropriate responses. UAs are typically located at the SIP endpoints, and the end user can interact with them through a user interface. UAs are the main focus of this research.

### 2.1.4.2 Registrars

As earlier pointed out, a SIP UA needs to be registered before it can receive multimedia calls. Registration is a process by which a SIP UA communicates its current location and its externally visible identifier (formally known as the SIP Address of Record) to the registrar server. A registrar is a server that accepts registration requests from the UAs. It authenticates and registers users when they come on-line, and then stores information on the users' logical identities and the devices that they can use for communications. The devices are identified by their URIs [23]. When the registrar accepts the registration request, it places the received information (the mapping between user location and globally visible identifier) in a database called a Location Service.

### 2.1.4.3 Proxy Servers

A proxy server is an intermediary entity that makes requests on behalf of other clients. It primarily helps with SIP routing, which means that its main purpose is to ensure that a request is sent to another entity "closer" to the targeted user [45]. Basically, a proxy server takes SIP requests, processes them, and passes them downstream while sending responses

upstream to other SIP servers or devices. A proxy is involved only in the setup and tear-down of a communication session. After a UA establishes a session, communication occurs directly between the parties involved [23] unless otherwise required.

Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets and, if necessary, rewrites specific parts of a request message before forwarding it.

There may be a set of proxies between UAC and UAS that help to route requests. Two specific types of SIP proxies will be discussed.

- **Outbound Proxy**

An outbound proxy helps the UAs to route outgoing requests. UAs are usually configured to route all their requests to an outbound proxy, which will route the requests for them.

- **Inbound Proxy**

An inbound proxy is a proxy server that handles incoming requests for an administrative domain. It basically helps to route incoming requests to the appropriate UA within the domain it is responsible for. When an inbound proxy receives a request for a user belonging to the domain for which that proxy is responsible, the proxy queries the Location Service, determines the contact address of the UA to which this request is directed, and forwards the request to that address.

## 2.1.5 Types of Services Enabled by SIP

In this section, we will look more closely into some of the different types of services that can be enabled by SIP. We will specifically examine those services that are implemented in the JSAP so that we can verify that they are working properly. Furthermore, this will allow us to reuse some of these functions when we upgrade the client to be IMS compliant.

### 2.1.5.1 Basic Session Management Services

As already alluded to, SIP plays a crucial role in providing the main control functions needed in IP multimedia communication scenarios. SIP can be used to enable communications based on a variety of media, such as: voice communication, video communication, IM communication, text over IP, peer to peer gaming, white-boarding and file transfer to name but a few. Additionally, SIP provides support for combining different types of media in the same communication session. There are several possible combinations that



one can imagine but the most common are: voice combined with video (so called video telephony), voice combined with IM, voice combined with real time text, voice combined with on-line transfer of a picture, voice combined with the on-line transfer of a generic file, voice combined with gaming and voice combined with white-boarding.

As it may be deduced from above, the main media component is voice, with an additional media added to it. These particular scenarios are sometimes referred to as “rich voice” [45].

### 2.1.5.2 Presence

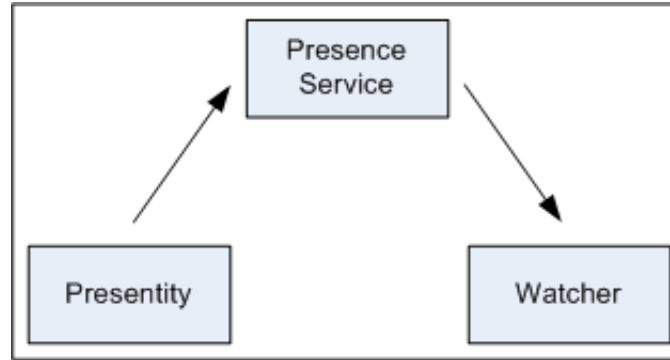
In order to improve communication among users, it is useful for them to see the presence of their “buddies”. Presence is a standard method of representing and querying the status of an individual, both physical (e.g., a user’s location) and on-line (e.g., status of avatars) [8]. SIP offers the tools for publishing, subscribing, and notifying watchers about availability and willingness of users to set up multimedia communications. To access presence information, users often refer to a presence server. Presence servers accept, store, and distribute presence information.

The Instant Message and Presence Protocol (IMPP) Working Group of the IETF, define an abstract model for describing IM and presence systems in RFC 2778 [13] . The model defines three different entities:

- The presentity is the entity that provides presence information. For instance, Chiedza may want to provide her presence information (on-line, busy, and so on) to her buddies. The presentity is an abstract concept that represents Chiedza for the presence service.
- The watcher is the entity that receives presence information. For instance Tino, a buddy of Chiedza’s, might be interested in “watching” her presence information. There are two types of watchers, namely:
  - Subscribers request notification of future changes in the presentity’s presence information from the presence service.
  - Fetchers do not subscribe to a presentity’s presence information but simply request the current value of the presentity’s presence information. A special type of fetcher that requests information on a regular basis is called a poller.
- The presence service receives presence information from the presentities and distributes it among the watchers.

This is represented in Figure 2.2:

Figure 2.2: Presence Operation



In addition to this basic model, the presence service is typically related to other services that are responsible for managing lists of groups of users (buddy lists). Presence information can typically be shared only within these groups of users. These capabilities allow the development of community based services [45].

## 2.2 SIP and Other Protocols

Various services can be offered on top of the Internet, that is, on top of an IP network [45]. Among these are streaming services (which allow users to access, in real-time, either live or stored time-based media content) such as Video-on-Demand (VoD) and Internet Protocol Television (IPTV) and communication services (those that allow people to communicate with each other using different types of media) such as Voice over IP (VoIP) and email exchange [45].

As pointed out earlier in section 2.1, SIP plays a crucial role in the delivery of multimedia communication services over the Internet. However, SIP by itself, is not capable of delivering multimedia communication services. It needs to work alongside other protocols to accomplish that function. Most importantly, because SIP is a signalling protocol, it needs to work together with other protocols at the media layer [10]. In this section, we will explain what multimedia communications are, the role of signalling and media protocols in IP multimedia communications.

In order to bring to light the role of signalling and media protocols in IP multimedia communications, we will look at what is required to set up the exchange of multimedia data between two communicating parties. Let us assume that Chiedza and Tino want to have a voice conversation.

1. First of all, we need a mechanism by which Chiedza can first signal to Tino her desire to start conversing with him. This would be like an invitation signal sent from Chiedza's PC to Tino's.
2. Secondly, when this signal reaches Tino's PC, it would need to trigger some alerting mechanism that can attract Tino's attention.
3. It may take some time for Tino to respond so in the meantime we have to inform Chiedza about the progress of the communication attempt. For instance, Chiedza may need to know that her invitation went through and that Tino is being alerted [45].
4. The fourth aspect refers to the fact that in order to send voice samples over the network, they first need to be encoded. Likewise, the encoded data needs to be decoded at the receiving end. There are a variety of standard ways to code and decode the voice signals, and it is crucial that the CODEC (COder/DECoder) used in Chiedza's PC matches the one used by Tino. It is therefore necessary that, prior to starting the voice communication, Chiedza and Tino agree on the codecs that they will use for this particular communication.
5. Finally, Chiedza needs to add Tino's computer IP address as the destination address in the IP packets that she sends to Tino.

### 2.2.1 Session Description

In this section we will look at how multimedia sessions can be described. We will focus on the SDP protocol which defines the syntax for describing multimedia sessions. This is because JSAP uses the SDP protocol to describe multimedia sessions and there is need to verify that these functions were correctly implemented before we upgrade the client.

Most, if not all, of the aspects illustrated above highlight a need to exchange some extra information between Chiedza and Tino. This is not the actual voice information (media), but rather, information that helps Chiedza and Tino to control the way voice communication occurs. This control information is sent in messages between Chiedza's and Tino's computers according to some signalling protocol. SIP is one such signalling protocol that can convey this type of information, but there are others. The relevance of signalling in this context is important, not just to cope with the basic call scenarios, but also to enable more complex multimedia value added services.

SIP is used to control multimedia communications irrespective of the session being established. This works perfectly well because SIP does not need to care about the nature of the session in order to deliver its functions [45]. However, there is still a need at specific times, such as session creation, to describe the characteristics of the session and convey that information to the participants of the session. Such descriptions, which are actually dependent on the nature of the session would include parameters such as media types, transport addresses, start time and duration of the session, and so on. This knowledge is crucial for the participants in the session. For example, in a two-party voice call; before the actual voice transmission can start, the participants need to learn what IP addresses and ports they need to send the media packets to. Moreover, they also need to agree on what voice codec to use for transmission and reception. This is done using SDP. SIP messages carry SDP session descriptions that allow participants to agree on a set of parameters needed for the multimedia communication. SIP does not need to know about the session specifics.

SDP specified in RFC 4566 [26] defines a general-purpose format for describing multimedia sessions. SDP defines a language for representing the key parameters that characterise a multimedia session. SDP is text based. An SDP message contains three levels of information:

1. Session level description: contains lines that describe characteristics of the whole session.
2. Time description: contains lines indicating time-related aspects of the session.
3. Media description: contains lines that characterise the different media present in the session [45].

Tables 2.4, 2.5 and 2.6 taken from RFC 4566 [26] show the different types of lines for each level indicating whether the field is required (R) or optional (O).

Table 2.4: Session Level Description SDP Lines

Field	Field description	R/O
v	Protocol version	R
o	Originator and session identifier	R
s	Session name	R
i	Session information	O
u	URI of description	O
e	Email address	O
p	Phone number	O
c	Connection information	O
b	Bandwidth information	O
z	Time zone adjustments	O
k	Encryption key	O
a	Session attribute	O

Table 2.5: Time level Description SDP Lines

Field	Field Description	R/O
t	Time the session is active	R
r	Repeat time	O

Table 2.6: Media Level Description SDP Lines

Field	Field Description	R/O
m	Media name and transport address	R
i	Media title	R
c	Connection information	R
b	Bandwidth information	O
k	Encryption key	O
a	Attribute line	O

### The Offer/Answer Model

The use of SDP in communication requires defining a negotiation framework so that the communicating parties can agree on the session characteristics. Such a negotiation framework is called the offer/answer model, and is defined in RFC 3264 [52]. A party that wants to communicate indicates the desired session description from his/her point of view. This is called the SDP offer. The offer contains, among other things:

- The set of media streams that the offerer wants to use.

- The desired characteristics of the media streams as qualified by the format parameter and the media-line attributes.
- The IP addresses and ports which the offerer wants to use to receive the media.
- Additional parameters, if needed, that further qualify the media transport.

When the remote party receives the offer, it replies with an SDP answer. The answer contains the following pieces of information:

- Whether a media stream is accepted or not.
- The media streams characteristics that will be used for the session.
- The IP addresses and ports that the answerer wants to use in order to receive media.

The offerer receives the answer, and, at this point, if the answerer has accepted at least one media stream, both parties have found an overlap in their respective desired session descriptions, and communication can start. In the case of media types that are conveyed using RTP, the offer/answer model enables the negotiation of the type of codecs [45]. The set of functions and elements that participate in the processing and exchange of the signalling are said to constitute the Control (or signalling) Plane.

### 2.2.2 The Media Plane

Not all the services are delivered through manipulation of the signalling [45]. The simplest multimedia call requires some media level handling at the endpoints, in order to capture and present the media as well as to receive and transmit. Thus, there is a need for the applications at the endpoints to have direct access to some form of media handling capabilities. In the example presented above, when Chiedza starts talking to Tino, voice samples are created that can be sent directly over IP. However, application level protocols called media protocols are generally used to carry media. Different media transport protocols are suited for specific types of media. For example, RTP is typically used if the media is voice/video. This is because RTP contains features that facilitate the transport of pure real-time traffic.

Since both SIP and RTP are application level protocols, they use the services provided by transport protocols such as UDP/TCP. This means that VoIP faces latency and integrity issues which rise from the IP protocol. Notwithstanding these issues, the constraints on real time behaviour of VoIP are strict. In order for real time communication to work, this

entire process has to be done with minimal latency. Psaiar [47] argues that interruptions lasting more than 200ms are unacceptable in a VoIP conversations. One solution to this problem is to combine SIP, RTP and RTCP. SIP takes the role of the session control protocol and initiates calls. RTP transmits the voice data while RTCP deals with the exchange of connection information to monitor the quality of the connection.

The set of functions and elements that participate in the processing and exchange of the media are said to form the media plane. The media plane and the control plane are integral parts of any IP multimedia communication system. Thus, it was crucial to verify that JSAP's implemented media functions could support audio/video before we could upgrade it into an IMS client. In the Chiedza and Tino example we discussed how a simple voice communication may be enabled on the Internet, and we have highlighted the need for:

- A signalling transport protocol to carry the control information (signalling).
- A media transport protocol to carry the real time user information (media).
- An application in the endpoints that is able to:
  - capture the voice samples from the microphone and send them over the network using a media transport protocol.
  - receive the media transport protocol packets, get the voice samples, and feed them to the sound-card to be played.

As has already been mentioned, many applications that transmit/stream audio and/or video over an IP network typically use RTP as the media transmission protocol. Such applications have corresponding profiles called the audio video profiles (AVPs) and payload format specifications. These profiles are defined in a combined document, RFC 3551 [55]. This RFC includes a definition of several possible payload types for audio and video. Some of the most common ones are presented in Table 2.7.

Payload types can be static or dynamic. Static payload types are defined with a fixed identification number. Dynamic payload types do not have a number statically assigned. The assignment is done in a dynamic way, typically via signalling (for instance, using SDP). Identification numbers between 96 and 127 are allocated to dynamic payload types.

Table 2.7: AVP Payload Types

Payload Type	Encoding name	Media Type	Clock Rate	Channels
0	PCMU	Audio	8000	1
3	GSM	Audio	8000	1
4	G723	Audio	8000	1
8	PCMA	Audio	8000	1
26	JPEG	Video	Variable	-
31	H261	Video	90000	-
34	H263	Video	90000	-
96-127	Dynamic	Audio/Video	-	-

## 2.3 The IP Multimedia Subsystem

In this section, we will introduce the IMS. In order to explain what IMS is, we will use the previous section on SIP and SIP network architectures to outline how, starting from a SIP network and adding IMS requirements, we end up with the IMS architecture.

### 2.3.1 IMS Overview

The IMS is a global, access independent and standard based IP connectivity and service control architecture that enables various types of multimedia services to be made available to end-users using common Internet-based protocols [46]. Its core network has a common IP based transport and signalling, which can be accessed by different networks. SIP matches the network access requirements for IMS because it allows applications to remain agnostic of the access network. Hence, it was chosen as the main standardised signalling protocol for the IMS.

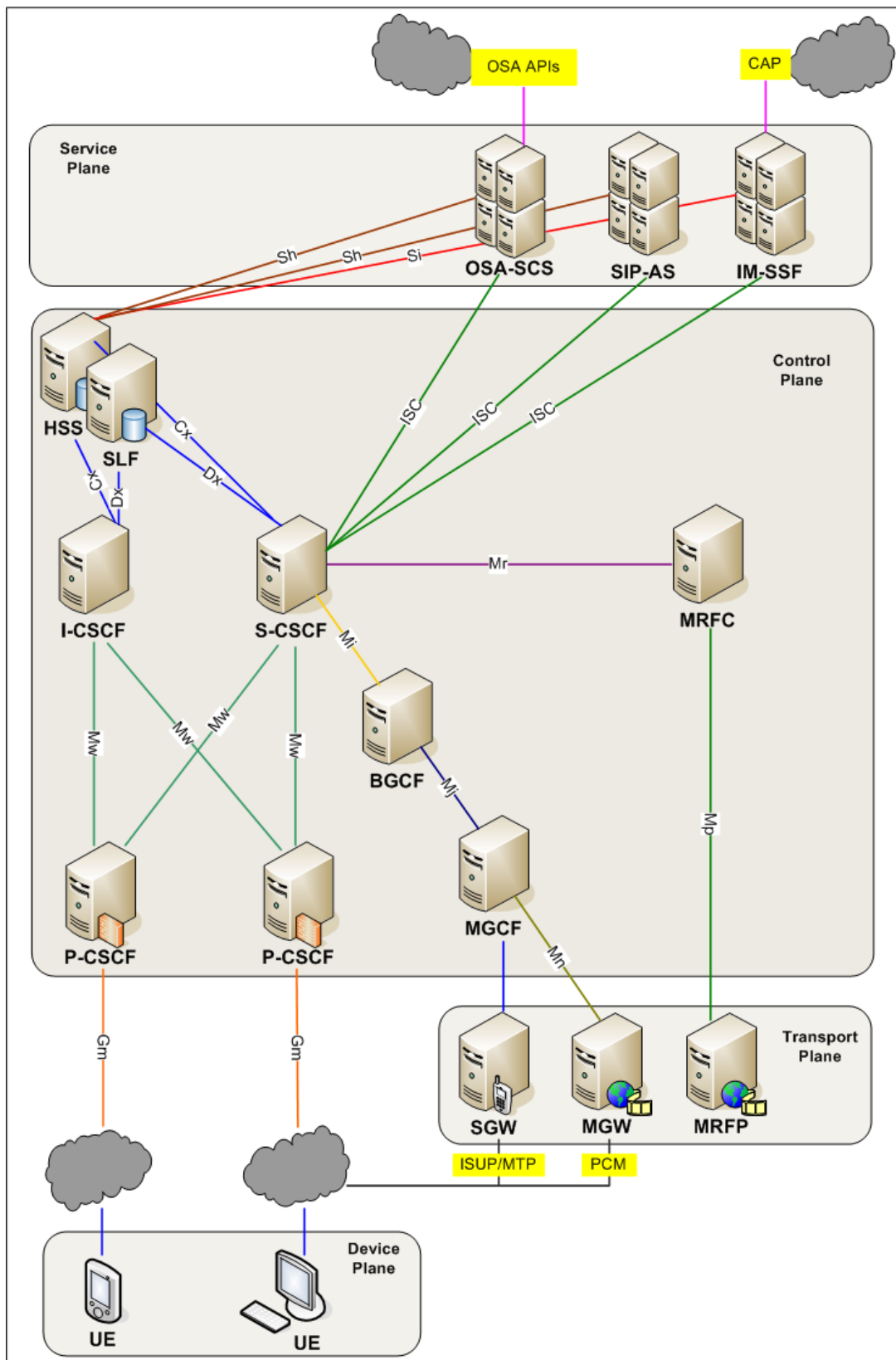
Since its introduction, IMS has been adopted by several major telecommunication standardisation bodies in mobile and fixed networks as the basis for the Next Generation Network (NGN). Unlike traditional IP-based networks, IMS guarantees end-to-end quality of service (QoS) within the network. Similar to IP-based networks, IMS creates an infrastructure that enables the fast deployment of new IP-based services and flexible billing, while maintaining compatibility with existing applications [39].

### 2.3.2 IMS Architecture

The IMS architecture defined in 3GPP TS 23.228 [5] is at the heart of the convergence of voice, data, fixed and mobile networks and is based on a wide range of IETF protocols.



IMS combines and enhances these protocols to allow real-time services in addition to 3GPP mobile packet-switched (PS) domain and the wire-line NGN [39]. The IMS architecture comprises four logical planes, or layers, which correspond to discrete functions as depicted in Figure 2.3.

Figure 2.3: IMS Functional Planes (Adapted from Oguejiofor *et al* [43])

Each plane consists of IMS functional components that together provide the supported functions at that layer [43]. The device plane consists of the user terminals used to access the IMS services. An IMS-capable device or an IMS client can be used to access IMS network. The device plane is where the work of this thesis is situated. The device plane includes smart-phones, switch-phones and other advanced IP phones. The standardisation of this plane also forms part of the work of the 3GPP and other major IMS standardisation bodies, for both wireless and wire-line networks.

The transport plane refers to the access network used by IMS terminals to access the IMS network. Included in this plane are IMS components such as routers, media gateways and switches. These components translate protocols between the IMS core network and the connecting network. The transport plane also shields the upper layers of the IMS architecture from the network access technologies by providing a common access interface to the components in this plane [43].

The Call Session Control Functions (CSCFs) form the core of the IMS control layer. There are three types of CSCFs namely: the Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF) and the Serving-CSCF (S-CSCF). The Home Subscriber Server (HSS) database is another element within the control layer. Other elements forming the IMS control plane include the Breaking Gateway Control Function (BGCF), Media Resource Function (MRF) and others, which will be discussed in the next section.

The service plane consists of Application Servers (ASs). All the IMS services run within the ASs and a single AS can handle multiple multimedia services. ASs also provide interfaces with the control layer using SIP. For example, the IMS-Service Control (ISC) interface is a reference point between S-CSCF and ASs whose main functions are to:

- Notify the ASs of the registered IMPU, registration state and UE capabilities.
- Supply the AS with information to allow it to execute multiple services.
- Convey charging function addresses.

Some examples of ASs include the presence servers, group list management servers, IM servers and conferencing servers [39].

### 2.3.3 Elements of the IMS Architecture

The IP multimedia core network subsystem (IMCNS) includes the different functional components of network infrastructure for delivering multimedia services [46]. As earlier

pointed out, these components are divided into four planes (as shown in Figure 2.3) each of which performs a specific function. The components include databases for maintaining subscriber information, call and session control components, media and application servers, media/signalling gateways and user equipment for accessing the network. Only components that are important to IMS registration and session setup will be discussed in this thesis.

### 2.3.3.1 IMS Terminals

Typically referred to as User Equipment (UE) are the IMS capable terminals used by subscribers to access IMS services. They contain the SIP UA that generates and terminates SIP messages on the user's behalf. Once an IP address has been allocated for registration, the UE cannot change it while engaged in an active Dialog [39]. As the name IP multimedia subsystem suggests, a fundamental requirement is that UEs must have some form of IP connectivity in order to access the IMS. Examples of UEs are mobile phones, personal data assistants (PDAs) and computers.

### 2.3.3.2 SIP Servers

These are also known as Call Session Control Functions (CSCFs). They perform session control functions for IMS sessions. CSCFs can be categorised into three groups, based on their functionality:

#### 1. Proxy - CSCF (P-CSCF)

The Proxy Call Session Control Function (P-CSCF) is the first contact point for users within the IMS. P-CSCF performs the role of a SIP Proxy Server for inbound and outbound messages from an IMS Terminal (UE). This means that all SIP signalling traffic from the UE will be sent to the P-CSCF. Similarly, all terminating SIP signalling from the network is sent from the P-CSCF to the UE.

There are four unique tasks assigned for the P-CSCF:

##### (a) SIP compression

Given that SIP is a text-based protocol, it contains a large number of headers and header parameters, including extensions and security-related information. This means that typical SIP message sizes are larger than those in binary-encoded protocols. For speeding up the session establishment and reducing

bandwidth consumption on the access network, 3GPP has mandated the support of SIP compression between the UE and P-CSCF. The P-CSCF compresses messages if the UE has indicated that it wants to receive signalling messages compressed.

(b) IPsec security association

P-CSCF is responsible for maintaining Security Associations (SAs) and applying integrity and confidential protection for SIP signalling. This is achieved during SIP registration as the UE and P-CSCF negotiate IPsec SAs. After the initial registration, the P-CSCF is able to apply integrity and confidential protection to SIP signalling.

(c) Interaction with Policy and Charging Rules Function (PCRF)

(d) Emergency session detection

P-CSCF plays an important role in IMS emergency session handling. It is responsible for the detection of emergency requests. P-CSCF is expected to reject/re-route emergency attempts based on operator policy (e.g. user is attempting to make an emergency call via home P-CSCF when roaming) or based on network capability [43, 46].

## 2. Serving - CSCF (S-CSCF)

The S-CSCF is the hub of all signalling functions in an IMS network. It is responsible for handling registration processes (performs the role of a SIP registrar), recording the location of each user and also for performing the user authentication, call processing and routing of calls to the Application Servers (ASs). When a user sends a registration request, it will be routed to the S-CSCF, which downloads authentication data from the HSS. Based on the authentication data, it generates a challenge to the UE. After receiving the response and verifying it, the S-CSCF accepts the registration and starts monitoring the registration status. After this procedure, the user is able to initiate and receive IMS services.

All incoming/outgoing messages to/from a UE traverse the allocated S-CSCF, which inspects these messages in order to establish the steps that need to be taken (for example authorising a user for a particular action, based on the user profile). An S-CSCF therefore performs routing functions based on the message it receives. When the S-CSCF receives a UE-originating request via the P-CSCF it needs to decide if ASs are to be contacted prior to sending the request further on. After possible interaction with ASs, the S-CSCF either continues a session in IMS or breaks to

other domains (CS or another IP network).

Similarly, the S-CSCF receives all requests which will be terminated at the UE. Although, the S-CSCF knows the IP address of the UE from the registration, it routes all requests via the P-CSCF, since the P-CSCF takes care of SIP compression and security functions. Prior to sending a request to the P-CSCF, the S-CSCF may route the request to an AS (for instance, to check possible redirection instructions). The user profile (downloaded by the S-CSCF from the HSS) instructs an S-CSCF whether the SIP signalling message should be routed to one or more ASs before it is routed to the final destination [43, 46].

### 3. Interrogating - CSCF (I-CSCF)

I-CSCF is responsible for querying the HSS to determine the S-CSCF for the user. It is the contact point within an operator's network. It is also responsible for establishing the interface between two different IMS networks such as the home and visitor network. An I-CSCF has the Topology Hiding Inter-network Gateway (THIG) which can be used by the network operator to hide network configuration and topology. This function hides the addresses of operator network entities from being passed outside the operator's network [39].

Strictly speaking, an I-CSCF is also a SIP Proxy Server. However, its location and function is more specific. It is located at the edge of an administrative domain of a network. When a P-CSCF wants to find the next hop for a SIP message, it obtains the address of the I-CSCF of the destination network. I-CSCF is therefore the contact point within an operator's network for all connections destined to a subscriber of that network operator.

There are three unique tasks assigned to the I-CSCF:

(a) Obtaining next hop name

Accessing the name of the next hop (either S-CSCF or application server) from the Home Subscriber Server (HSS).

(b) S-CSCF Assignment

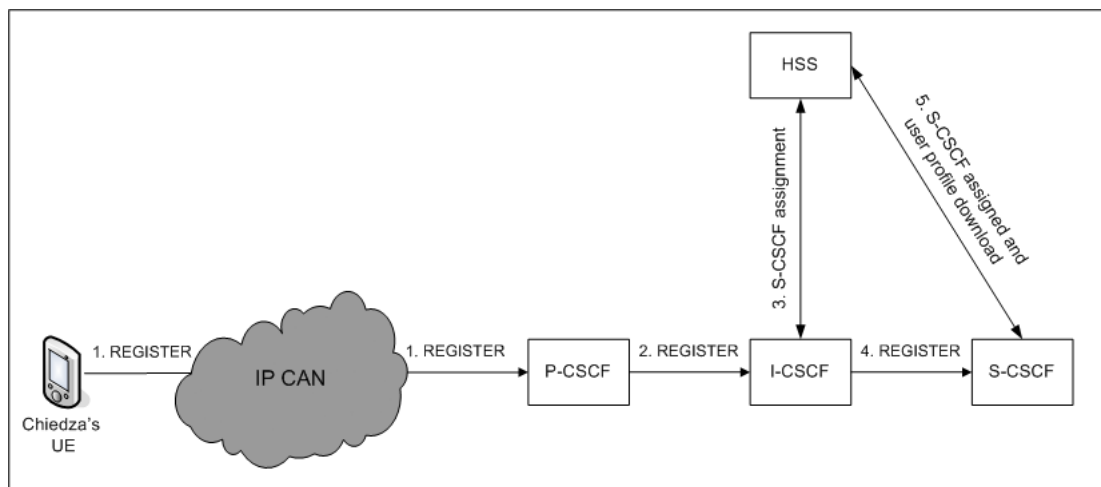
Assigning an S-CSCF based on received capabilities from the HSS. The assignment of the S-CSCF will take place when a user is registering with the network or a user receives a SIP request while they are unregistered from the network but has services related to an unregistered state (e.g. voice mail).

## (c) Request Routing

Routing incoming requests to an assigned S-CSCF or the application server (in the case of public service identity). The I-CSCF uses its Diameter interface with the HSS/SLF to find the S-CSCF assigned to the UE. It subsequently forwards the incoming SIP message to the appropriate S-CSCF [43, 46].

Figure 2.4 demonstrates how the various CSCF types (P, S and I) work together.

Figure 2.4: <P, S and I> - CSCF working together



### 2.3.3.3 User Databases

The IMS architecture contains two main databases: home subscriber server (HSS) and the subscription locator function (SLF). The HSS provides the main data storage for all subscriber and service-related data of IMS. The data stored in the HSS includes public and private user identities, registration information, access parameters, service-triggering information, and user-specific requirements for S-CSCF capabilities. The SLF is used by network operators who have multiple HSSs as a resolution mechanism that enables the I-CSCF, S-CSCF and the AS to determine the address of the HSS that holds the subscriber data for a given user identity [39, 43, 46].

### 2.3.4 UE Procedures

Among the IMS components discussed, the UE is a critical entity for the overall success of the IMS value chain. This is because the UE is the only component that lists to the user IMS services found on the network. Bachman [9] argues that the presentation of these

services to the end user determines the return on investments on IMS. To highlight the role that the UE performs in the IMS playing field, the steps that the UE must perform in a GPRS/UMTS network before the IMS services can be accessed will be presented. Attention will be focused on procedures relevant to a UE built for end user devices that do not use ISIMs.

#### 2.3.4.1 Local P-CSCF Discovery

Our UE accesses the IMS directly from a packet switched fixed network, so the first step is to discover the local P-CSCF before the user can register on the IMS network. The P-CSCF is the first contact point for the UE in the IMS network. The 3GPP suggested two methods (in 3GPP TS 23.228 [5] and 3GPP TS 24.228 [2]) that can be used by the UE to discover the P-CSCF :

1. Use of dynamic host configuration protocol (DHCP)  
DHCP can be used to provide the user with the domain name of a P-CSCF and the address of a Domain Name System (DNS) server that is capable of resolving the P-CSCF name as specified in RFC 3319 [57].
2. Use of IP-core access network (IP-CAN) provisioned services  
Some IP-CANs provide the capability to derive the P-CSCF address as part of the access bearer establishment process.

Another approach that is used in some deployments and not recommended, consists of manually configuring the name or address of the P-CSCF in the terminal. Once assigned to a user, the P-CSCF does not change while the user remains connected to the access network [45].

#### 2.3.4.2 Application Level SIP Registration

The third step to accessing IMS services is UE application level SIP registration. Registration creates bindings in a location service for a particular domain that associates an address-of-record (AOR) uniform resource identifier (URI) with one or more contact addresses [54]. The UE uses the same registration procedure for registering on the home or visited network. Furthermore, it can also register multiple public identities through a single IMS registration procedure.



### 2.3.4.3 Service Route Discovery During Registration

Earlier in this chapter we saw that outgoing requests from a UE need to traverse the originating user's S-CSCF so that the S-CSCF can apply for services on the user's behalf. We also saw that S-CSCFs are assigned to the users dynamically at registration. Furthermore, we discussed how the UE determines what P-CSCF to use for sending originating requests, but not how the UE determines which S-CSCF to use for outgoing requests. This issue is resolved by a new SIP extension defined in RFC 3608 [65]. This extension defines a new header field called the *Service-Route* header, which is generated by the registrar (the S-CSCF in the IMS case) and is included in successful responses to the REGISTER message. The *Service-Route* header conveys the name of the home service proxy (S-CSCF) where the UA must direct its requests. Once the UE has received the response, that is, the "200 OK" to the REGISTER, it will include both the P-CSCF name and the S-CSCF name in the *Route* header of all outgoing requests [11]. Once the above steps have been performed successfully, the UE is ready to establish a SIP session to access IMS services.

## 2.3.5 IMS Concepts

In the previous section, we have seen that the IMS architecture requires additional functions on top of the basic SIP architecture. Next we will describe in detail some fundamental IMS concepts, and highlight the differences they present when compared to a basic SIP network.

### 2.3.5.1 IMS Identities

In a basic SIP network, the end user is typically assigned a public identity and some security credentials. The public identity typically has the form of a SIP URI such as `chiedza@open-ims.test`. When Chiedza registers to her SIP server, she uses her public identity, which is then authenticated by the server. The public identity is also employed by other users in order to request communication with Chiedza.

In IMS, the end user is assigned two identities by the home network operator: IP Multimedia Private Identity (IMPI) or Private User Identity (PrUI), and IP Multimedia Public Identity (IMPU) also referred to as Public User Identity (PUI). The PUI represents the identity that is employed by other users to request communication with the user. The PUI therefore identifies the user and used to route SIP requests. On the other hand, PrUI is exclusively used for identifying the user's subscription and authentication purposes. The

PrUI is unique to the UE, that is, it is used to identify the user's device. A user can therefore have multiple PUIs per PrUI.

### 2.3.5.2 IMS Security

Security is a fundamental requirement in every telecommunication system and the IMS is not an exception. The IMS has its own authentication and authorisation mechanisms between the UE and the IMS network in addition to access network procedures. Moreover, the integrity and optional confidentiality of the SIP messages is provided between the UE and the IMS network and between IMS network entities regardless of the underlying core network. This means that IMS provides at least a similar level of security as the corresponding GPRS, circuit-switched or packet switched networks: for example, the IMS ensures that users are authenticated before they can start using services, and users are able to request privacy when engaged in a session [46]. IMS security encompasses two aspects:

1. Access Security (AS)

AS, described in 3GPP TS 33.203 [3] refers to the provision of security services such as authentication, integrity, and confidentiality for the SIP signalling path between the user and the IMS network. Mutual authentication between the user and the network is based on the UMTS Authentication and Key Agreement (AKA) protocol. SIP employs a user authentication scheme that is based on the HTTP Digest mechanism. Therefore, there is a need to map the AKA parameters onto HTTP Digest authentication. Such a mapping is described in RFC 3310 [40].

2. Network Domain Security (NDS)

NDS, described in 3GPP TS 33.210 [7] refers to the provision of authentication, confidentiality, integrity, and replay protection between different IMS networks (security domains) or between nodes within the same security domain. In order to achieve NDS, security gateways (SEG) are deployed in the interconnecting networks. Each SEG is responsible for setting up and maintaining security associations with its peer SEGs. The SAs are negotiated using the Internet Key Exchange (IKE) protocol defined in [31]. The authentication is based on pre-shared secrets [45].

Because the IMS has its own authentication and authorisation mechanisms that are used between the UE and the IMS network we needed to add these extensions in order to make JSAP IMS compliant.

### 2.3.6 Private SIP Extensions for 3GPP IMS

SIP includes a specific type of extension referred to as “private”. These extensions are either not ready for standards track, but may be negotiated for use by communicating UEs, or they are private/proprietary in nature, because a characteristic motivating them is usage that is known not to fit the Internet architecture for SIP [37]. Private headers include the “P-” prefix and are typically defined for SIP usage in non-Internet, controlled network scenarios such as those occurring in telecom operators’ networks e.g. when we deal with IMS.

RFC 3455 [22] defines a number of private SIP extensions that were introduced due to IMS requirements. Next we briefly describe some of the extensions that we needed to add in order to make JSAP IMS compliant.

1. *P-Visited-Network-ID* Header

When a user roaming in a visited network attempts to register, there is a need to convey the information about the visited network to the home S-CSCF so that it can check if there exists a roaming agreement with the visited network. In order to convey this information, a new private header has been defined that contains a text string that identifies the visited network. The P-CSCF in the visited network adds this header into the REGISTER message that is sent to the home S-CSCF. Example: P-Visited-Network-ID=“Telecom Italia Mobile” [46].

2. *P-Access-Network-Info* Header

There are cases, especially when a wireless-access network is used, when the services to apply may depend on the technology of the access network or the location of the user (e.g., the cell from which a call or other IMS service originates). The new private *P-Access-Network-Info* header is capable of conveying that information from the UE to the IMS network. This header is populated by the UE based on the information it gets from other sources (for example, radio signalling) [45].

3. *P-Associated-URI* Header

We saw in previous sections that an IMS user may be associated with more than one PUI. When the user sends a REGISTER message to the network in order to register a particular PUI, the S-CSCF responds with a “200 OK” that includes the *P-Associated-URI* header that lists all the associated Public User Identities. The presence of a URI in the P-Associated-ID does not mean that such a URI is registered, only that it is associated with the Public User Identity that has been registered.

### 2.3.6.1 IMS Services

IMS being essentially a SIP based multimedia network, most of the SIP services that we discussed in previous sections can be offered over an IMS infrastructure. In many cases, these services running as IMS applications are located in the terminals, true to the end-to-end nature of SIP. In other cases, the applications sit on Application Servers on top of the IMS network. Hybrid situations are also common.

The aim of 3GPP is not to standardise all the applications, but rather to provide service capabilities. Nevertheless, there are some important applications that have been specified both by 3GPP and/or open mobile alliance (OMA), given the need to ensure interoperability and inter-working across different operators' networks.

## 2.4 Summary

In this chapter, we explained what multimedia communications are and the role that SIP plays in this regard. We also looked at some examples of services that might be delivered through SIP. We have seen that a true multimedia communication system requires in terms of information exchanges:

- Exchange of media information (voice or others). This is governed by a media transport protocol such as RTP or others.
- Exchange of control information (signalling). This is governed by a signalling protocol such as SIP or others.

In this chapter, we have also seen that SIP plays a major role in IMS. It has also been shown that the IMS, in addition to the core SIP specification, incorporates many SIP extensions and SIP network functions. In some cases, these extensions have already been proposed and standardised while in other cases, they have been defined based on specific IMS requirements and are still awaiting standardisation.

# Chapter 3

## Existing IMS Clients and JSAP

The RUCRG IMS client project was started with the goal of upgrading the JSAP into an IMS compliant client. Although the focus of this research was not to compare available IMS clients, we found it interesting to examine the IMS clients which were used in the Rhodes University Convergence Research Group (RUCRG). This chapter provides an overview of the feature sets of three freely available IMS clients that are currently being used in the RUCRG. Furthermore, this chapter will also provide an in-depth overview of the JSAP which was used as the foundation for the RUCRG IMS client.

### 3.1 IMS Clients

In this section we will provide a brief assessment of three, free IMS clients used within the RUCRG: IMS Communicator, UCT IMS client and Mercurio. Table 3.1 provides a comparative assessment of the important features for IMS compliance that each of these clients possesses. Also included are the platforms that the clients are compatible with as well as the licensing of the clients.

Table 3.1: Feature Summary of UCT IMS client, IMS Communicator and Mercurio. Adapted from “The UCT IMS Client” [62]

	UCT IMS client	IMS Communicator	Mercurio (Bronze)
Registration	AKAv1/2-MD5; MD5	AKAv1-MD5; MD5	AKAv1/2-MD5; MD5
IMS Signalling	PRACK support Pre-condition support	PRACK support Pre-condition support	PRACK support Pre-condition support
Media Support	Audio / Video	Audio / Video	Audio / Video
Presence Support	Presence support Watcher authentication	No presence support	Presence support Watcher authentication
Instant Messaging	Pager mode / Session-based	No support	Pager mode / Session-based
XCAP Support	XCAP support	No XCAP support	XCAP support
Platform	Linux	Windows / Linux	Windows
License	GPLv3 (free and open source)	LGLP (free and open source)	Free and closed source

### 3.1.1 IMS Communicator

IMS Communicator is an IMS client based on the SIP Communicator Java project [28]. It is implemented on top of the JAIN SIP stack [30] and the Java Media Framework (JMF) API [59]. The use of JMF as a media API presents a variety of challenges. Firstly, there have been significant improvements in video coding technologies over the last few years but JMF supports a limited set of these codecs. Secondly, Sun Micro-Systems ceased to support JMF in 2003 [59]. Lastly, JMF installation and configuration is complex, especially for ordinary users.

IMS Communicator does not store user data in a central repository on the network. The presence list is stored on the client, meaning that subscriptions are created and managed for each presentity in the list by the client. Furthermore, IMS Communicator classes are overloaded with responsibilities, making them difficult to debug, test and extend. For example, SIP messages are received and processed by the same class. Registration with the FOKUS IMS Core typically fails, and there are existing bugs that have not been fixed in a long time.

### 3.1.2 UCT IMS Client

The UCT IMS client is a free open source implementation of a 3GPP IMS client developed in ANSI C [62]. It supports a variety of IMS applications such as IM, presence, VoD/IPTV, and the XCAP protocol among others. It was designed to be used on the Linux platform and has been key in helping the RUCRG build their IMS testbed at Rhodes University.

### 3.1.3 Mercurio IMS Client

Mercurio IMS client is closed source, proprietary and comes pre-compiled thus cannot be extended. It comes in various versions one of which is free and supports a limited set of functions [14]. Similar to the UCT IMS client, Mercurio is not cross platform. It is built only for the Windows environment. The Mercurio IMS client project has been stopped and the development team has been dissolved [15]. Given that the development and standardisation of IMS and its associated services is an ongoing process this presents a big challenge. Client development needs to keep up to date with changes in the IMS to remain compatible with evolving IMS standards.

## 3.2 JAIN SIP Applet Phone (JSAP)

JSAP is an open source project which possesses some of the basic features which are required in a SIP/IMS compliant client such as voice and text instant messaging (IM). It was chosen as the foundation of the RUCRG IMS client for the following reasons:

- The JSAP project leadership was at Rhodes where one of the key developers of the initial project was based.
- JSAP was written in Java.
- It supported core SIP signalling.
- It used JAIN SIP (a low level Java API for SIP signalling for flexible handling of the SIP protocol).

There was need to perform an extensive assessment of the JSAP, particularly because there was no documentation and that, it only supports core SIP functionality. This section will critically look at the architecture of the JSAP and some of its limitations.

### 3.2.1 JSAP Architecture

In order to make the necessary IMS enhancements, a full analysis of JSAP was carried out. The assessment involved carrying out systematic experiments that included: tracing messages sent by JSAP using network analysis tools to check their sequencing and well formedness, and reverse engineering to find out the relationship among the various classes.

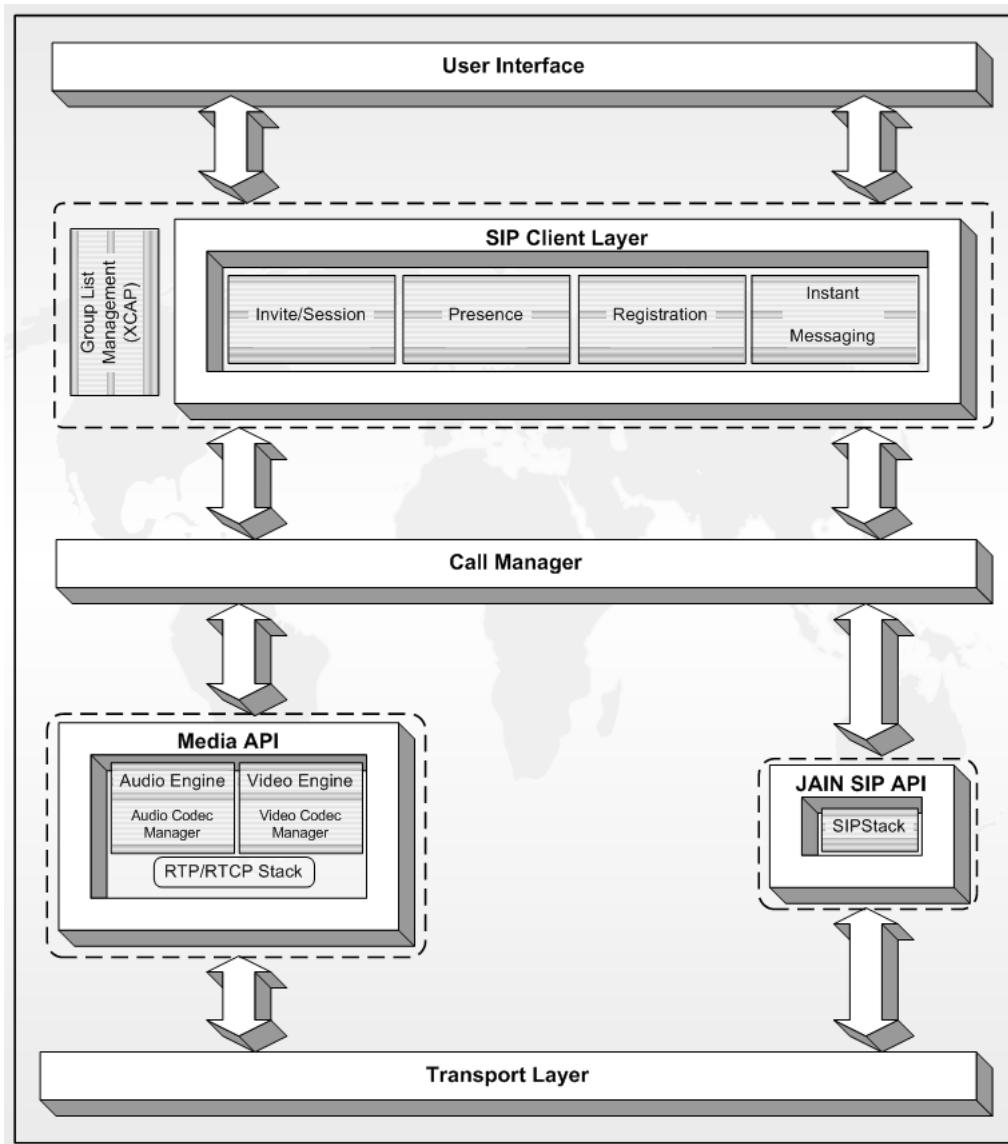
The classes which needed to be modified, removed or replaced were identified. This was done in light of the fact that IMS functionality is built on top of ordinary SIP functionality. Since the JSAP already supports ordinary SIP registration and SIP session setup, IMS specific parameters needed to be added to allow the existing SIP headers to be reused whenever possible for IMS. In summary the process, in consultation with some of the original developers of JSAP, involved:

- Studying the structure of the JSAP and identifying the classes which needed to be modified to add IMS support.
- Adding helper classes for populating IMS specific parameters.
- Removing and/or replacing some existing classes with optimised ones that allow the support of IMS.
- Adding XML support to allow the populating of IMS/SIP attributes and to allow persistence user data such as user-names and proxies.

Having gone through the aforementioned processes a structure of JSAP was drawn up. Figure 3.1 gives an overview of the architecture of the JSAP that resulted from the study above and specific experiments that were done in the cases in which static analysis was inefficient or inconclusive.



Figure 3.1: JSAP Derived Architecture



The dotted lines show various components of JSAP that needed to be added or modified to make it IMS compliant.

A brief overview of some of the components that were identified in the JSAP are as follows:

1. Invite/Session - provides high level management to call control. The call setup procedure for an IMS call is more complex as the SIP precondition and reliable provisional response mechanisms are used.
2. Presence - provides functionality to manage presence information of the client and associated contacts. Client/Server mode needed to be added.

3. Registration - hides the complexity of the SIP registration process including dealing with multiple types of user identities. The registration procedure to the IMS is more complex as the Authentication and Key Agreement (AKA) algorithm is used together with md5.
4. Instant Messaging - enables sending and receiving of IMs to and from buddies. JSAP supported pager mode IM in which messages were sent in the body of SIP MESSAGE requests and no sessions were established.
5. Call manager - provides the mechanisms for controlling calls. It acts as an interface for controlling SIP related communications. This includes SIP based calls and registration. It interfaces with presence and IM module to provide proper signalling for IM and presence.
6. SIP stack - provides a low level API that provides full control over SIP communication between the client and IMS.
7. RTP/RTCP Stack - provides low level API to provide full control over real time data transport between the client and the application server or another client.

### 3.2.2 Limitations

The assessment of the JSAP also brought to light a variety of limitations in the client:

- JSAP lacked support for IMS functionality, that is, it was an ordinary SIP client that could not be used in an IMS setting.
- Presence in JSAP was implemented in a peer to peer manner but ideally should also support Client/Server (JSAP lacked support for network storage of user data).
- Video implementation in the JSAP was not fully functional and required attention (JMF failed to initialise video capture devices in Linux but worked under Windows. JMF also lacked support for some of the new high quality well compressed codecs).
- JSAP only supported basic SIP signalling.
- JSAP assumed that media payload formats were always static.

### 3.3 Summary

Freely available IMS client applications lacked features required to test the applications being developed (they support a subset of the required functions) by the RUCRG. For instance, some of the IMS clients discussed could only be used on specific platforms [61, 15, 62] while others supported a limited range of video and audio codecs [43, 21]. This meant that researchers were forced to switch between clients or adjust their systems during testing thus posing challenges and extending time to market for applications.

# Chapter 4

## Development Tools

As explained in chapter 2, an IMS client requires several IETF protocols in order to perform its various functions; for example it requires Session Initiation Protocol (SIP) [54], Session Description Protocol (SDP) [25], Real-time Transport Protocol (RTP) [56] and XML Configuration Access Protocol (XCAP) [51]. There are several existing APIs that implement these protocols. This chapter provides an in-depth overview of the APIs that were used for developing the RUCRG IMS client. Additionally, this chapter will also discuss the software and tools that were used for developing the RUCRG IMS client.

### 4.1 Application Programming Interfaces (APIs)

As alluded to in the previous section, the RUCRG IMS client uses a number of existing APIs that implement different IETF protocols. These APIs can be categorised in several ways.

Firstly, they may be split into:

- Proprietary - vendors expose functionality in their product by defining their own APIs that can be used only within their platform.
- Open standard - standardisation bodies define a number of standard APIs for application development.

Another possible categorisation of APIs refers to the level of abstraction the interface provides:

- High - level APIs completely hide the functionality, and offer an abstract programming model that is largely decoupled from the concepts.
- Low - level APIs give the programmer the capability to manipulate the objects at the lowest level.

For the purposes of the RUCRG IMS client, open standard low-level APIs were used for development. This was done in an effort to reduce cost of development because open standard APIs are free and can be used with limited restriction. Consequently this allows a much larger community of developers to work on the client in the future since no licensing will be required. Additionally, low level APIs allow a lot of flexibility meaning more complex applications can be developed.

#### 4.1.1 Media APIs

The transmission of high quality multimedia data over IP based communication links, has been made possible by significant increases in network bandwidth along with the improvement of audio/video coding technologies. This has led to the increase in the demand of audio/video services [38]. The available quality of the delivered media is closely related to the system used for delivery. The main challenge to clients is to provide decoders, encoders and transmission formats that fit at least one of the requirements of the remote user equipment (UE) that they are communicating with [45].

In order to create applications that manipulate media such as voice and video, a media API is required to access the media capabilities of the underlying platform. A number of media APIs are currently available, as proprietary platform APIs or standard cross-platform APIs. They all try to solve the issue of media delivery quite differently [47]. Still, they all process media using various handlers for formats, streams and contents.

A good media API should provide specialised libraries and interfaces that make it possible to combine new and customised multimedia solutions as well as a plug-in architecture that allows addition of new codecs, formats, capture devices and communication procedures comfortably [38]. The Java Media Framework (JMF) API enables audio, video and other time-based media to be added to applications and applets built on Java platform technology. It is the media API used in the JSAP, and one of the major drawbacks in the client's development due to; the bugs it contains (which require workarounds), the lack of support for new media codecs and its difficult installation and configuration procedure. Further, Sun Micro-systems ceased to support JMF in 2003 and has since been acquired

by Oracle [44]. According to Wikipedia [63], JMF has not been enhanced since 1999, and the last news item on JMF's home page was posted in September 2008.

Due to the reasons explained above, the use of JMF as the media API in JSAP necessitated the need to find an alternative media API. There were a variety of open source implementations that were assessed. Among the APIs that were investigated there have been several efforts to implement open-source alternatives that use JMF as a building block; for example FMJ (an open source initiative which was started to implement and extend JMF). Next, we briefly describe some of the popular APIs used for direct media manipulation that were investigated as possible replacements to JMF.

#### 4.1.1.1 FMJ

FMJ is an open-source project that was established with the goal of providing an alternative to JMF, while remaining compatible with it. According to Ken Larson (FMJ project leader), FMJ aims to produce a single API/framework which can be used to capture, playback, process, and stream media across multiple platforms. He further argues that FMJ extends beyond Sun Micro-system's JMF by enhancing platform-specific support, or performance packs, for Mac OSX and 64-bit Linux, providing modern codecs, such as MPEG4, improving overall performance, and simplifying installation [21]. Codec support is addressed in FMJ by wrapping a platform's native media applications, such as DirectShow, Quicktime and Gstreamer [20]. FMJ has two sub-projects and one sister project. The sub-projects, FFMPEG-Java and Theora-Java, are Java wrappers for FFMPEG and Vorbis respectively. The sister project is LTI-CIVIL and it is used as the primary video capture device library [21].

Since FMJ is compatible with the latest JMF, one may use existing JMF applications without modifying them. However, several areas of the FMJ project are under development, and sometimes workarounds are needed, if existing JMF applications do not work. This may mean several code modifications as development progresses.

#### 4.1.1.2 FFMPEG

FFMPEG is a complete, cross-platform solution to record, convert and stream audio and video [18]. It includes libavcodec [64] an open source GNU lesser general public licence (LGPL) licensed library of codecs for encoding and decoding video and audio data. It is one of the leading audio/video codec libraries and is used in many open-source multimedia applications and frameworks. FOBS (FFMPEG OBjectS) is an open source

object oriented wrapper for ffmpeg. FOBS relies on the FFMPEG library, but provides developers with a much simpler programming interface. However, FFMPEG is currently available only in C++. The Java version (Fobs4JMF) has been implemented as a JMF plug in that allows JMStudio (a media player included with the JMF programming tools) to play the most common formats and codecs (ogg, mp3, m4a, divx, xvid, h264, mov, avi, etc.). Binaries for this enhanced version of JMStudio are available for Windows, Linux and Mac OSX which can be used to include support for other formats and codecs into JMF applications without altering the original code [58].

#### 4.1.1.3 JFFMPEG

This is a Java wrapper for FFMPEG. The JMF plug-ins system lets one use JMStudio or other Java applications to play mpeg1, h263, mpeg4 (divX), etc. streams. It is based on a Java port of parts of the FFMPEG project, supporting a number of codecs in pure Java code. Where codecs have not yet been ported, a Java native interface (JNI) wrapper allows calls directly into the full FFMPEG code. However there is a feeling among the developer community that JFFMPEG is dead, and FOBS which also acts like a wrapper is a better alternative.

#### 4.1.1.4 FFMPEG-Java

FFMPEG-Java is not the same thing as JFFMPEG. FFMPEG-Java is a Java wrapper around FFMPEG, using JNA (Java Native Access). It assumes that dynamic libraries for FFMPEG have been compiled, and are included in one's library path [19].

#### 4.1.1.5 LTI-CIVIL

LTI-CIVIL (Larson Technologies Inc. Capturing Images and Video in a Library) is a Java LGPL licensed library for capturing images from a video source such as a USB camera. It provides a simple API and does not depend on or use JMF. The FMJ project integrates LTI CIVIL into the JMF architecture by providing a civil data-source in place of a regular JMF data-source.

**Current Capture Rates** According to LTI-CIVIL [35], basic image capture works on the following platforms at the specified rates and quality:

- 20fps at 320x240 on Windows 2000/XP/Vista

- 7fps at 160x120 on GNU/Linux 32/64-bit
- 7fps at 640x480 on Mac OS

#### 4.1.1.6 Gstreamer

Gstreamer is a framework for creating streaming media applications. Most of the valuable qualities that the Gstreamer framework possesses come from its modularity. The framework is based on plug-ins that provide various codecs and other functionality. Gstreamer can seamlessly incorporate new plug in modules. The plug-ins can be linked and arranged in a pipeline. This pipeline defines the flow of the data. Gstreamer's development framework makes it possible to write any type of streaming multimedia application. The Gstreamer framework is designed to make it easy to write applications that handle audio and/or video. It is not restricted to audio and video as it can also process any kind of data flow. The pipeline design is made to have as little overhead as possible above what the applied filters induce. This makes Gstreamer a good framework for designing even high end audio applications which put high demands on latency [60].

Gstreamer allows programmers to configure media processing scenarios that combine different input, output, and processing options. It offers a high level API to manage the data capture, presentation, and processing of time-based media. Additionally, it also offers a low-level API that supports the seamless integration of custom processing components and extensions. We will be focusing on the Gstreamer high level API. This API does not give the programmer real-time access to the low-level media-processing functions; instead, it allows him/her to configure and manipulate a set of high level objects that encapsulate the main media functions such as players, processors, data sinks, and so on, to build the desired media-handling scenario in a Java application.

#### 4.1.1.7 Gstreamer-Java

Gstreamer-Java is a Java interface to the Gstreamer framework. Although Gstreamer is commonly associated with the gnome desktop, Gstreamer itself, and these bindings are portable across operating systems [24].

Having considered the above media platforms, some experimentation was carried out by building some example systems to compare audio and/or video streaming support. Gstreamer (accessed through Gstreamer-Java wrapper) was chosen to replace JMF media API for receiving, decoding and displaying multimedia content because:



- It supports most major audio and video codecs.
- It is still under active development, which guarantees support.
- Its Java bindings are portable across operating systems.
- It can be extended to support additional media types and perform custom processing.
- It defines a relatively simple RTP API that enables the transmission and reception of RTP streams.
- Gstreamer uses Video4Linux/V4L in Linux and Direct-draw in Windows for video capture. Both support several USB webcams, TV tuners and other devices. Additionally, these APIs are closely integrated into the respective kernels of their operating systems making them more efficient.

It is worth noting that preference was given to APIs either implemented in Java or that having existing interfaces to Java. The preference was motivated by the fact that one of our objectives was to produce a client that is platform agnostic.

### 4.1.2 Signalling APIs

Signalling APIs abstract several key components for session setup, control and termination. As earlier mentioned there are a several APIs that implement the SIP protocol. The Java Community Process (JCP) through the Java APIs for Integrated Networks (JAIN) initiative, defines APIs for using Java technologies to provide next generation telecommunications services. In the following subsections we discuss two APIs developed under JCP and JAIN initiative that support SIP programming for call control and messaging.

#### 4.1.2.1 JAIN SIP API

JAIN SIP is a Java API specification for SIP specified in JSR 032 under the JCP developed for the J2SE environment. It is a Java standard for a low-level SIP interface that provides access to SIP at the SIP protocol level. It provides application developers with a standardised interface for SIP services that are functionally compatible with the RFC 3261 specification. JAIN SIP, being low level, gives access to the full power in the SIP protocol and enables the creation of SIP applications of any type. More specifically, JAIN SIP API provides the application developer with an interface to:

- Build and parse SIP messages.
- Use the transaction sub-layer (i.e., send/receive messages statefully).
- Use the transport sublayer (i.e., send/receive messages statelessly).

To date, there have been three versions of the JAIN SIP API. The first version (1.0) was based on SIP specification RFC 2543 [27]. As we already know, that SIP specification was replaced by RFC 3261 [54]. So, a newer version (1.1) of the JAIN SIP API, which had compliance for RFC 3261, was developed. The latest JAIN SIP version is 1.2. It incorporates some enhancements to the 1.1 specification, and it is the one that was used in this thesis. Version 1.2 of the JAIN SIP specification complies with the base SIP specification defined in RFC 3261 and with the following SIP extensions as earlier discussed:

- INFO method [RFC 2976]
- Reliability of provisional responses [RFC 3262]
- Event Notification Framework [RFC 3265]
- UPDATE method [RFC 3311]
- Reason header [RFC 3326]
- MESSAGE method [RFC 3428]
- REFER method [RFC 3515]
- Distributing Authoritative Name Servers via Shared Unicast Addresses [RFC 3258]
- PUBLISH method [RFC3903]

It is worth mentioning that there are some open-source reference implementations for the JAIN SIP API. The NIST (National Institute of Standards and Technology) reference implementation of the JAIN SIP API is the one that was used to develop the SIP/IMS client discussed in this thesis because it has support for the IMS extensions discussed in chapter 2.

#### 4.1.2.2 JAIN SDP API

As we have seen, IP communication applications that use SIP will in many cases need to describe sessions using Session Description Protocol (SDP). Such descriptions are transported as part of the SIP message payload. From the developer's perspective, there is a need then to be able to encode and parse SDP content. There are a number of different ways to do this. One possible way to accomplish this is by using an implementation of the JAIN SDP API. JAIN SDP API defines a Java interface to facilitate the manipulation of SDP content. JAIN SDP is part of the Java network API family to which JAIN SIP is a member. JAIN SDP API corresponds to JSR 141, but at the time of writing was not yet an approved standard [45]. Given that we are using JAIN SIP API it seemed appropriate to embrace JAIN SDP for our SDP programming. JAIN SDP is a very simple API that just allows us to encode and decode SDP content.

#### 4.1.3 Mobicents XCAP API

The Mobicents XCAP client API provides a means to send XCAP requests to an XCAP Server such as the Mobicents XDM Server. The Mobicents XCAP client API depends on Java HTTP client API to provide the core HTTP functionality and Java HTTP client core API to provide low level HTTP transport components for building services with a minimal footprint.

## 4.2 Other Tools Used for Developing the Client

A large number of libraries and software are available for use by developers to develop rich applications for IMS. The Java Community Process (JCP) through the Java APIs for Integrated Networks (JAIN) initiative, define APIs for using Java technologies to provide next generation telecommunications services [41]. In this section we will discuss some of the key tools used to facilitate development of the RUCRG IMS client.

### 4.2.1 JDK 1.6

The JDK consists of a Java compiler, written in Java, and a run-time interpreter for a particular platform. It can be downloaded and installed free from the Oracle website. For the development of the IMS client, in this thesis, JDK 1.6 was used.

### 4.2.2 Netbeans 6.8

Netbeans 6.8 Integrated Development Environment (IDE) was used to develop and manage the project due to the abundance of its productivity features, such as team collaboration, context-sensitive code editors, debugging and project management. Furthermore, Netbeans IDE can be extended with various plug-ins order to provide a more complete IDE experience to developers.

## 4.3 Summary

This chapter provided descriptions of some of the open source media APIs for multimedia service creation that were investigated as possible alternatives to JMF. Among the projects investigated were a variety of implementations building on top of JMF such as FMJ.

# Chapter 5

## Enhancing Signalling and Media Support in JSAP

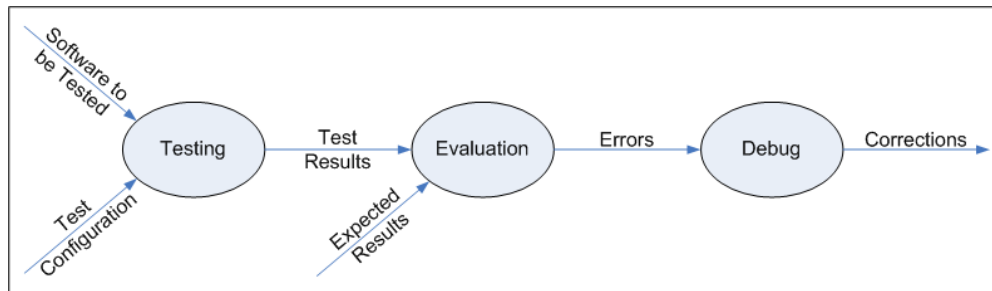
This chapter describes the various enhancements that were made to the JSAP client in preparation of making it IMS compliant. The first section discusses how JSAP was tested to identify errors and how the errors were fixed. The section that follows describes the SIP extensions that were incorporated into the JSAP to consolidate the signalling. Finally, a discussion is given of how the media library was overhauled to use Gstreamer media library to ensure reliable voice and video support.

### 5.1 Preliminary Analysis

Before new features could be integrated, it was necessary to validate and verify against the standards the already implemented SIP features in the JSAP. On the one hand, this was done to reinforce the understanding of how JSAP worked. On the other, this exercise was carried out with the intention of limiting the occurrence of unexpected behaviour by the client, caused by defects inherited from the original implementation.

The preliminary analysis entailed the systematic testing of the JSAP. The analysis summarised in Figure 5.1 involved running tests based on varying configuration of inputs and taking appropriate action following the evaluation of the test results.

Figure 5.1: Testing Information Flow (Adapted from Luo [36])



In Figure 5.1, “Software to be Tested” refers to the JSAP client. The “Test Configuration” includes test cases, test plan and procedures. The evaluation compared the results that were produced by running JSAP to expected outputs. For example, incorrect data was intentionally introduced (fault injection) in order to assess how the client behaved if users accidentally entered unexpected input.

It is worth mentioning that the testing was not meant to identify all the defects within JSAP. Instead, it was done to establish that the client functions properly/improperly under specific conditions. Below is a discussion of the various problems that were identified in JSAP and how they were resolved.

### 5.1.1 Unhandled Server Responses

When a user enters incorrect authentication parameters, there is need for him/her to be prompted to re-enter their credentials. However, JSAP did not have a mechanism to handle “403 Forbidden” responses sent back by the server. The user interface (UI) would remain unchanged with the user unaware of what was happening in the background, while the server waited for a response. This error was corrected by adding a method to handle “403 Forbidden” responses. This method calls the authentication GUI which prompts the user to reenter the credentials.

### 5.1.2 Password re-prompting

In the case that the user managed to enter the correct authentication parameters, when the registration expired, the user would be prompted again for their credentials. Furthermore, sending any request through an authenticated proxy meant that the user had to enter their authentication parameters for every request that passed through the proxy.

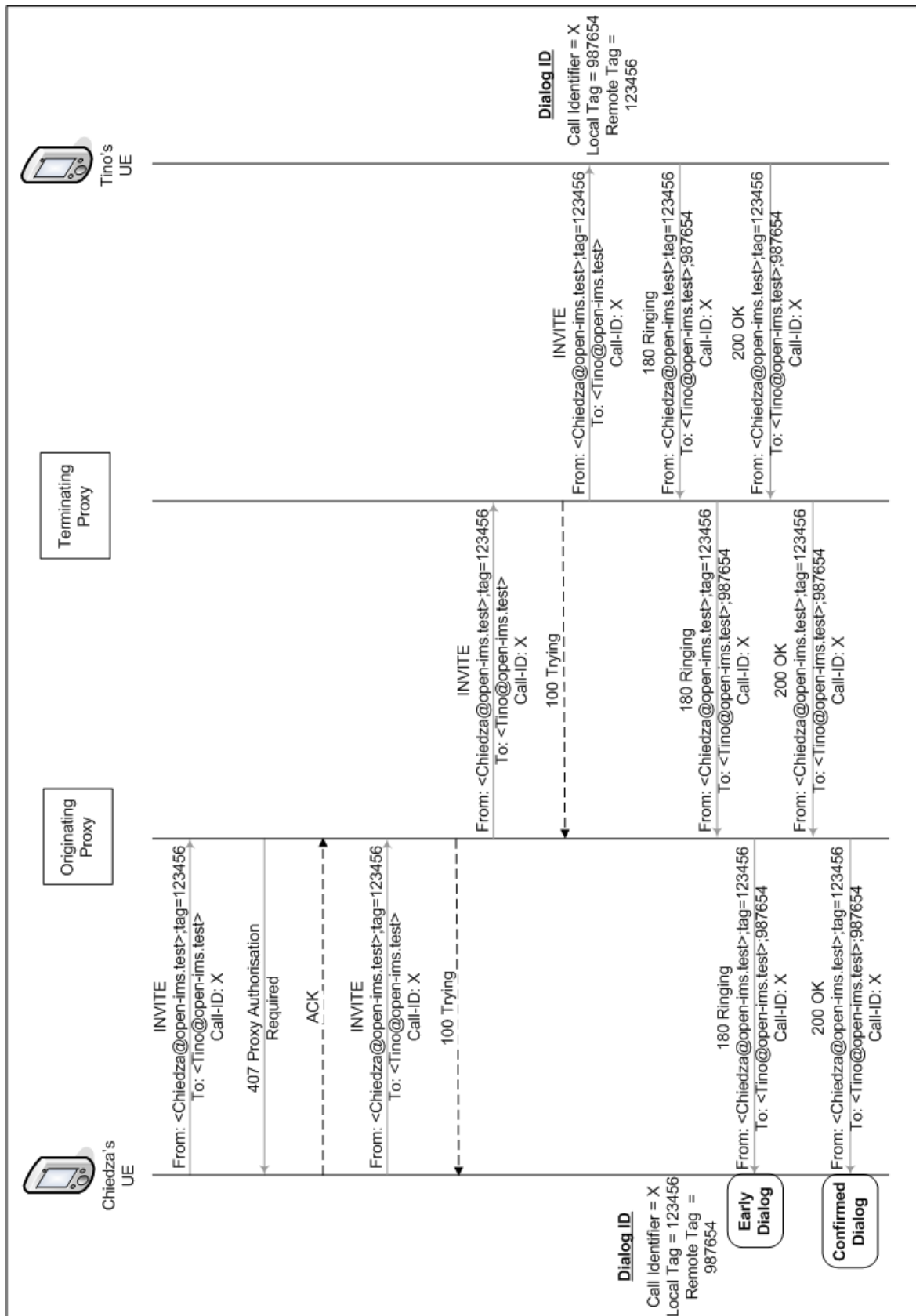
This was corrected by caching the correct credentials and only prompting the user to enter new credentials if the ones initially provided were not accepted by the authenticating

server. New classes were introduced: `UserCredentials` class for collecting the user-name and password and storing them; `CredentialsCache` class for caching credentials throughout the duration of the session such that they can be used for other requests within the transaction that require authentication without further prompting the user for credentials. `CredentialsCache` verifies whether the credentials have been used before for a particular transaction. If so, it checks whether they have been successfully authenticated before forwarding the result to `SipSecurityManager` class which actually handles the authentication process. `CredentialsCache` obtains the credentials from the `UserCredentials` object and only requests the user to enter new credentials if the entered information is incorrect.

### 5.1.3 Cancelling Early Sessions

SIP Dialogs are created through the generation of 2xx or 1xx responses to INVITE requests. Figure 5.2 illustrates how a Dialog is established between two parties in the process of establishing a call session.

Figure 5.2: Dialog Creation





Tags in the *From* and *To* headers together with the *CallID* are used to identify a Dialog. The tag in the *From* header is set by the calling UA and provides only half of the Dialog identification. The other half is set by the the recipient of the request (called party) by including a tag in the *To* header of the provisional and successful final responses. The called party (Tino in Figure 5.2) therefore establishes all the three parameters needed to identify the Dialog before the calling party (Chiedza in Figure 5.2). The result is that the Dialog is established by the called party before the calling party as shown in Figure 5.2.

With JSAP, the caller could not cancel a session while the called party was in ringing state, that is, when JSAP had received a “180 Ringing” response. This was because JSAP was setting the Dialog ID before receiving the remote tag. As such, the UA could not find the transaction (incorrect Dialog ID) to cancel when requested to do so because the remote tag was missing.

This was corrected by setting the Dialog ID after receiving the “180 Ringing” or “183 Session Progress” response from the remote client.

#### 5.1.4 Re-registration

Re-registration in JSAP was not functioning properly: once a user was de-registered, they could not re-register without restarting the client. This was due to the fact that the client logic incorrectly represented the registration state. When the client received a “200 OK” to a de-registration request it would correctly update the UI but not update the registration state to unregistered. The result was when the user attempted to re-register, the UI would not update because the client logic reflected that the user was still registered.

The solution was to update the registration status when client received a “200 OK” to a de-registration request.

#### 5.1.5 De-registration

Registrations are “soft state” meaning that they expire if they are not refreshed within a time interval specified by the registrar. However, a client can influence the expiration interval selected by the registrar. For example, registrations can be explicitly removed by the client by specifying an expiration interval of “0” for a contact address in a REGISTER request. RFC 3261 [54] specifies that all UAs *should* support this mechanism so that bindings can be removed before their expiration interval has passed.

From the testing that we carried out, it was discovered that the client was not updating the stored registration request as it refreshed its registration. This resulted in the client

sending a de-registration request with an incorrect sequence number to the registrar when it needed to remove the binding. This did not have a direct impact on the functioning of the client but it was violation of the SIP specification.

The solution was to save the most recent REGISTER request every time one was sent out. This would allow the client to send out the correct de-registration request when it needed to de-register.

## 5.2 Enhancements

Having attempted to fix the major defects in the JSAP, the next task was to identify the core functions that needed to be added to prepare the client for the addition of IMS capabilities. The enhancements were guided by:

1. The need to increase modularity in the JSAP in order to allow easy modification of the client.
2. The need to incorporate extensions to the base SIP protocol.
3. The need to overhaul JSAP media capabilities.

This section will outline how JSAP was modified to perform the following functions:

- Use XML to load its start-up configuration.
- Establish sessions using reliable mechanisms specified in RFC 3262 [53].
- Negotiate media codecs (static and dynamic).
- Exchange audio/video with another client using Gstreamer.

From now on, the new client resulting from this enhancement process will be called JSAP+.

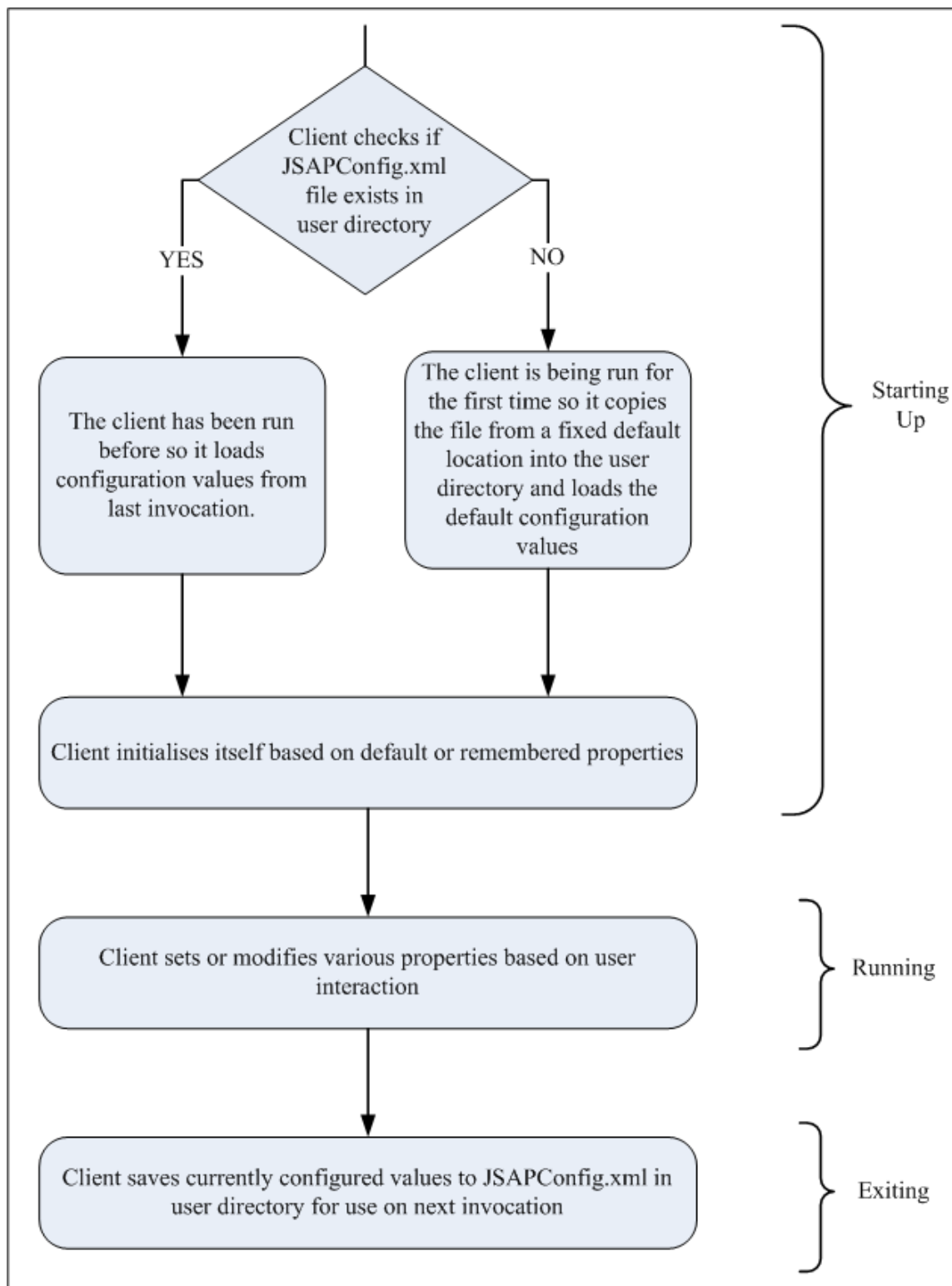
### 5.2.1 Configuration

Configuration parameters in JSAP were hard-coded in the `Configuration` class. For example, the IP address of the machine the client was being run on had to be known beforehand in order to be manually configured into the code. This made the client inflexible.

To overcome this issue and other inherent limitations, a configuration utility was developed to allow the client access its start-up context. An XML based configuration mechanism was introduced to allow for the persistence of the information that the user entered. New classes were also added to manage automatic configuration of the IP address. Changes were made to the JSAP to allow the configuration of client parameters at start-up using a `Properties` object and XML.

Figure 5.3 illustrates how the configuration parameters are managed in JSAP+ (the name we gave to the new, enhanced JSAP, as mentioned above) using the JSAPConfig.xml file, from the time the client starts up until the time it is closed.

Figure 5.3: Client Configuration Life Cycle



The configuration parameters are passed to the configuration file via a UI: the user now enters the client configuration parameters in the configuration area of the UI. At client start-up, or whenever the user presses the **Apply** button in the UI, the configuration parameters are conveyed as a `Configuration` object to `MessageListener` in the `userInput()` or `updateConfiguration()` methods. The `JSAPConfig.xml` file is just a data structure to hold the parameters entered by the user.

### 5.2.2 Reliability of Provisional Responses

SIP defines two types of responses: provisional and final. These responses are often sent over UDP, which means they can be lost. To deal with this problem, final responses are sent reliably. However, provisional responses are not sent reliably [54]. To increase the chances of provisional responses successfully reaching their destination, some of these provisional responses are retransmitted. However, there are cases where it would be important to guarantee the delivery of a provisional response. Examples of these cases are:

- Playing a message to a user that informs him/her that the call will be cancelled due to lack of funds in their account.
- Playing an announcement when a call is being forwarded or queued.
- When a provisional response contains an SDP answer as a result of an SDP offer sent in an INVITE request.

The cases mentioned above (and many others) rely on the reliability mechanism for setting up communication parameters before sessions are established. Hence, it was necessary to integrate support for reliability of provisional responses, into JSAP. Fortunately, SIP includes mechanisms that support the delivery of provisional responses reliably. In this section, we discuss how the reliability of provisional responses was implemented in JSAP. Our client acts as both a UAC and a UAS, so both scenarios will be discussed.

#### 5.2.2.1 UAC Behaviour

When a UAC creates a new request, it can insist on reliable delivery of provisional responses by inserting a *Require* header field with an option tag *100rel*. If the UAC does not wish to insist on using reliable provisional responses, but merely indicate that it supports them, a *Supported* header must be included in the request with the option tag *100rel* [53].

Our client is built to support both basic SIP and IMS. This means that it may encounter legacy SIP UEs that do not support the reliability extension. As a result it does not insist on the use of reliable provisional responses, allowing the remote device to decide whether it needs to use reliability mechanisms or not. Nokia [42] argues that the use of *Supported* header instead of *Require* header in the originating UE results in better interoperability with clients not supporting a particular SIP extension. Our client therefore includes a *Supported* header with a *100rel* option tag in all the INVITE requests that it sends out to indicate that it supports reliability mechanisms. It also includes an SDP offer with every INVITE request. If it receives a provisional response that contains a *Supported* or *Require* header field (containing a *100rel* option tag) for an initial INVITE request, it uses reliability mechanisms. However, this mechanism excludes “100 Trying” responses because of their hop-by-hop nature.

If a Dialog is not yet created (as discussed in sub-section 5.1.3), the UAC establishes it immediately after receiving a provisional response. It then creates a new request (PRACK) to acknowledge receipt of the provisional response. The PRACK request contains a *RAck* header field, which indicates the sequence number of the provisional response being acknowledged. This request is sent within the Dialog associated with the provisional response. Once the answer has been received, the UAC establishes the session based on the parameters of the offer and answer, even if the original INVITE has not been responded to.

If the UAC receives another reliable provisional response to the same request, and its *RSeq* value is not one higher than the value of the previous sequence number, the response will not be acknowledged with a PRACK. This means that the response will be discarded and will not be processed further by the UAC. Furthermore, our UAC does not acknowledge reliable provisional responses received after the final response.

#### 5.2.2.2 UAS Behaviour

When an INVITE request is received containing a *Supported* header or a *Require* header with a *100rel* option tag, the UAS sends any non-100 provisional response to the INVITE reliably. Each reliable provisional response is assigned a sequence number, carried in the *RSeq* header field. The *RSeq* value allows the UAS to keep track of the provisional responses the PRACKs are acknowledging. Provisional responses for different requests may use the same values for the *RSeq* number, as specified in RFC 3262 [53]. This is because the *RSeq* numbering space is within a single transaction.

If the INVITE contained an SDP offer, a “183 Session Progress” response containing an

SDP answer is sent back to the caller. Since the response is sent reliably, a *Require* header is included that contains the option tag *100rel*. When the PRACK request is received, it confirms delivery of the “183 Session progress” response that was sent. A “200 OK” response to the PRACK is then sent. If a PRACK request is received that does not match any unacknowledged reliable provisional response that was sent, the UAS responds to the PRACK with a “481 No Dialog Found” response. Furthermore, the UAS does not send a second reliable provisional response, until an acknowledgement is received for the first one.

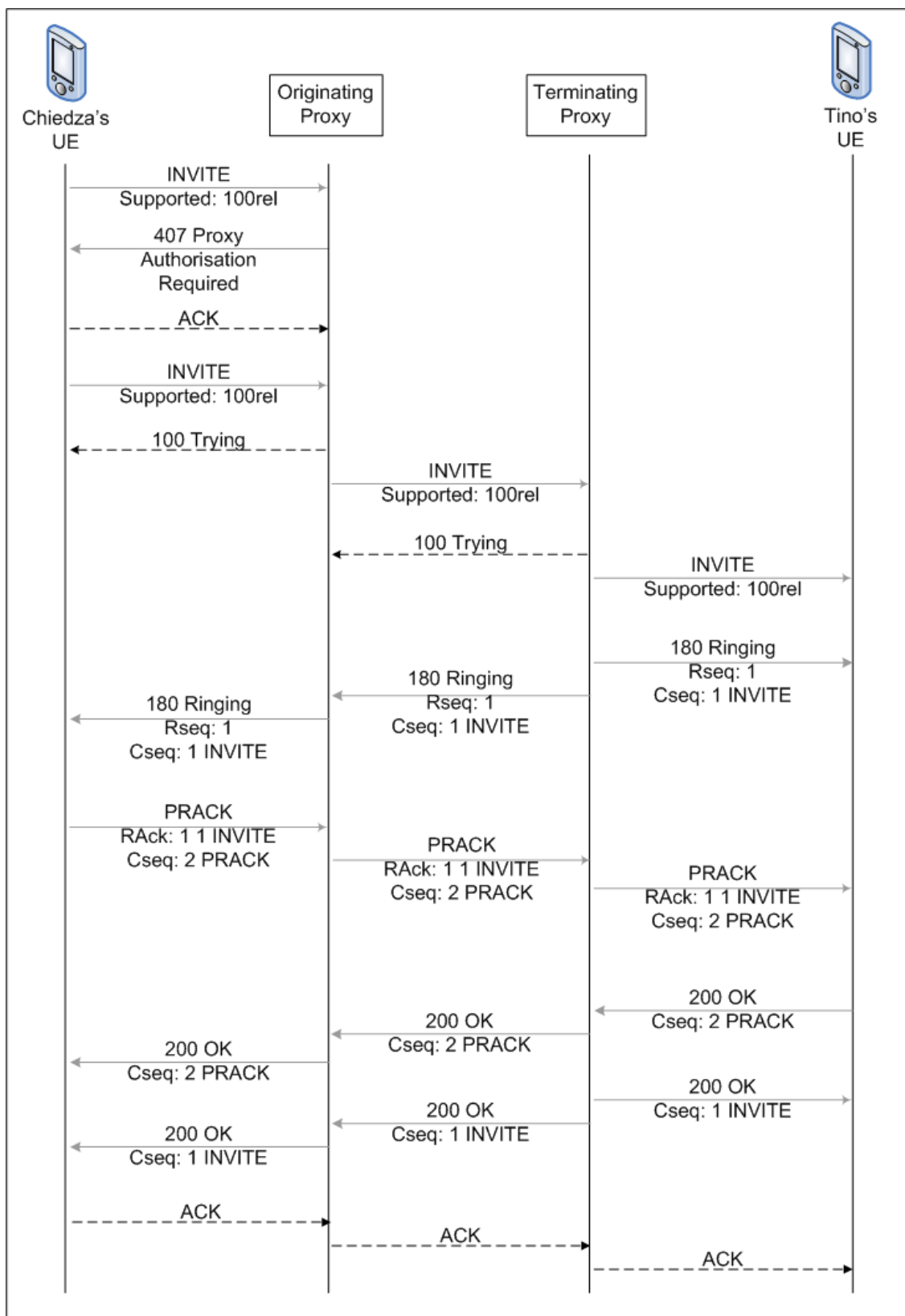
If an originating UAC sends us an INVITE request containing a *Supported* header or a *Require* header with a *100rel* option tag, but without an SDP offer, our first reliable provisional response will contain an SDP offer. The remote UAC will therefore receive a provisional response requiring the use of reliable provisional response mechanisms with an offer. The remote UAC must generate an answer in the PRACK.

Once the answer has been sent or received, the UAS establishes the session based on the parameters of the offer and answer, even if the original INVITE has not yet been responded to. However, media streams are only opened after a “200 OK” to the INVITE is received because our client does not support early media.

In cases where we include a session description in a reliable provisional response and the INVITE is accepted before the reliable provisional response is acknowledged, our UAS delays sending the 2xx until the provisional response is acknowledged. Otherwise, the reliability of the 1xx cannot be guaranteed.

Figure 5.4 provides a summary of how the reliability of provisional responses mechanism is handled in JSAP+.

Figure 5.4: Session setup Using Reliability Mechanisms





### 5.2.3 Session Description Handling

SDP defines the syntax for describing IP multimedia communication sessions. As already stated in chapter 4, JSAP uses the JAIN SDP API for encoding and parsing SDP content. The SDP support was however, not sufficient: there were a variety of functions that needed to be integrated to allow the client to handle a wider range of video and audio payloads. Firstly, JSAP was only capable of handling static payload formats. This meant that there was a need to integrate the ability to encode/decode attribute-lines in order to cater for dynamic payloads. Secondly, the client needed to be modified to describe multimedia sessions in a way that is compatible with the new media API (Gstreamer) that was introduced. The SDP encoding/decoding needed to be modified to suit Gstreamer. The idea was also to preserve as much of the original JSAP implementation as possible. In this section, we will look at how the client's SDP handling was modified to cater for the issues highlighted above.

#### 5.2.3.1 Dynamic Payload Coding and Decoding

The SDP media and transport line (**m-line**) includes information about a particular media. A session description may contain several “**m-lines**”, implying that the session may contain several media. Each “**m-line**” indicates:

- The type of media: voice, video, and so on.
- The port where the sender expects to receive media packets.
- The protocol to use for media transport.
- The media format.

The interpretation of the media format depends on the actual media transport protocol. When RTP/AVP is used, the media format represents the RTP payload type number. In chapter 2, we stated that the RTP payload number can be static or dynamic. If it is static, there exists a well-known ID number associated with it, so there is no need to include further information about the payload type in the SDP. Below is an example of an **m-line** with a static payload type (0), which indicates PCM  $\mu$ -law encoding for audio.

```
m = audio 40000 RTP/AVP 0
```

However, if the payload type is dynamic, there is no fixed ID associated with it. Dynamic payload types are randomly assigned numbers between 96 and 127. Since payload type numbers are dynamically assigned (as such, they change), extra information that characterises the format is required by the remote party receiving the SDP in order to be able to identify the payload type. Attributes (**a-lines**), are the primary means for providing such kind of information in SDP. They may be used as session-level attributes, media-level attributes, or both. In our case we were more interested in the media level attribute “**rtpmap**”, because we needed to provide support for non-static media codecs. The “**rtpmap**” attribute is used to map the payload type in an “**m-line**” with some parameters characterising the payload type, such as the encoding name, clock rate, or encoding parameters. The next example shows an “**m-line**” with a dynamic payload type.

```
m = audio 49230 RTP/AVP 961
a = rtpmap:96 L8/8000
```

In this case, there is an additional “**a-line**”, which is used to describe the encoding type (L8), encoding at a sampling rate of 8000Hz and dynamically assigned to payload type 96.

The SDP component that we build simplifies the task of setting or getting these pieces of information from an SDP message. The component is a Java class called `SdpManager`. We also used another Java class called `SdpInfo` which is a data structure that holds the value of the five parameters we are interested in. The `SdpManager` class offers two methods:

1. `byte[] createSdp(SdpInfo sdpinfo)`
2. `SdpInfo getSdp(byte [] sdpcontent)`

The first one receives as input an `SdpInfo` object, and creates as output a byte array representing the SDP content. The second one gets an SDP message as a byte array, and produces an `SdpInfo` object with the key info we are interested in. It is worth highlighting that the port and the media format parameters are obtained through a `Media` object, not directly through the `MediaDescription` object. So, in order to get these parameters, we had to:

1. Obtain the `MediaDescription` from the `SessionDescription`.
2. Obtain the `Media` object from the `MediaDescription`.

---

<sup>1</sup>Any number between 96 and 127 could have been used

3. Obtain the desired parameters from the `Media` object.

In chapter 2, we explained the way to describe multimedia sessions using SDP. RFC 3264 [52] describes the SDP offer/answer model. It also describes possible options to activate the media reception and transmission at the different stages in the model. The approach that we implemented was based on the following considerations:

1. The calling party sends the SDP offer.
2. The called party receives the offer and generates an answer. As soon as the SDP answer is sent, the answerer commences media transmission and starts listening on the receive ports specified in the SDP answer.
3. When the offerer receives the SDP answer, it starts listening on the receive ports that were specified in the SDP offer; and commences media transmission.

As was the case with reliability of provisional responses, our client acts as both UAC and UAS.

### 5.2.3.2 Sending the SDP Offer with Preferred Codec

In order to build the SDP offer, our client checks the media configuration parameters. If the configured media is audio only, then the SDP will contain only an audio “m-line”. If, on the other hand, it is audio and video, the SDP will contain an audio “m-line” and a video “m-line”. Also taken from the configuration parameters are the offered codecs: `myAudioCodec` and `myVideoCodec`. These are set by the user in the UI, and conveyed to `MessageListener` through the `Properties` object. The SDP will contain only one codec per media. The ports for audio and video are taken from the configuration parameters: `myAudioPort` and `myVideoPort`. These are also set by the user in the UI:

```
offerInfo = new SdpInfo();
offerInfo.setIpAddress(myIP);
offerInfo.setAudioPort(myAudioPort);
offerInfo.setAudioFormat(myAudioCodec);
offerInfo.setVideoPort(myVideoPort);
offerInfo.setVideoFormat = (myVideoCodec);
ContentTypeHeader contentTypeHeader = myHeaderFactory.createContent
TypeHeader("application", "sdp");
byte[] content = mySdpManager.createSdp(offerInfo);
myRequest.setContent(content, contentTypeHeader);
```

If the video component is not desired, `vPort` and `vformat` are set to `-1`, causing the `SdpManager` not to include the video “m-line” in the SDP.

### 5.2.3.3 Receiving the SDP Offer and Selecting Preferred Codec

When an INVITE is received that contains an SDP offer, the UA will get the SDP content and obtain the relevant parameters (ports and codecs):

```
byte[] cont = (byte[]) myRequest.getContent();
offerInfo = mySdpManager.getSdp(cont);
```

Having obtained the relevant information about the remote party’s preferences, we build the SDP answer as follows:

- The audio port in the answer is the configured port for audio (`myAudioPort`).
- The audio format in the answer is the same as the audio format in the offer.
- If the offer does not contain a video m-line, then the answer will not contain it either (`vport= -1`).
- If the offer contains video, but the recipient UA only wants audio then the video component is rejected (`vport=0`):

```
answerInfo.setIpAddress(myIP);
answerInfo.setAudioPort(myAudioPort);
answerInfo.setAudioFormat(offerInfo.getAudioFormat());
if(offerInfo.getVideoPort() == -1)
{
    answerInfo.setVideoPort(-1);
}
else if (myVideoPort() == -1)
{
    answerInfo.setVideoPort(0);
    answerInfo.setVideoFormat(offerInfo.getVideoFormat());
}
```

```
    }
    else
    {

        answerInfo.setVideoPort(myVideoPort);
        answerInfo.setVideoFormat(offerInfo.getVideoFormat());

    }
```

#### 5.2.3.4 Sending the SDP Answer

When the called party accepts the call, he/she issues a “200 OK” message that contains the SDP answer previously constructed. The called party will also start listening for media and will start transmitting media:

```
    ContentTypeHeader contentTypeHeader = myHeaderFactory.createContent
    TypeHeader(“application”, “sdp”);

    byte[] content = mySdpManager.createSdp(answerInfo);

    myResponse.setContent(content, contentTypeHeader);

    myVoiceTool.startMedia(offerInfo.getIpAddress(), offerInfo.getAudio
    Port(), answerInfo.getAudioPort(), offerInfo.getAudioFormat());

    if (answerInfo.getVideoPort()>0)
    {

        myVideoTool.startMedia(offerInfo.getIpAddress(), offerInfo.get
        VideoPort(), answerInfo.getVideoPort(), offerInfo.getVideo
        Format());

    }
```

#### 5.2.3.5 Receiving the SDP Answer

When the calling party receives the “200 OK”, he/she will start listening on the receive ports for the offered media. The client will also extract the SDP answer and begin transmitting media toward the address present in the answer. This is captured in the code snippet below:

```

byte[] cont =(byte[]) myResponse.getContent();
answerInfo = mySdpManager.getSdp(cont);
myVoiceTool.startMedia(answerInfo.getIpAddress(), answerInfo.getAudio
Port(), offerInfo.getAudioPort(), answerInfo.getAudioFormat());
if (answerInfo.getVideoPort()>0)
{
    myVideoTool.startMedia(answerInfo.getIpAddress(), answerInfo.get
VideoPort(), offerInfo.getVideoPort(), answerInfo.getVideoFormat());
}

```

RTP formats supported by Gstreamer were added to JSAP MediaManager. Table 5.1 shows some of the mappings from SDP to Gstreamer.

Table 5.1: SDPConstants mapping to Gstreamer formats

	SDPConstants	Gstreamer
Video	SdpConstants.H263	h263
	SdpConstants.JPEG	jpeg
	SdpConstants.H261	
	SdpConstants.MPV	mpv
	SdpConstants.MP2T	Mp2t
Audio	SdpConstants.G722	
	SdpConstants.G723	
	SdpConstants.GSM	gsm
	SdpConstants.PCMU	pcmu
	SdpConstants.DV14-8000	
	SdpConstants. DV14-16000	
	SdpConstants.PCMA	pcma
	SdpConstants.G728	
SdpConstants.G729	g729	

## 5.2.4 Media Plane

IP multimedia communications comprise of two planes: a signalling plane and a media plane. We have already dealt with how the signalling plane was modified in the JSAP to prepare everything that RTP needs, like determining the address where RTP packets need to be sent and negotiating the format that audio and video need to be encoded in.

In this section we will look into the media plane. At a minimum, a multimedia call requires some media-level handling at the endpoints in order to capture and present the media as well as receive and transmit the media packets over the network. This means, even in the most basic case, there is a need for the applications at the endpoints to have direct access to the media-handling capabilities of the user terminal.

As discussed in the previous chapter, the JSAP media plane functions were implemented using JMF, which was found to have a variety of shortcomings. An investigation into alternative, open source media APIs to the JMF was carried out. Among the projects investigated were a variety of implementations building on top of JMF such as FMJ. However, Gstreamer (accessed through Gstreamer-Java wrapper) was chosen to replace JMF because of the numerous advantages it possesses.

#### 5.2.4.1 Gstreamer Concepts

Gstreamer-Java is a Java interface to the Gstreamer API for handling time based media in Java applications [24]. It allows programmers to develop applications in Java to capture, present, store, and process time-based media. It can be extended to support additional media types and perform custom processing. Additionally, Gstreamer defines an RTP API to enable the transmission and reception of RTP streams. Gstreamer is a powerful yet easy API for building media/multimedia applications. The JSAP was redesigned to use Gstreamer as the media API (in place of JMF media API) to:

- Capture media and transmit over the network.
- Receive media over the network and render it to the user.

Gstreamer API defines several “elements” that model media processing [60]. Elements are an important class of objects in Gstreamer. By linking together different elements, a pipeline is created to perform a task such as media playback or capture. Below we introduce the main Gstreamer elements that were used to build our audio and video implementations in JSAP using Gstreamer.

- **Source:** is an element that encapsulates a media stream. During the media handling process, different data sources may represent the underlying media streams at different stages of the process, as shown in Figure 5.5, where the data source is represented as “src\*”. Every pipeline needs a data source to receive data. This can be a file or a network stream. In Figure 5.5 b, the `udpsrc` element is used to receive

UDP packets from the network while in Figure 5.5 a, the `udpsrc0` element performs a similar function.

- **Depayloader:** is needed to extract video/audio information from RTP packets. Depayloaders are specific to the video encoding used e.g. `rtph263depay` element is used to extract h263+ video from RTP packets.
- **Decoder:** is needed once the video packets are in usable form. The decoder processes the video packets back into a format that can be displayed (generally in the form of raw RGB or raw YUV video). FFmpeg H.263 video decoder (`ffdec_h263`) is an example of a decoder.
- **Filter:** is needed to convert video from one colorspace to another e.g. the `ffmpeg-colorspace` element. This was a necessary step when utilising the `xvimagesink` since a bug exists in the ATI driver which advertises a broken YV12 format. However, the YV12 format works on non ATI based machines regardless of whether they have the ATI graphics card or not.
- **Sink:** is an element that accepts data for storage/rendering. Disk writing, sound card playback and video output would all be implemented by sink elements. Sink elements do not produce any data. They only have a sink pad that accepts incoming data.

In order for an application to obtain instances of objects that represent the main Gstreamer elements (such as the ones discussed above), the application makes calls to the `gst` element factory.

#### 5.2.4.2 Implementation of the Media Plane

We have described the main elements of the Gstreamer API, and now we will show how the API was used to implement the following operations:

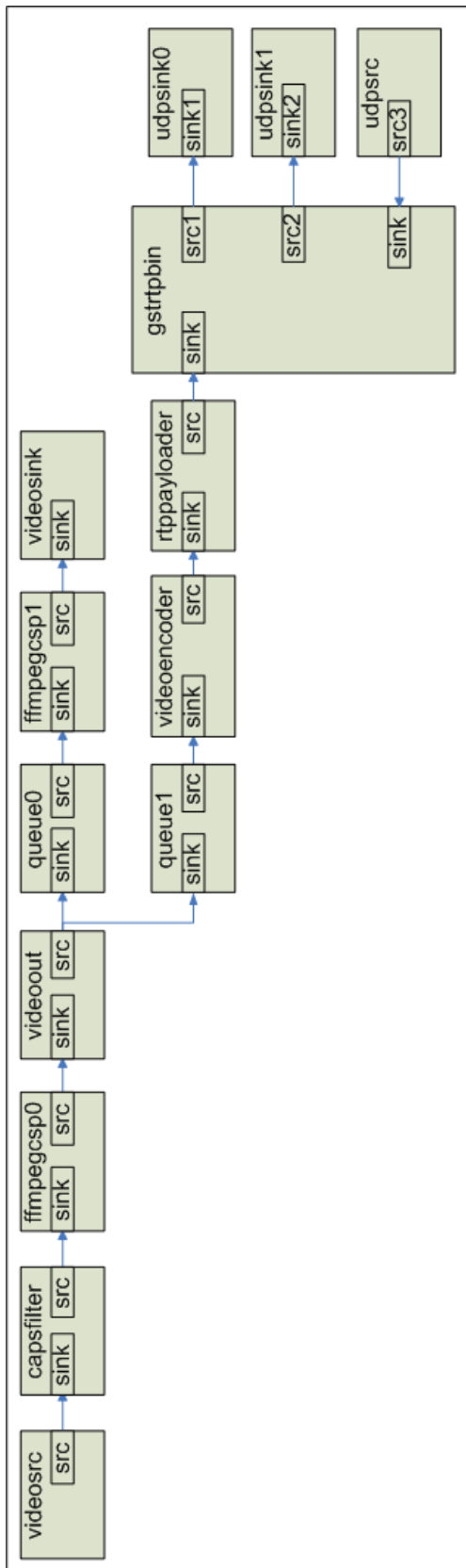
- Send media over the network.
- Receive media from network.
- Process the media.
- Present the media.

As an example, Figure 5.5 shows the various Gstreamer elements that were put together to create pipelines for sending and receiving live video over the network.

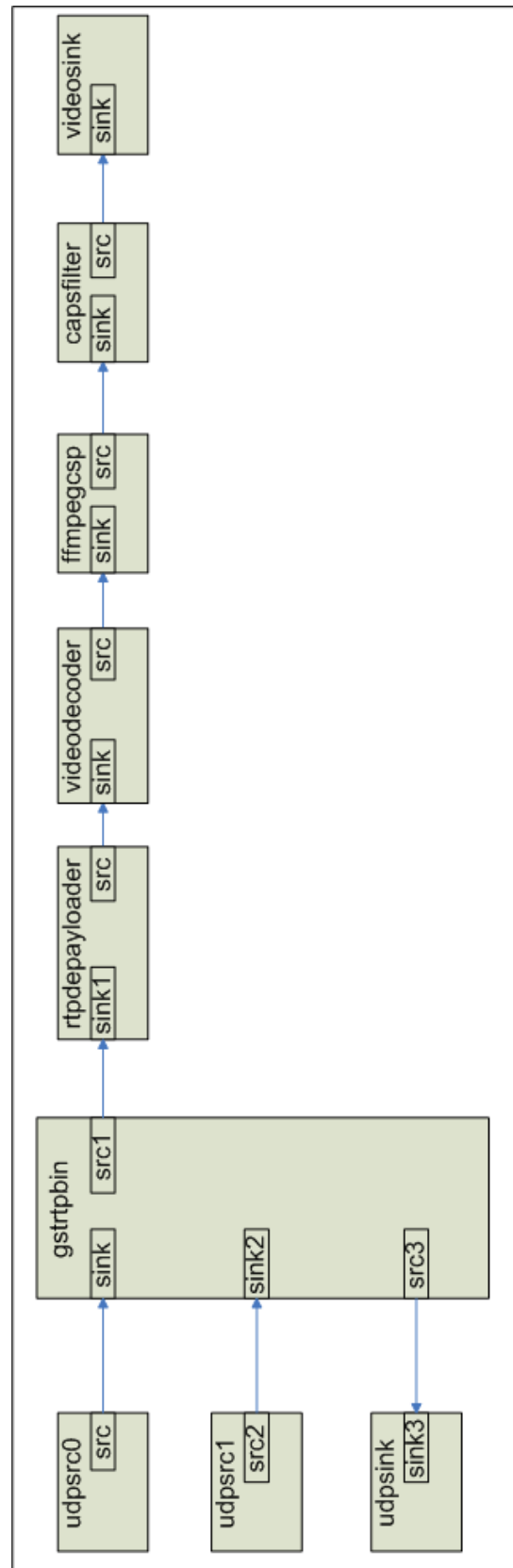


Figure 5.5: JSAP Gstreamer Video Pipelines

(a) JSAP Gstreamer Video Server Pipeline



(b) JSAP Gstreamer Video Client Pipeline



*Pads* are the element's input and output represented in the figure by “sink\*” and “src\*”. They are used to negotiate links and data flow between elements: these are the points where one can connect other elements. Pads have specific data handling capabilities, thus they can restrict the type of data that flows through it.

The `gst RTPbin` represents an entity that is used to manage and coordinate an RTP session. It keeps track of the participants in the media session and keeps track of the media being transmitted. It also handles the RTCP control channel. Thus, it offers methods to:

- Start and close an RTP session.
- Create RTP streams to be sent (in case we are transmitting the media).
- Add and remove peers.
- Obtain session statistics.

The `XOverlay` interface was used to solve the problem of embedding video streams in an application window. The application provides an `XWindow` to the element implementing this interface to draw on and the element will then use this `XWindow` rather than creating a new top level window. This can be useful to embed video in video players. This interface is implemented by the `Video4linux` and `Video4linux2` elements and by `ximagesink`, `xvimagesink`, and `sdlvideosink`.

### Ending Session

We also needed to stop media transmission and reception as soon as a BYE request was sent or received. If the client is in an established state and it receives a BYE request, then we need to add the following code:

```
myVoiceTool.stopMedia();
if (answerInfo.getVideoPort()>0)
{
    myVideoTool.stopMedia();
}
```

Similarly, when the client is in an established state and the user presses the `Stop` button, the UA will send a BYE request. We also included the above code to ensure that the media transmission and reception is stopped.

## 5.3 Summary

This chapter looked at how the various errors in JSAP were identified and remedied. Then it detailed the enhancements made with regard to signalling and media. The resulting client was renamed JSAP+ and will be referred to as JSAP+ in the rest of this thesis. In the next chapter we turn our attention to presence support.

# Chapter 6

## Improving Presence Support

Presence technologies are becoming widespread. For example, many SIP-based multimedia applications are now offering real-time communication services integrated with presence. Thus our client needed to be able to handle presence. The previous chapter discussed extensively the enhancements made to the signalling and media in JSAP resulting in JSAP+. In this chapter, we discuss how we integrated XCAP into JSAP+ in order to provide better presence support. The client resulting from this work was renamed JSAP++.

### 6.1 Presence

Presence allows users to publish their communication statuses, to indicate their availability and willingness to communicate. For instance, Chiedza's presence information might tell us that she is not connected, or that she is connected but in a meeting and cannot accept communications. Apart from this traditional use of presence, extended presence allows additional information (dynamic attributes) of individuals and devices to be provided. For example, if Chiedza's client supports extended presence, additional information may be provided about her mood, location and communication capabilities (depending on the device through which she is currently connected). Furthermore, presence information as a service enabler can be incorporated into any number of services, such as IPTV and presence enabled address books [38].

Presence has since been adopted for on-line collaboration, within enterprises as well as by service providers through the adoption of the IMS. A number of emerging applications and recent research efforts have benefited from presence technologies, particularly context

aware applications. Lei and Coulton [33] argue that SIP based presence will be a key enabler for achieving the envisaged rich multimedia experience within the IMS.

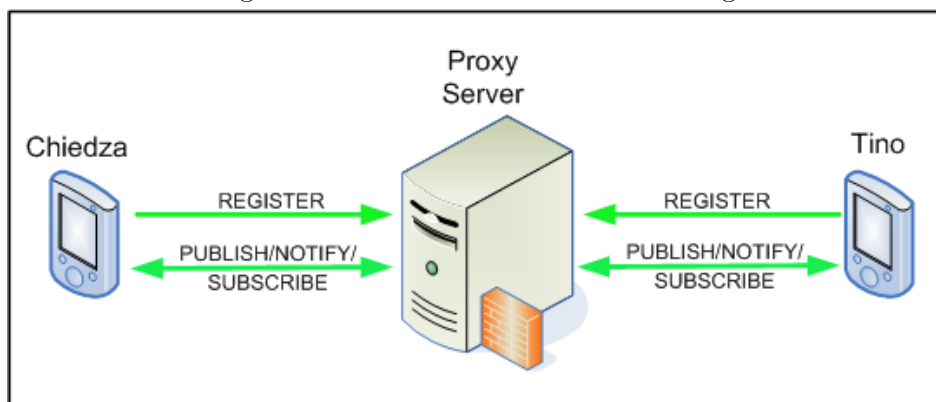
## 6.2 JSAP+ Presence

JSAP+ supported some form of SIP based presence which was integrated with IM. Similar to what we did for signalling and media, there was need to perform an extensive assessment of the current presence architecture of JSAP+ in order to establish its adequacy for RUCRG research purposes. The assessment included:

- Tracing messages sent by the client using network analysis tools to check
  - their sequencing; and
  - that they were well formed.
- Reverse engineering to find out the relationship among the various classes.

Having gone through the aforementioned process, a structure of how the client handled presence was drawn up. Figure 6.1 gives an overview of the structure that resulted from the experiments that were carried out.

Figure 6.1: JSAP+ Presence Handling



JSAP+ used a peer-to-peer presence mechanism based on a rudimentary implementation of SIMPLE (session initiation protocol for instant messaging and presence leveraging extensions) [12, 50, 48] as illustrated in Figure 6.1. In this setup, a SIP presence server and a resource list server (RLS) are not used. The client is the one that stores and manages the list of users whose presence status is desired. The client uses this locally

managed list to subscribe to the presence events of other users through a SIP proxy. This means that at each log-on, the client has to fetch the presence list from a file that it stores on the computer which it is runs on.

Since JSAP+ stored and managed user data locally (lacked mechanisms to store user data in a central repository), the data could not be accessed when the user moved from one device to another. As such, we needed to extend the client to support network based storage. This section will outline how JSAP+ was modified to store and retrieve user data on the network using XCAP (an HTTP-based protocol that allows a client to manage this user data).

Because JSAP+ already supported some form of presence, we wanted to preserve what was potentially useful. We carried out a full analysis of JSAP+ classes that handled presence. The process involved:

- Identification of classes which needed to be modified, removed or replaced.
- Studying the structure of the identified classes.
- Adding helper classes for populating XCAP specific parameters.
- Removing and/or replacing some existing classes with optimised ones to allow (efficient) support of XCAP.

## 6.3 Network Storage of User Information

Frequently, presence based applications require some back-end infrastructure, to store user information [45]. When this information resides within the network, its management can be done from anywhere, through a multiplicity of devices and modalities, including the web, wireless handsets, or PC applications [51]. Examples of this type of information are access control lists in VoIP application servers, presence authorisation lists, resource lists, and so on.

For cases where this information is based on XML, the IETF has defined an HTTP-based protocol called XCAP, which allows a client to manage user data. XCAP [51] allows a client to read, write, and modify application configuration data stored in XML format on a server. XCAP maps XML document sub-trees and element attributes to HTTP URIs, so that these components can be directly accessed by HTTP [45]. XCAP resources are accessed using HTTP methods (GET to read, PUT to create or modify, DELETE to remove). The key to XCAP operation is that the protocol defines an algorithm for

constructing a URI that can be used to reference a component within an XML document. A component can be any element or attribute within the XML document.

### 6.3.1 Choice of Technology

An XCAP client API was required to provide a means to send XCAP requests to the XCAP Server. The Mobicents XCAP client API was chosen because it is a free and open-source Java library. This choice aligns with the RUCRG philosophy of producing applications that are free and open-source. The API also integrated well in the client since the client was originally developed in Java. Furthermore, the use of a Java API allowed us to retain the platform independence feature of the client.

It should however be noted that, the Mobicents XCAP client API is an incomplete implementation of the XCAP protocol. The mechanisms to forward XCAP server responses to the application have only been implemented for the GET operation but not for the other XCAP operations (PUT and DELETE). Despite this discrepancy, the client and the server interacted in a reliable way (operations were successfully executed on the XCAP server); hence we decided to use it. Another reason we used the Mobicents XCAP client API was that it was the only Java XCAP client API that we could find.

As a consequence of using a partially implemented API, we had to manually check the resource lists on the XCAP server after executing an operation to verify that it had been successfully executed. We also had to use the Mobicents XCAP server for testing all XCAP functions to avoid compatibility issues. Future work will therefore have to be done to complete and standardise this part of our work.

### 6.3.2 Integration of XCAP support

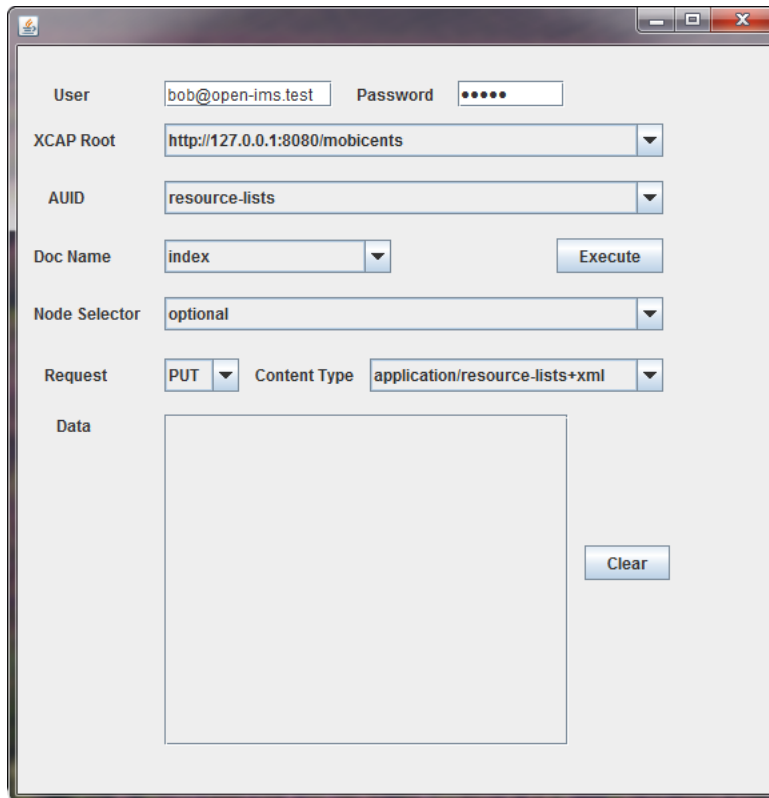
JSAP+ was extended to allow authentication with an XCAP server and to permit extracting, parsing and displaying of XCAP documents. A full discussion of how extensions were made follows.

#### 6.3.2.1 Authentication and Authorisation

XCAP has other functions apart from managing buddy lists. As such, researchers may need to manage other types of XCAP resources without needing to be registered with a SIP/IMS network. Fortunately, this is possible since SIP and XCAP are different protocols and they use different authentication and authorisation (AA) mechanisms.

An additional interface shown in Figure 6.2 was created to enable management of other types of XCAP resources and to collect the user's authentication parameters when they need to work with the XCAP server outside SIP/IMS.

Figure 6.2: RUCRG XCAP Client Interface



The screenshot shows a web-based interface for XCAP client operations. It includes the following elements:

- User:** Text input field containing 'bob@open-ims.test'.
- Password:** Password input field with masked characters (dots).
- XCAP Root:** Dropdown menu showing 'http://127.0.0.1:8080/mobicents'.
- AUID:** Dropdown menu showing 'resource-lists'.
- Doc Name:** Dropdown menu showing 'index', with an 'Execute' button positioned to its right.
- Node Selector:** Dropdown menu showing 'optional'.
- Request:** Dropdown menu showing 'PUT'.
- Content Type:** Dropdown menu showing 'application/resource-lists+xml'.
- Data:** A large, empty text area for entering request data.
- Clear:** A button located at the bottom right of the data area.

This UI allows advanced users to choose various options such as the XCAP server to use, the type of document, the document name as well as the request type. This UI also allows them to modify XCAP resources on the XCAP server without the having to go through a SIP/IMS network. The user can directly log-on to the XCAP server when they need to modify their XCAP resources. Furthermore, the UI allows the user to view responses from the XCAP server when the GET operation is invoked.

The Execute button has been placed in the middle of other options that the user can select because some XCAP requests only require the options before the button to be supplied in order to complete the XCAP operation.

### 6.3.2.2 Client Operations

- **Adding and Modifying**

Adding and modifying work in similar ways; they both use the HTTP PUT request



and the body of the request is never empty. Their specific behaviour depends on whether the URI that the client constructs refers to an existing resource or not.

In the case of adding or modifying a document, the client constructs a document URI that references the location where the document is to be placed. This URI contains the XCAP root and a document selector. The MIME content type is set to the type defined by the application usage. For example, it would be “application/resource-lists+xml” for a (RLS) services document. The XCAP server checks if the resource exists. If the URI resolves to an existing document, the new content replaces the content selected by the URI resulting in the modification of the contents. If not, the operation results in the addition of new content.

Adding or modifying elements and attributes works in a similar manner to adding or modifying documents. To create/replace an element (within an existing document) or an attribute (in an existing element of a document), the client constructs a URI whose document selector points to the document to be modified. A node selector is also added to the URI to help identify a single element or attribute. The MIME content types are set to “application/xcap-el+xml” and “application/xcap-att+xml” respectively.

An illustration of how a PUT operation is performed on a document is shown below:

```
PUT http://127.0.0.1:8080/mobicents/services/resource-lists/users/
Chiedza/friends.xml HTTP/1.1
```

```
Content-Type: application/resource-lists+xml
```

```
<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
">
<list name="friends" uri="sip:friends@open-ims.test" subscribable=
"true">
</list>
</resource-lists>
```

- **Retrieving**

This is accomplished by performing an HTTP GET request. In order to retrieve

a document, the client sets the request URI to the document URI. In the case of retrieving an element of a document, the client constructs a URI whose document selector points to the document containing the element to be fetched. The node selector identifies the element to be fetched.

To fetch an attribute in a document, the client constructs a URI whose document selector points to the document containing the attribute to be fetched. The node selector contains an expression identifying the attribute whose value is to be fetched. Retrieving is always followed by a “200 OK” response from the server if the request was successful.

In the case of retrieving an attribute, the “200 OK” response will contain an “application/xcap-att+xml” document with the specified attribute.

An illustration of how a GET operation is performed on an attribute is shown below:

```
GET http://127.0.0.1:8080/mobicents/services/resource-lists/users/
Chiedza/friends.xml?
resource-lists/list/list/entry[@name="Ruvarashe"]/@uri HTTP/1.1
```

*The server responds:*

```
HTTP/1.1 200 OK
Content-Type: application/xcap-att+xml
Content-Length: ...

sip:Ruvarashe@open-ims.test
```

- **Deleting**

Deleting is achieved by invoking an HTTP DELETE operation. To delete a document, the client constructs a URI that references the document to be deleted. In a similar way to creating or replacing a document, the URI is a document URI.

In the case of deleting an element from a document, the client constructs a URI whose document selector points to the document containing the element to delete.

The node selector identifies the element to be deleted.

To delete an attribute from the document, the client constructs a URI whose document selector points to the document containing the attribute to be deleted. The node selector evaluates to an attribute in the document to be deleted.

An illustration of how a DELETE operation is performed on an element is shown below

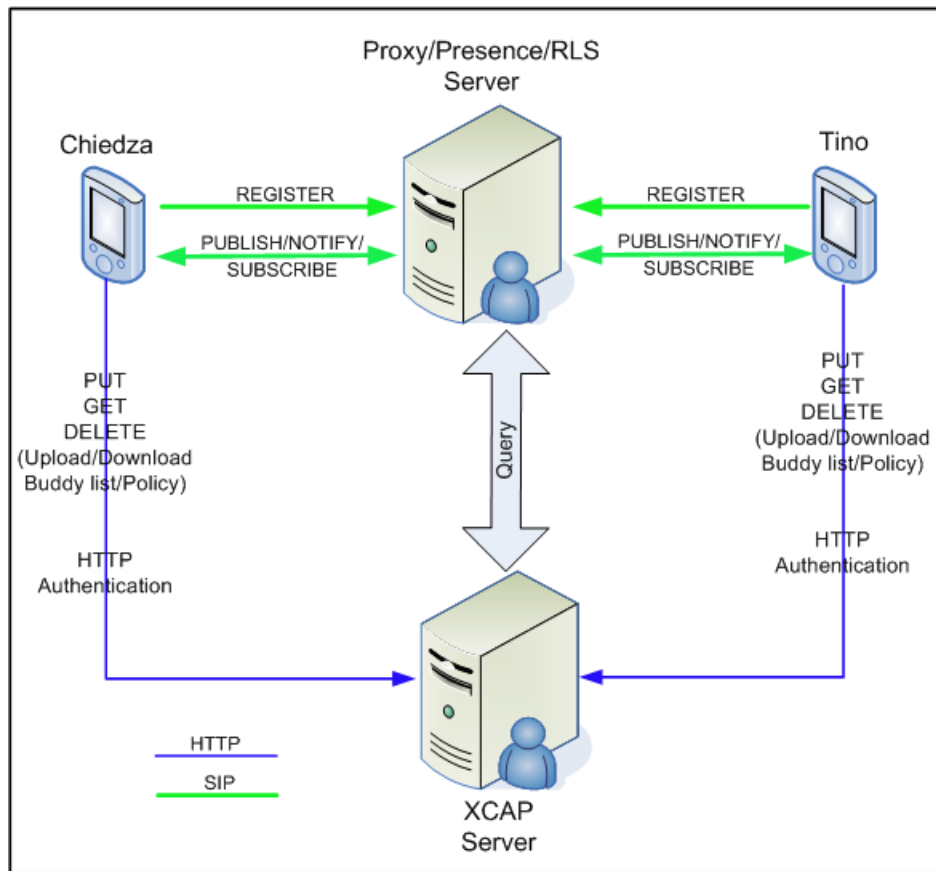
```
DELETE http://127.0.0.1:8080/mobicents/services/resource-lists/users/  
Chiedza/friends.xml?  
  
resource-lists/list/list/entry[@name="Tasara"] HTTP/1.1
```

The operations discussed above are invoked through the UI shown in Figure 6.2. The user supplies parameters through this interface and these are then used to construct the relevant XCAP requests that trigger the behaviour explained above. For example, if the user intends to delete a subscriber from a list on the XCAP server, they choose DELETE as the request method in the UI and fill in the relevant parameters pertaining to the user. When the Execute button is pressed, the XCAP client constructs a URI that points to the location of the specified subscriber and calls the XCAP DELETE operation on that URI. If the element exists, it is deleted.

### 6.3.3 Presence Lists

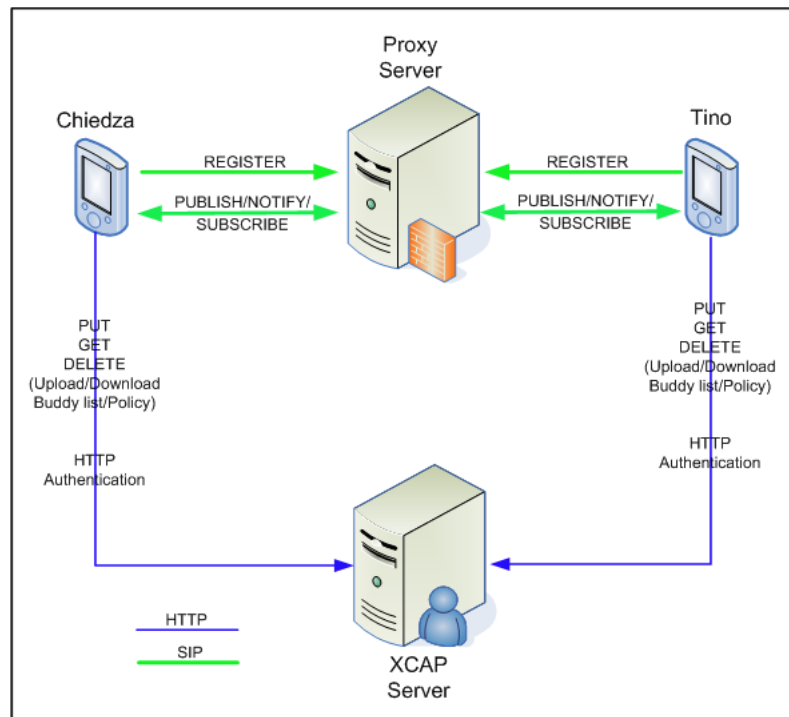
Presence lists are lists of users whose presence status is desired by a watcher. Presence authorisation policies define rules about which watcher is allowed to subscribe to which presentity, and what specific information they are allowed to access. There are several ways of accessing presence information for a list, but only two will be discussed. One way to obtain presence information for the list is to subscribe to a resource which represents that list. In this case, a RLS has to access this list in order to process a SIP SUBSCRIBE requesting it as shown in Figure 6.3.

Figure 6.3: Presence implementation using RLS with XCAP



Another way to obtain presence information for the users on the list is for a watcher to subscribe to each user individually. In this case, it is convenient to have a server to store the list: when the client boots, it fetches the list from the server. These two implementations allow a user to access their resource list from different clients, as shown in Figure 6.3 and Figure 6.4.

Figure 6.4: JSAP++ Presence implementation with XCAP

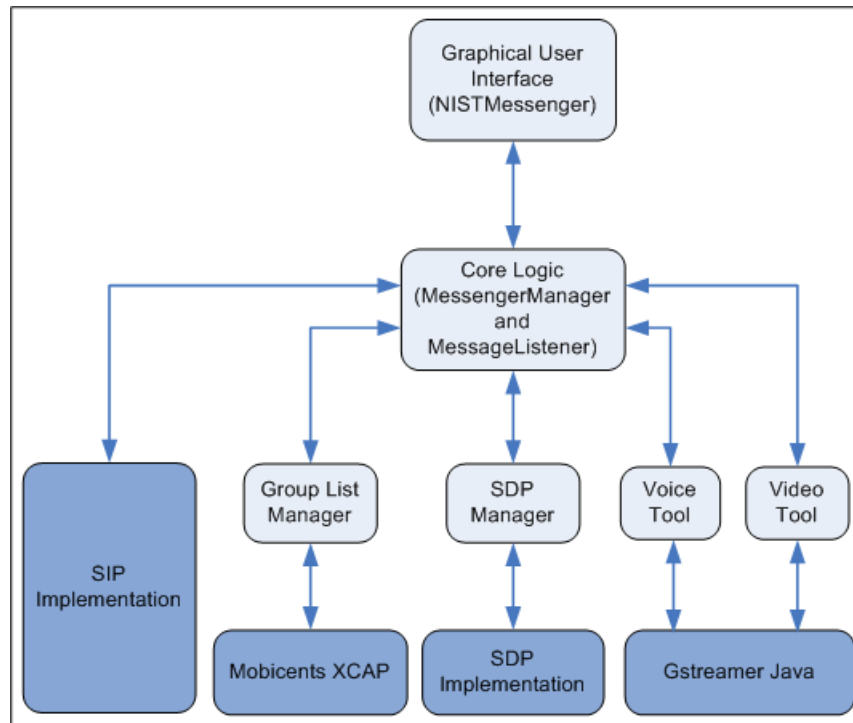


The model in Figure 6.4 is the one that we implemented in JSAP++ client because it was more aligned to the original implementation.

## 6.4 JSAP++ Architecture

Figure 6.5 shows a high level architectural snapshot of the JSAP++ client that resulted from the enhancements made to the JSAP.

Figure 6.5: JSAP++ Architecture



The client comprises the following components:

- User interface: it is implemented by the `NISTMessenger` class, and shows the graphical user interface that allows the user to interact with the client.
- Client application core logic: it is implemented by the `MessageListener` class. It consists of a finite state machine that receives events from GUI and from the SIP stack, and coordinates the execution of all the other components.
- SIP implementation: in our case, it is the SIP stack from NIST, which offers JAIN SIP 1.2 standard interface. It also provides the means to manage the presence information of the UE and associated contacts.
- Group list manager: implements procedures for retrieving, updating, and storing user data on the network.
- SDP manager: is a custom wrapper software layer that abstracts and simplifies the functionality in the JAIN SDP API for the purposes of our client application.
- SDP implementation: in our case, it is the SDP stack from NIST, which implements the JAIN SDP interface.

- **VoiceTool and VideoTool:** These are custom components which we created to offer simple APIs for capturing/presenting the voice or media streams respectively, and transmitting/receiving them over the network. They use the services of Gstreamer.
- **Gstreamer-Java implementation:** We used the Gstreamer-Java interface for all our media related handling. This piece of software implements the Gstreamer presentation API and the Gstreamer RTP API.

## 6.5 Summary

This chapter began by reviewing the way JSAP+ handled presence and identifying the areas where the client needed to be modified to allow for an improved presence model. We then provided context and discussed the work done to provide support for storing user data on the network, particularly relating to presence. This was achieved by extending the JSAP+ client to make use of XCAP for the storage and retrieval of user data.

In the next chapter, a detailed discussion on the design and development of the RUCRG IMS client will be given.

# Chapter 7

## Adding IMS Compliance

The previous chapter discussed how XML Configuration Access Protocol (XCAP) support was integrated into the JSAP+ client for the purposes of managing application configuration data. The result of this upgrade was an enhanced JSAP+ client that we called JSAP++. This chapter will provide details on how JSAP++ was transformed from a basic SIP client to an IMS compliant client which we now call the RUCRG IMS client.

### 7.1 Development Process

According to 3GPP IMS requirements, an IMS compliant end user device has to provide AKA (authentication and key agreement)v1/2-MD5 authentication, IMS SIP signalling support, basic voice and video, IM and presence [32].

Now that we had a working SIP client that could reliably register with a SIP registrar, setup and terminate SIP sessions using reliability mechanisms, send and receive voice and video using Gstreamer media API as well as manage application configuration data on the network using XCAP, the next step was to make the it IMS compliant. In order to make JSAP++ IMS compliant we needed to add the ability to:

- Register with an the IMS network, while preserving its ability to register with an ordinary SIP registrar.
- Establish IMS sessions while preserving its ability to setup sessions through ordinary SIP proxies.
- Negotiate media codecs during IMS session establishment using the SDP offer/answer mechanism.



- Cancel an early IMS session using the CANCEL method while preserving its ability to cancel ordinary SIP sessions.
- Exchange voice or voice and video with another IMS client.
- Terminate an IMS session using the BYE method while preserving its ability to terminate ordinary SIP sessions.

The following sections describe how these capabilities were integrated into JSAP++.

## 7.2 IMS Registration

As alluded to in chapter 2 (section 2.3.4), it is necessary for an IMS subscriber to be registered to his/her home IMS network in order to access IMS services. This meant that support for AKAv1/2-MD5 needed to be added to the JSAP++ client to make it IMS compliant. In this section, we discuss how AKAv1-MD5 was integrated into JSAP++.

To preserve the currently working MD5-based SIP authentication, we built IMS registration on top of the current SIP authentication mechanism. The MD5 infrastructure was reused. Only the AKA parameters and supporting methods that were not part of the MD5 scheme were added to the client. AKA, which was discussed earlier, is based on one time password generation mechanisms for HTTP Digest Authentication [40].

A shared secret (K) is established beforehand between the client/user and the authentication centre (AuC). The secret key is the password that the user enters when challenged to authenticate. A user registers with their home IMS network. If the user is not known to the domain, such a user will be unable to register.

As discussed in chapter 2, the P-CSCF serves as the initial SIP proxy into the IMS. Our IMS client (which we will also refer to as a UE, user equipment) has to send an initial REGISTER request to a P-CSCF that will forward requests on its behalf. This requires that the client establish the address of the P-CSCF before sending the request. Chapter 2 highlighted the different mechanisms that are used to determine what P-CSCF to use for sending requests. In our case, the user manually configures the IP address of the P-CSCF as shown in the configuration screen in Figure 7.12 (section 7.7) below. The user also configures in the GUI, the public and private identities (public identity is routing requests to a user while the private identity used for identifying the user's subscription and authentication purposes as explained in chapter 2).

The client programmatically establishes the IP address of the machine that it is being run on when it is started. It also programmatically configures the client and server ports. These ports will be included in the REGISTER message sent to the P-CSCF.

The user/subscriber sends a REGISTER request by selecting the Register menu item on the GUI. The UE also adds a *Via* header to record that the message has traversed the UE. The REGISTER message also includes the server and client ports. The message itself is sent on the standard SIP port (5060) unless a different port is explicitly specified. The REGISTER message also includes the private identity of the user. This identity will be used by the S-CSCF and HSS to identify the user.

The trace below shows the initial REGISTER request sent from the RUCRG IMS client to the P-CSCF:

```
REGISTER sip:146.xxx.xxx.xxx SIP/2.0
Call-ID: 6b0a822f0e25a6f60ed726d3d6d86b27@146.xxx.xxx.xxx
CSeq: 1 REGISTER
Max-Forwards: 70
Expires: 3600
User-Agent: RUCRG IMS Client Version 1
Contact: <sip:chiedza@146.xxx.xxx.xxx:5060;transport=udp>
Via: SIP/2.0/UDP 146.xxx.xxx.xxx:5060;branch=z9hG4bK71a3d2dd1089c2
2f489bb25d8112cbd2
Authorization: Digest response="",username="chiedza@open-ims.test",
nonce="",realm="open-ims.test",uri="sip:146.xxx.xxx.xxx",algorithm=
AKAv1-MD5
From: "chiedza" <sip:chiedza@open-ims.test>;tag=113324400
To: "chiedza" <sip:chiedza@open-ims.test>
Content-Length: 0
```

The sections highlighted in italics (in the traces) show the major differences between SIP and IMS messages. We will highlight these differences in the same manner for all subsequent traces.

When the P-CSCF receives the REGISTER message, it adds a *Via* header and removes the *Route* header. The REGISTER message will then be routed to the IP address specified in the request. The P-CSCF then forwards the request to the I-CSCF.

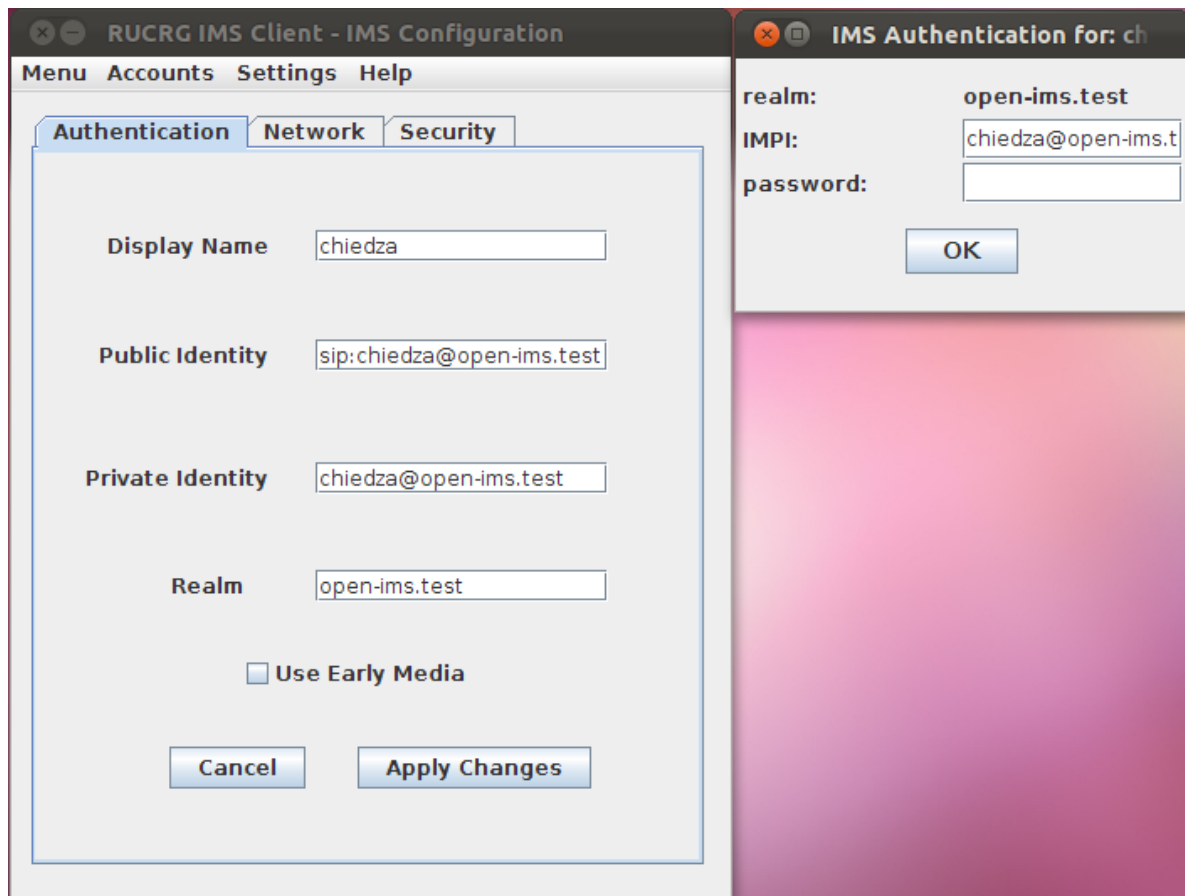
The I-CSCF queries the HSS to assign the S-CSCF and the HSS replies with a number of possible S-CSCFs. The I-CSCF then selects an S-CSCF based on the capabilities of S-CSCFs provided by the HSS. Once the S-CSCF assignment is completed, the I-CSCF forwards the REGISTER message to the selected S-CSCF.

The S-CSCF queries the HSS for the user's authentication details. The HSS passes the random number (RAND), authentication token (AUT), signed result (XRES), cipher key (CK) and integrity key (IK) to the S-CSCF. At this point we are not yet authenticated, so our registration request is rejected by the S-CSCF. The S-CSCF then challenges our UE with a "401 Unauthorized" response, which contains a nonce value, RAND, AUTN, CK and IK.

The "401 Unauthorized" message is passed to the P-CSCF, which in turn saves the CK and IK then removes them from the *WWW-Authenticate* header. These keys will be needed for establishing the IPsec security association. The P-CSCF then passes the nonce, RAND and AUTN values to the subscriber.

When the UE receives the "401 Unauthorized" response, it prompts the user to enter the K which the user and the AuC exchanged beforehand. Figure 7.1 shows the interface that the user is presented with in order to enter their credentials.

Figure 7.1: Authentication Screen



Using the  $K$  and a sequence number generated by the AuC of the home network after the first REGISTER request is received, the UE verifies the AUTN. If the verification is successful, the network has been authenticated by our client.

Using the nonce,  $K$  and RAND as input to the AKA algorithm, the UE then computes an authentication response (RES). The RES is sent to the S-CSCF in a second REGISTER request. This REGISTER message contains the RES in the *Authorization* header as shown in the trace for a second REGISTER request sent from the RUCRG IMS client:

```
REGISTER sip:146.xxx.xxx.xxx SIP/2.0
Call-ID: 6b0a822f0e25a6f60ed726d3d6d86b27@146.xxx.xxx.xxx
CSeq: 2 REGISTER
Max-Forwards: 70
Expires: 3600
P-Access-Network-Info: IEEE-802.11
```

```
User-Agent: RUCRG IMS Client Version 1
Supported: path
Contact: <sip:chiedza@146.xxx.xxx.xxx:5060;transport=udp>
Via: SIP/2.0/UDP 146.xxx.xxx.xxx:5060;branch=z9hG4bK71a3d2dd1089c2
2f489bb25d8112cbd2
Authorization: Digest response="a782123e7eeb8afedc44a1025acca6ce",
cnonce="88cec05a31ef9d62b459f0261f87139b",username="chiedza@open-
ims.test",auts="8As0A4UvPmSRpPTGLzQ=",nc=00000001,qop=auth-int,
nonce="Ed80Mh8tIo+QqSk6v5UtE6k06j77WAApFsbVQ29Hy0=",realm="open-
ims.test",uri="sip:146.xxx.xxx.xxx",algorithm=AKAv1-MD5
From: "chiedza" <sip:chiedza@open-ims.test>;tag=113324400
To: "chiedza" <sip:chiedza@open-ims.test>
Content-Length: 0
```

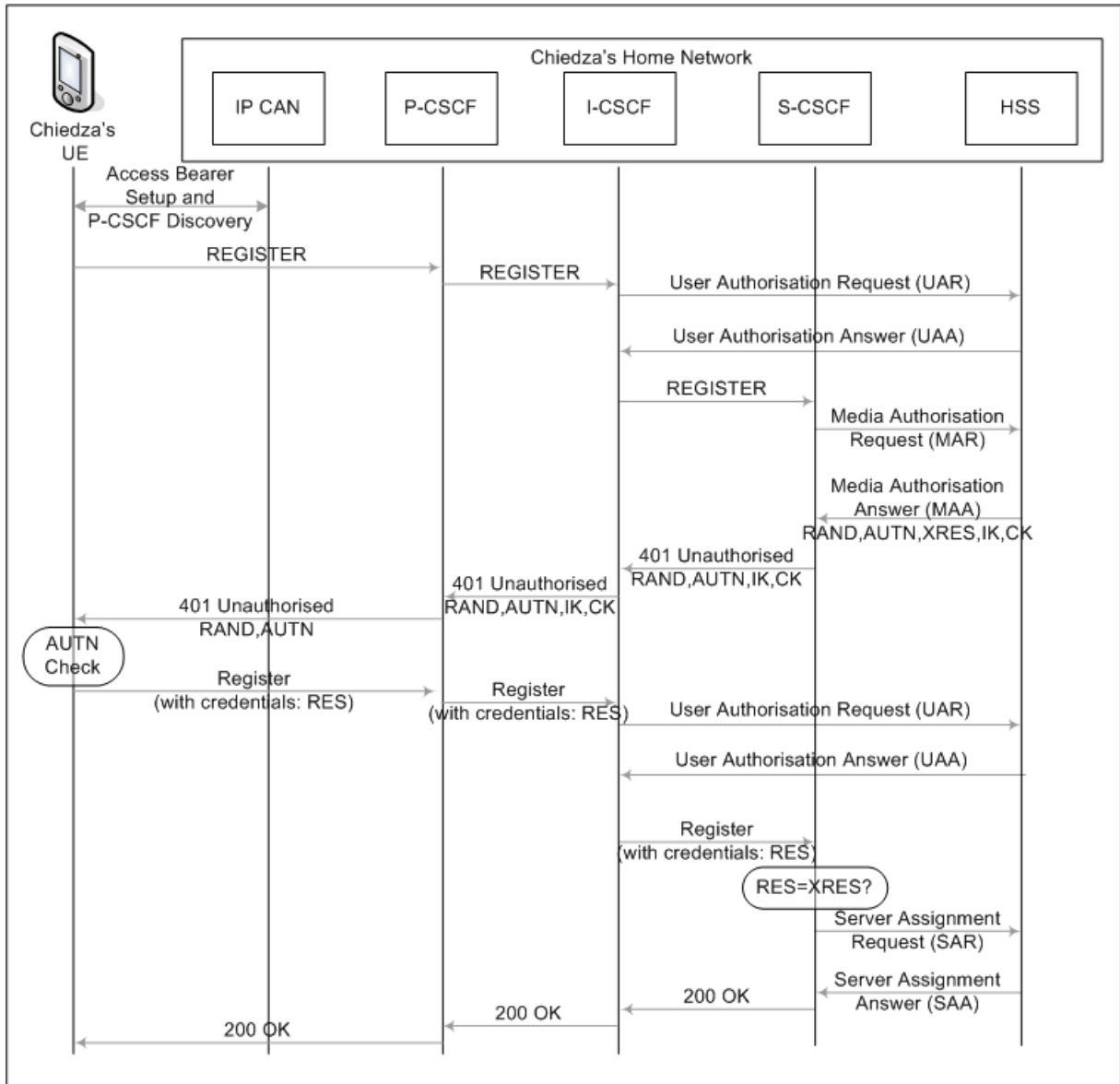
The RES is delivered to the server in the second REGISTER request. Upon receiving the request, the S-CSCF compares the RES with the XRES. If the two match, the S-CSCF registers the user's public identity and associates it with the client's IP address and port number. The S-CSCF replies with a "200 OK" message which is relayed back to the P-CSCF. This message serves to inform the user that they have been successfully registered on the network. The "200 OK" response to the REGISTER request also contains the *Service-Route* header. The *Service-Route* header conveys the name of the home service proxy (S-CSCF) where the UA must direct its requests. As soon as the UE receives this response, that is, the "200 OK" to the REGISTER, it stores the S-CSCF record and includes both the P-CSCF name and the S-CSCF name in the *Route* header of all outgoing requests [11]. Once the above steps have been performed successfully, the UE is ready to establish a SIP session to access IMS services. The IMS registration of the user is now complete.

The sequence diagram in Figure 7.2 summarises the IMS registration process followed by the RUCRG IMS client. The call flow shows Chiedza registering in her home network. The IMS registration goes through the following sequence:

1. IP address assignment to the client.
2. Unauthenticated IMS registration attempt: the client attempts an IMS registration but is challenged by the IMS network to authenticate itself.

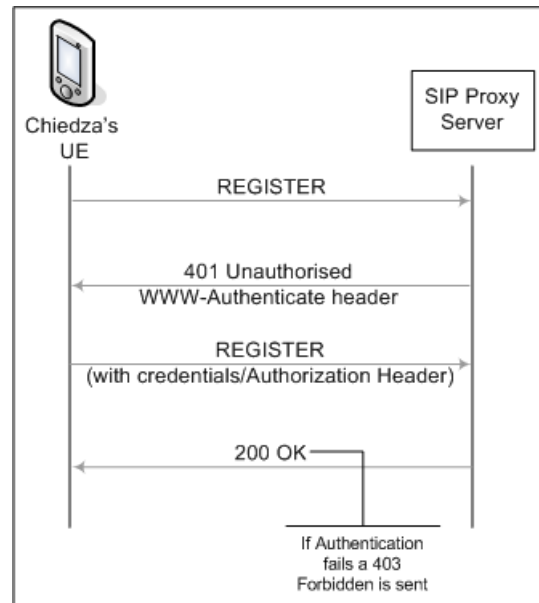
3. Authenticated IMS registration: registration is reattempted but this time the user is successfully authenticated and accepted.

Figure 7.2: IMS Registration (Non-roaming Case)



When this is compared to standard SIP registration procedure in Figure 7.3 the differences in the setup procedures are immediately evident.

Figure 7.3: SIP Registration

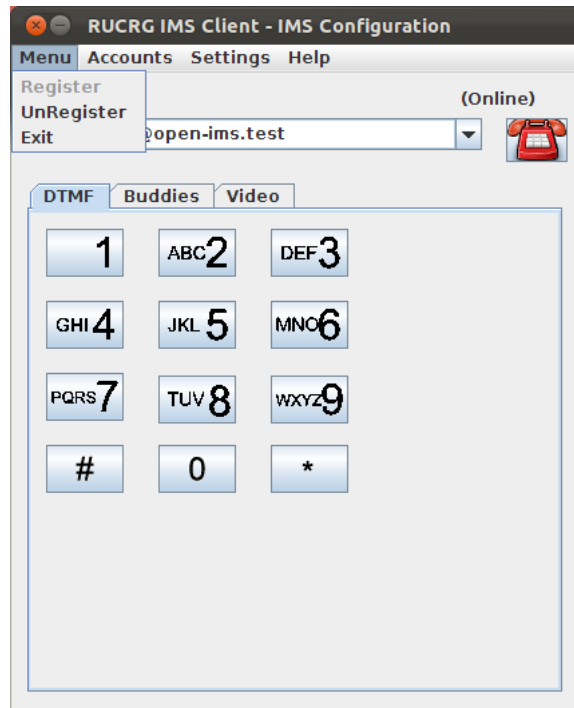


In order to avoid multiple registrations, the IMS client displays a progress screen showing a Please wait message while it communicates with the IMS network entities. If the IMS registrar or the whole IMS network is not running, the client will time-out after 30 seconds, and allow the user to try again.

## De-registration

If the user decides to de-register the client he/she can do so by selecting the UnRegister menu item as shown in Figure 7.4.

Figure 7.4: De-registration menu



When the UnRegister menu item is selected, a REGISTER request will be sent to the IMS network. This initial REGISTER request will be challenged by the network in a similar manner to a normal REGISTER request. However, when the client receives the “401 Unauthorized” response, the user is not prompted to enter their credentials since these were cached during registration. As a result the de-registration is completed in the background. The user will be presented with the Unregistered screen as soon as the “200 OK” response for the second REGISTER request is received.

The trace below shows the de-registration request sent from the RUCRG IMS client to the IMS network:

```
REGISTER sip:146.xxx.xxx.xxx
SIP/2.0 Call-ID: ee948510f0068dd4ea2b13d20d1d887a@146.xxx.xxx.xxx
Max-Forwards: 70
Expires: 0
P-Access-Network-Info: IEEE-802.11
User-Agent: RUCRG IMS Client Version 1
Supported: path
```



```
Contact: <sip:chiedza@146.xxx.xxx.xxx:5060;transport=udp>
CSeq: 4 REGISTER
Via: SIP/2.0/UDP 146.xxx.xxx.xxx:5060;branch=z9hG4bK74afcb1fb6d877
77dc71662af19d602f
Authorization: Digest response="c397ba38c6ae1278c78ba68ffcb283e3",
cnonce="88cec05a31ef9d62b459f0261f87139b",username="chiedza@open-
ims.test",auts="NbX+DUho2Y/CbrGjmmY=",nc=00000001,qop=auth-int,
nonce="/8XmN4COXSh8r5G755XIEA+TnZeAFQAAeibc6GKLR0g=",realm="open-
ims.test",uri="sip:146.xxx.xxx.xxx",algorithm=AKAv1-MD5
From: "chiedza" <sip:chiedza@open-ims.test>;tag=1429718272
To: "chiedza" <sip:chiedza@open-ims.test> Content-Length: 0
```

From the trace one can see that the REGISTER request for de-registration is similar to the one sent for registration. The only difference is the value of the *Expires* header, which is set to zero.

## 7.3 IMS Session Establishment

In this section, we discuss how IMS session establishment mechanisms were implemented in JSAP++ to transform it into the RUCRG IMS client. Like a SIP UA, an IMS client is made up of the user agent client (UAC) and the user agent server (UAS). The UAC generates the requests and processes the responses, while the UAS processes the requests and then generates the responses.

There are two issues that complicate IMS call setup procedures when compared with SIP call setup procedures:

1. IMS calls have quality of service (QoS) requirements.

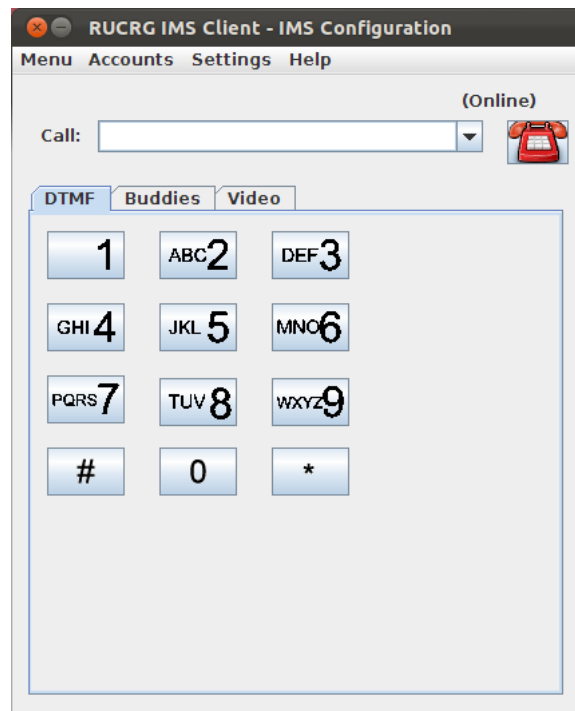
IMS strives to offer a quality of experience equal to, if not better than, traditional circuit-switched telephony [11]. Therefore, the 3GPP have stipulated a strict call setup procedure in 3GPP TS 23.218 [4] that ensures that both the originating and terminating access networks have provisioned adequate channels to ensure minimal delay and packet loss, which would negatively impact the media quality. What this implies is that the session is not established until the originating client's network and the remote client's network have provisioned the resources required for that call.

2. Reliability is mandatory in IMS calls.

It is a requirement that all IMS provisional responses are sent reliably. This is because provisional responses may contain SDP answers as a result of SDP offers sent in INVITE requests, as discussed in chapter 5. Thus, it is crucial for an SDP answer (to an SDP offer that was in an INVITE) to be in a reliable non-failure message in order to guarantee its delivery. Fortunately, the reliability mechanisms implemented in chapter 5 are applicable to IMS, save for the fact that they are effected by default.

After a user is successfully registered with their home IMS network (i.e. after receiving a “200-OK” response to the REGISTER request), the client displays a Ready-for-calls screen, to allow the user to establish a session with another IMS client. Figure 7.5 depicts the RUCRG IMS client display after a successful registration to IMS network.

Figure 7.5: Ready for calls



From this Ready-for-calls screen, there are two possibilities that can result in the establishment of a communication session:

1. The user can establish a session by entering a destination SIP URI, followed by pressing the telephone icon shown in the upper right hand corner.
2. The user can respond to an incoming call.

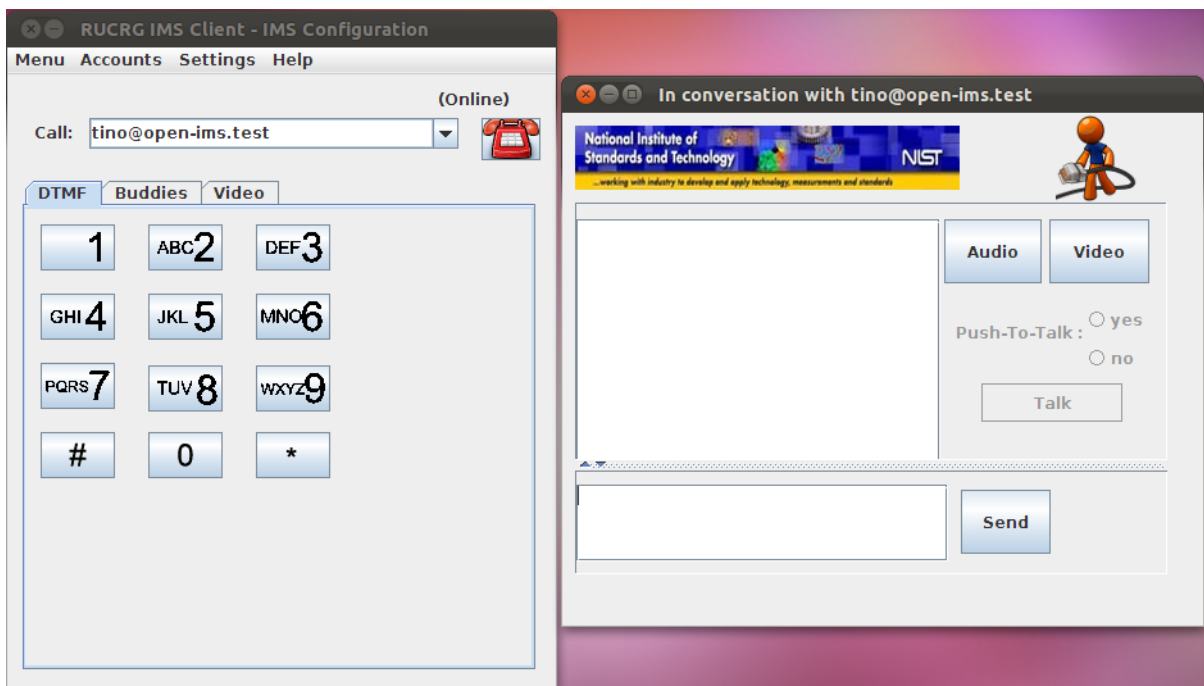
In other words, our client can take the role of a UAC or a UAS. Both scenarios will be discussed in the following sections.

### 7.3.1 UAC Behaviour

In this section, our client initiates the call and will be referred to as the caller's UE/the client.

When the telephone icon is pressed, the client displays the Invite screen, shown in Figure 7.6. This interface consists of two buttons which allow the user to make a choice between establishing an audio call or an audio/video call with the remote client.

Figure 7.6: Invite Screen



The client prepares a list of supported voice codecs or voice and video codecs depending on the choice made by the user. This information is included as the first SDP offer in the initial INVITE as discussed in chapter 5. The client sends the INVITE to the destination SIP URI selected by the user.

The trace below shows an INVITE request that was sent from the RUCRG IMS client (the user is Chiedza) to another IMS Client B (the user is Tino).

```
INVITE sip:tino@open-ims.test SIP/2.0
```

```
Call-ID: f81a412b16968b26f684d579b488f30b@146.xxx.xxx.xxx
CSeq: 1 INVITE
From: "chiedza" <sip:chiedza@open-ims.test>;tag=113324400
To: <sip:tino@open-ims.test>
Via: SIP/2.0/UDP 146.xxx.xxx.xxx:5060;branch=z9hG4bK0eecb09deaa5ad2cc
8f0689bfde894fc
Max-Forwards: 70
Contact: <sip:chiedza@146.xxx.xxx.xxx:5060;transport=udp>
Route: <sip:146.xxx.xxx.xxx:4060;transport=udp>, <sip:orig@scscf.open-
ims.test:6060;lr>
Allow: INVITE,ACK,CANCEL,BYE,MESSAGE,PRACK,UPDATE
P-Preferred-Identity: "chiedza" <sip:chiedza@open-ims.test>
Supported: 100rel,precondition
P-Access-Network-Info: IEEE-802.11
User-Agent: RUCRG IMS Client Version 1
Content-Type: application/sdp
Content-Length: 737
```

The message contains *Route* entries for the UE and the S-CSCF address that was extracted from the *Service-Route* header in the registration “200 OK” message. *To* and *From* headers are also included in the message. These headers do not play a role in call processing. In the above trace, the SDP part has been stripped from this message and will be discussed later.

The INVITE is received by the home P-CSCF, which verifies that the preferred public identity that we specified in the INVITE request is currently registered. The P-CSCF then queries the DNS to obtain the IP address of the S-CSCF in the called party’s home network. The INVITE request is relayed to the called party’s UE through the home IMS network and through the terminating IMS network elements.

When the called party’s UE receives the INVITE, it examines the codec(s) list in the SDP and prepares a list of codec(s) common to both UEs (caller and callee). The common codec(s) list is included in the “183 Session Progress” response sent by the called party’s UE responds. The “183 Session Progress” message retraces the path of the original INVITE. Each node that the “183 Session Progress” message traverses removes its own entry from

the *Via* header and forwards the message to the *via* entry at the top. As the message moves through the network, the *Record-Route* header remains unchanged.

When the caller's UE receives the "183 Session Progress" from the called party, it extracts the list of common codec(s), examines it and selects the most appropriate codec(s) to activate. A PRACK request, which includes the list of the selected codec(s), is sent back to the called party's UE.

According to 3GPP TS 27.060 [1], information about the required resources must be provided at this point to the network, to ensure a successful context activation attempt. The RUCRG testbed, does not support resource reservation but it remains important that the signalling is able to handle this requirement. The UE therefore assumes that the required resources have been provisioned without actually querying the network about their availability. As a result, instead of indicating in the PRACK that the resources needed for meeting the QoS requirements of the session are not available as per specification, we indicate that we have met the QoS requirements.

The called party's UE responds to the PRACK with a "200 OK" message, in which it also indicates that QoS for the session is met on its side. The final codec(s) at the called side is(are) also decided and relayed in the "200 OK" message. The Packet Data Protocol (PDP) context for the caller and called party are now active. The QoS for the call has now been met and all the resources for the call are in place. At this point, the called party's UE rings to notify the called party of the incoming call. The called party's UE sends the caller's UE a "180 Ringing" response to inform the caller that the called party is being alerted.

The caller's UE acknowledges the ringing message with a PRACK and the called party's UE responds to it with a "200 OK" message.

When the call has been answered, the remote UE notifies the caller by sending a "200 OK" message to the INVITE. The caller's UE acknowledges the "200 OK" message with an ACK request to complete the session establishment. At this point, the call is ready to enter conversation mode.

### 7.3.2 UAS Behaviour

In this section our client receives an incoming call. We will refer to our client as the called party's UE/called UE.

When the called party's UE receives an incoming call it examines the SDP list of available codecs. It prepares a list of codecs that are common between the UEs. This list is included

in the “183 Session Progress” message that is sent to the caller’s UE. The contact address in the “183 Session Progress” message is set to the called party’s IP address. The called UE copies the *Via* and *Record-Route* headers from the received INVITE and sends the response to the home P-CSCF.

The P-CSCF removes its own *Via* header entry and addresses the message to the top *Via* header (terminating S-CSCF in this case). As mentioned in the previous section, the “183 Session Progress” message retraces the path of the original INVITE because we did not change the *Via* and the *Record-Route* headers that we received in the INVITE.

Each node the response traverses removes its own entry from the *Via* header and forwards the message to the *via* entry at the top until the “183 Session Progress” response reaches the caller’s UE. The *Record-Route* header remains unchanged.

The caller’s UE then responds with a PRACK request to inform the called party’s UE about the codec(s) selected for the session. The called party’s UE responds to the PRACK with a “200 OK” message to indicate that QoS for the session is met on its side. This signifies that the called party has accepted the proposed codec(s). Similar to UAC behaviour, the PDP contexts for both the caller and the callee are active. The QoS for the call has now been met. All the resources for the call are in place.

At this point the called UE displays the incoming call screen to the called party as shown in Figure 7.7 and alerts the called party of the incoming call by ringing.

Figure 7.7: Incoming Call Screen



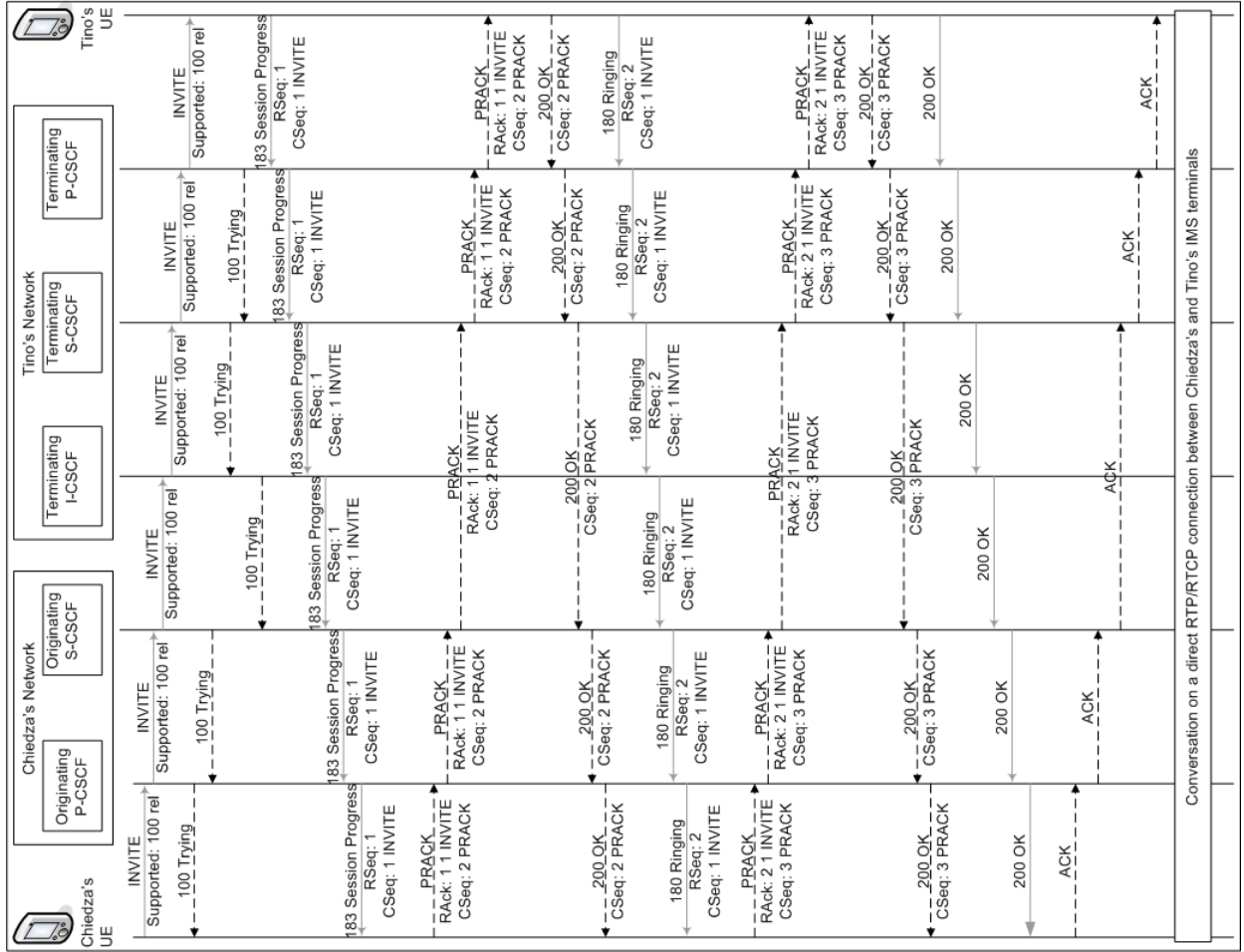
The called UE then sends a “180 Ringing” response to inform the caller that the called party is being alerted. When the caller acknowledges the ringing message with a PRACK, the called party’s UE responds to the PRACK with a “200 OK” message.

When the called party answers the call, the called UE notifies the caller by sending a “200 OK” message to the INVITE. The caller’s UE then acknowledges the “200 OK” message with an ACK request to complete the session establishment. At this point the call is now ready to enter conversation mode.

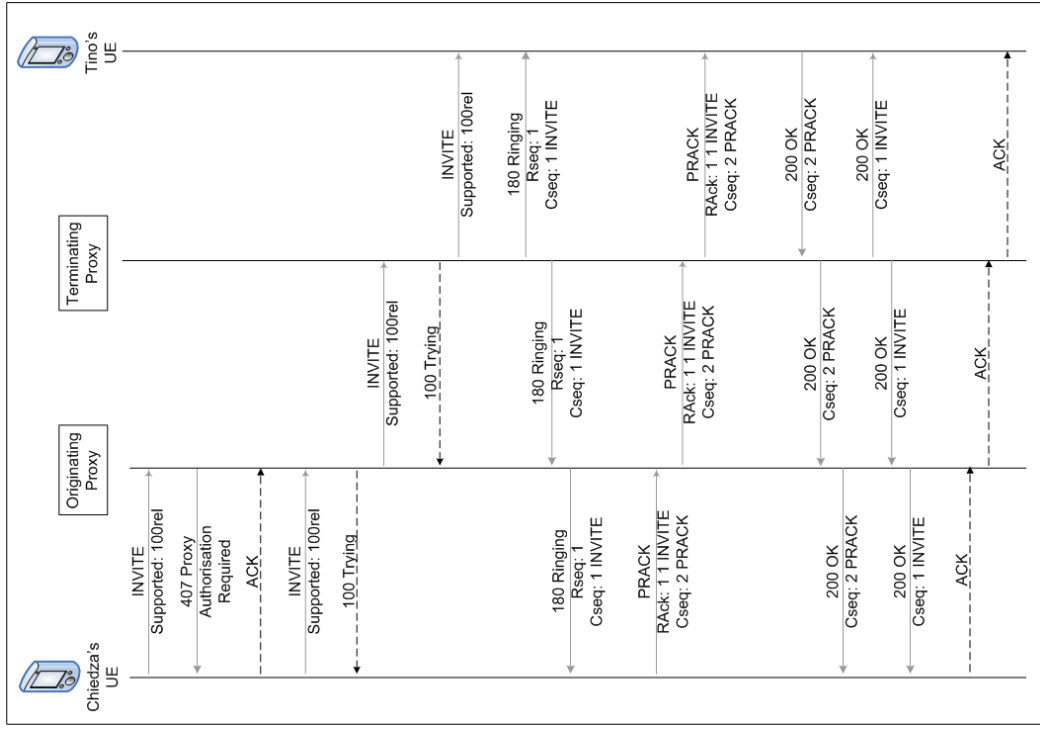
Figure 7.8a provides a summary of how the call setup procedure is handled in the RUCRG IMS client while Figure 7.8b shows how it is handled in JSAP++ using reliability mechanisms.

Figure 7.8: Session setup Call Flows

(a) IMS Session setup



(b) SIP Session setup Using Reliability Mechanisms





The differences between these call setup procedures are immediate. These figures also make clear the complexity of the IMS call setup procedure when it is compared to the SIP call setup.

## 7.4 SDP Codec Negotiation in IMS

In chapter 2, we established that SDP specifies a format for exchanging streaming related parameters between SIP subscribers. In this section, we discuss the signalling interactions between two IMS subscribers to illustrate how IMS codec selection using SDP was implemented in RUCRG IMS client. The discussion covers two phases of the SDP negotiation:

1. Codec selection between the calling and called IMS subscribers.
2. SDP signalling involved in exchanging QoS information.

As mentioned earlier, the IMS call setup starts with an initial INVITE request sent from the UAC to the P-CSCF. This INVITE contains a media offer, as discussed in the previous section. Similar to basic SIP, the presence of the SDP payload in the INVITE request is indicated by the `application/sdp` value in the *Content-Type* header, as illustrated in the INVITE request in the section 7.3.1.

The user initiates a call to a selected destination SIP URI. The caller's UE includes all the codecs that it supports in the SDP offer of the initial INVITE.

A trace of the SDP offer sent from the RUCRG IMS client is shown below, with media attribute fields integrated.

```
v=0
o=chiedza 960784 962153 IN IP4 146.xxx.xxx.xxx
s=-
c=IN IP4 146.xxx.xxx.xxx
t=0 0

m=audio 4152 RTP/AVP 0 3 9 4 5 6 8 15 18
b=AS:25
a=curr:qos local none
a=curr:qos remote none
a=des:qos mandatory local sendrecv
a=des:qos none remote sendrecv
```

```

a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:9 G722/8000
a=rtpmap:4 G723/8000
a=rtpmap:5 DVI4_8000/8000
a=rtpmap:6 DVI4_16000/16000
a=rtpmap:8 PCMA/8000
a=rtpmap:15 G728/8000
a=rtpmap:18 G729/-1

```

The UE sends the initial INVITE with nine voice codecs. It also indicates that it will need to allocate resources to meet the QoS requirements for codecs. The “m=” line specifies the caller-port 4152, the transport type (RTP/AVP) and the supported codecs IDs (0 3 9 4 5 6 8 15 18). The “a=rtpmap” lines map the codec IDs 0, 3, 9, 4, 5, 6, 8, 15 and 18 to PCMU, GSM, G722, G723, DVI4\_8000, DVI4\_16000, PCMA, G728 and G729. The “a=curr” lines show that the QoS for the caller (`local`) and the called party (`remote`) are not currently met. The first “a=des” line indicates that the caller (`local`) needs to allocate resources in send and receive directions to meet the QoS requirements for the codec. The last “a=des” line indicates that the caller has no specific requirements for the called party.

The called party’s UE examines the list of available codecs and prunes the list by excluding codecs that it does not support. This list will be included in the “183 Session Progress” message sent back to the caller.

A trace of the SDP answer sent from the RUCRG IMS client is shown below, with media attribute fields integrated.

```

v=0
o=chiedza 960784 962153 IN IP4 146.xxx.xxx.xxx
s=-
c=IN IP4 146.xxx.xxx.xxx
t=0 0

m=audio 5412 RTP/AVP 0 3 9 4 8 15
b=AS:25
a=curr:qos local none
a=curr:qos remote none
a=des:qos mandatory local sendrecv

```

```

a=des:qos mandatory remote sendrecv
a=conf:qos remote sendrecv
a=rtpmap:0 PCMU/8000
a=rtpmap:3 GSM/8000
a=rtpmap:9 G722/8000
a=rtpmap:4 G723/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:15 G728/8000

```

From the SDP trace above, the called party's UE replies back with 0, 3, 9, 4, 8 and 15 in the "m=" line. Codecs 5, 6, and 18 are removed since they are not supported. The called UE also uses the "a=curr" lines to specify that QoS for the session is currently not met. Note that the "a=des" lines now signify that the called party (local) also needs to allocate resources for meeting the QoS. This message also instructs the caller to inform the called party when the resources for meeting the QoS are acquired. This QoS confirmation is being requested in the line "a=conf".

When the caller's UE receives the "183 Session Progress", it examines the received common codec(s) list and selects the codec(s) to activate.

```

v=0
o=chiedza 960784 962153 IN IP4 146.xxx.xxx.xxx
s=-
c=IN IP4 146.xxx.xxx.xxx
t=0 0

m=audio 4152 RTP/AVP 0
b=AS:25
a=curr:qos local sendrecv
a=curr:qos remote none
a=des:qos mandatory local sendrecv
a=des:qos none remote sendrecv
a=rtpmap:0 PCMU/8000

```

The caller's UE now sends a PRACK to inform the called party about the selected codec(s). The PCMU codec with frequency 8000 has been selected for use in this session. This is signalled by the "m=" and "a=" lines. Due to the fact that the Rhodes University testbed does not support resource reservation as explained in the previous section, the caller's UE indicates that the QoS for the call has now been met and all the resources for

the call are in place. The caller's PDP context gets activated, and the caller's UE notifies the called party's UE that it can now meet the QoS in the send and receive direction. The "a=curr:qos local sendrecv" signals that caller's PDP context has been established.

Now that the caller has selected the codec(s) to be used and indicated that resources have been reserved for the selected codec, the called party responds to the PRACK with a "200 OK" message. The message also indicates that QoS for the session have been met by the called party.

```
v=0
o=chiedza 960784 962153 IN IP4 146.xxx.xxx.xxx
s=-
c=IN IP4 146.xxx.xxx.xxx
t=0 0

m=audio 5412 RTP/AVP 0
b=AS:25
a=curr:qos local sendrecv
a=curr:qos remote sendrecv
a=des:qos mandatory local sendrecv
a=des:qos mandatory remote sendrecv
a=conf:qos remote sendrecv
a=rtpmap:0 PCMU/8000
```

Note that the "a=curr" line for the called party (local) has been updated to indicate that called end QoS is also met.

Now all the resources for the call are in place, the called party's UE rings to notify the called party (callee) of the incoming call. The called party's UE also sends us "180 Ringing" response to inform the caller that the called party is being alerted. The caller's UE acknowledges the ringing message with a PRACK and the called party's UE responds to the PRACK with a "200 OK" message.

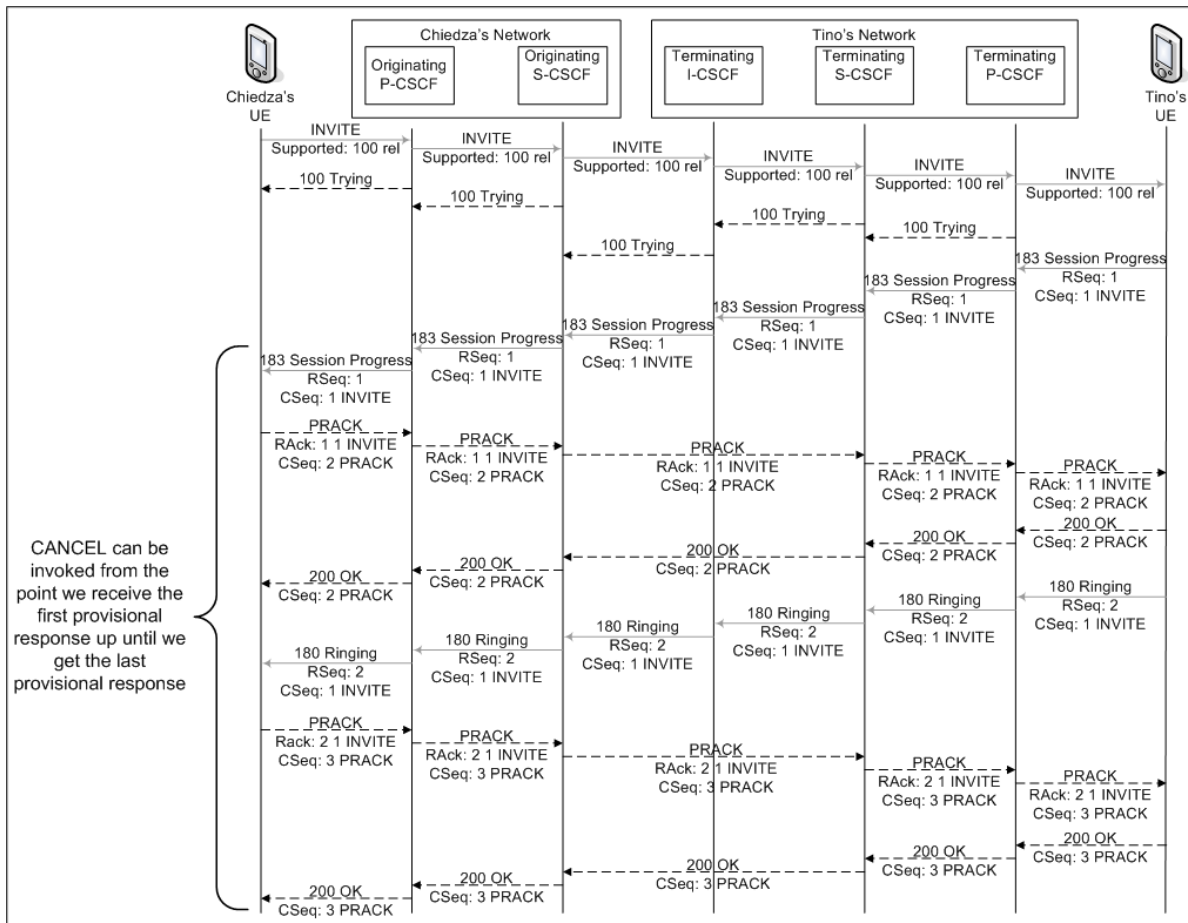
When the called party answers the call, their UE notifies us of this event by sending a "200 OK" message to our INVITE. Our UE then acknowledges the "200 OK" message with an ACK request to complete the session establishment. At this point, the call is now ready to enter conversation mode.

## 7.5 Session Cancellation

In this section, we discuss how the SIP CANCEL in JSAP++ was transformed to be IMS compliant. As usual, our client acts as both a UAC and a UAS.

Cancelling a SIP session is performed using the SIP CANCEL method as specified in RFC 3261 [54]. In the case of the RUCRG IMS client, a user can cancel an INVITE request by pressing the Cancel button in the GUI. During session establishment, a CANCEL request can only be sent from the moment we receive the first provisional response, up until just before we receive the final response as illustrated in Figure 7.9. Furthermore, we can only cancel sessions that we originated as specified in RFC 3261.

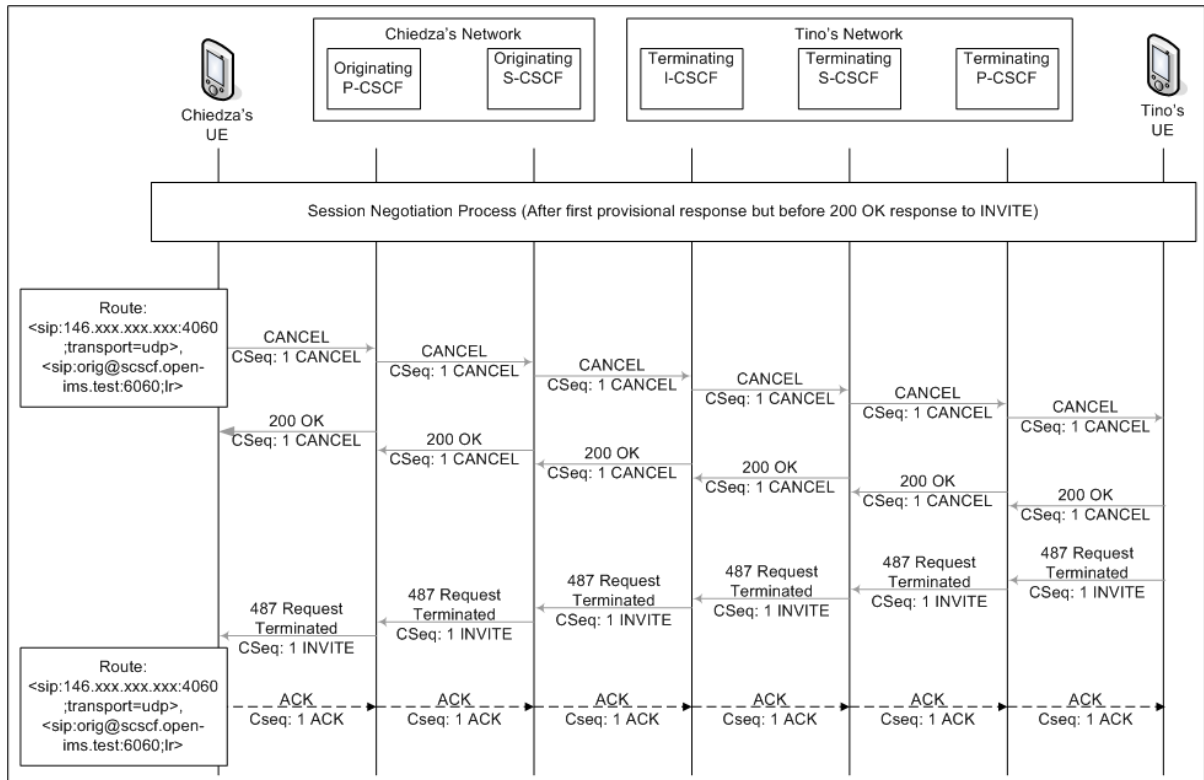
Figure 7.9: Points when CANCEL is enabled for RUCRG IMS client



In order for the CANCEL to traverse the correct IMS network elements, the P-CSCF name and the S-CSCF names are included in the *Route* header of the CANCEL request. The S-CSCF name is the name of the home service proxy that we received in the *Service-Route* header during registration. Figure 7.10 illustrates how a CANCEL request is sent

from the RUCRG IMS client to another IMS client.

Figure 7.10: RUCRG IMS client CANCEL sequence diagram

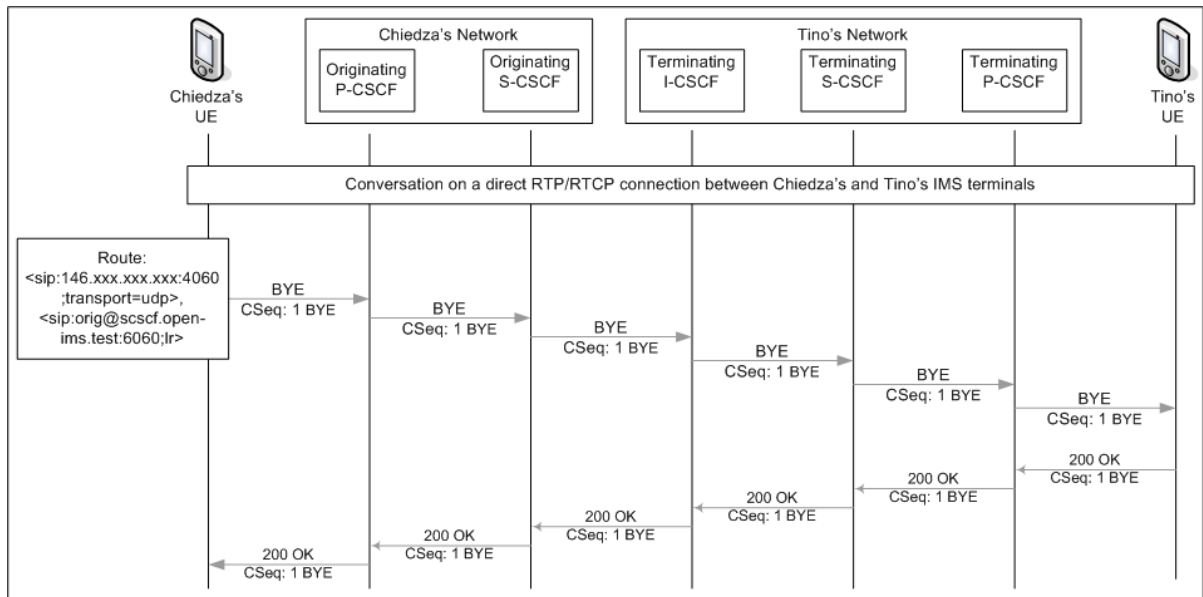


## 7.6 Session Ending

In this section, we discuss how the SIP BYE in JSAP++ was transformed to be IMS compliant. As in the CANCEL case, our client acts as both a UAC and a UAS.

A SIP session is ended using the SIP BYE method as specified in RFC 3261 [54]. In the case of the RUCRG IMS client, a user can terminate a request by pressing the Stop button in the GUI, as shown in Figure 7.11. Once the Stop button has been pressed, the UAC in the RUCRG IMS client will send a BYE request. Following the requirements of RFC 3261 [54], our IMS client does not send BYE requests outside of Dialogs. The client can only send BYE requests for either confirmed or early Dialogs if the call was locally initiated. If the call was originated remotely, the client can send BYE requests on confirmed Dialogs, but not on early Dialogs. Similar to the CANCEL, the P-CSCF name and the S-CSCF name are included in the *Route* header of the BYE request. Once again, the home service proxy name is established during registration from the *Service-Route* header.

Figure 7.11: RUCRG IMS client BYE sequence diagram



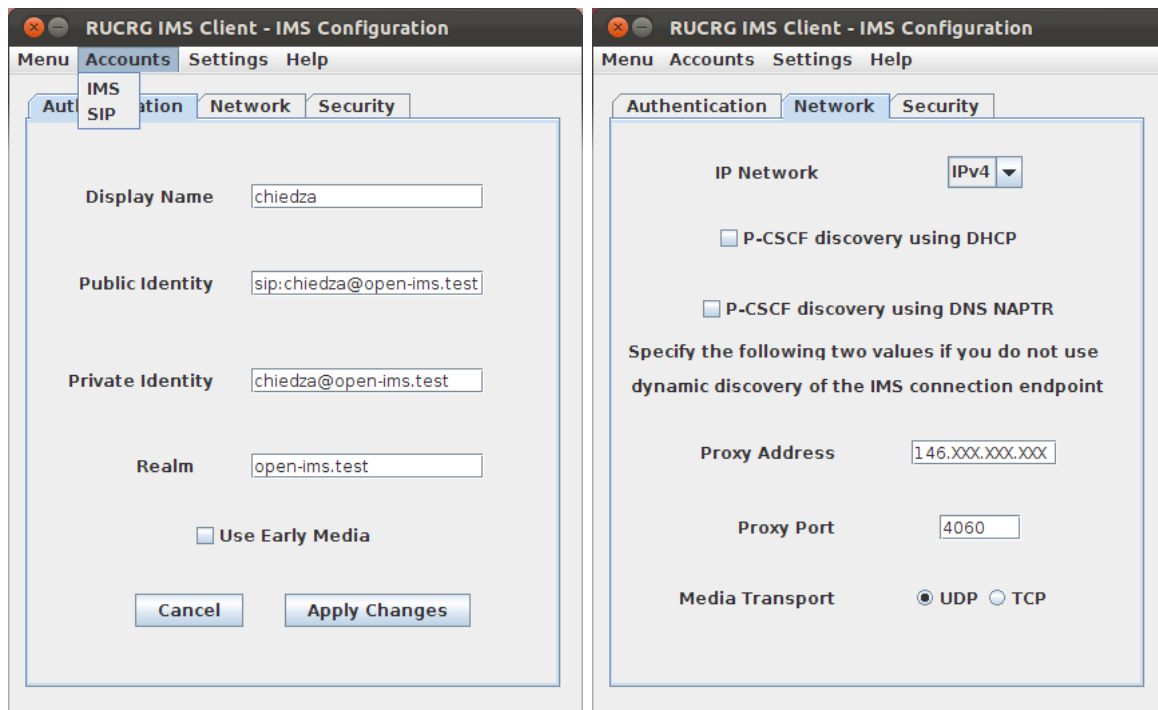
The final architecture of the RUCRG IMS client is the same as the one we presented at the end of chapter 6 (in section 6.4) but the call flow has changed to suite both SIP and IMS.

## 7.7 Putting Things Together

When the RUCRG IMS client application is started, the GUI is loaded through the `NISTMessenger` class. `NISTMessenger` creates an instance of `MessageListener` whose constructor method contains the parameters needed to initialise the JAIN SIP environment. Once the system has been initialised, `MessageListener` is ready to receive events from the `SipProvider` or from the user interface. The SIP port that will be used is introduced through the `JSAPConfig.xml`, and communicated to the `MessageListener`. The port always refers to a UDP port because we will always use UDP as the network transport. In order to create the Listening Point, the client needs to pass the IP address, port, and transport as an argument. The port and server address are found in the `JSAPConfig.xml`, whereas the IP address is directly obtained by the client.

Figure 7.12 shows the RUCRG IMS client display before the user can register with the home IMS network. As shown in the figure, the client is designed to allow the user to choose between registering with the IMS or an ordinary SIP network. Furthermore, the menu allows the user to terminate a registration or exit the application completely.

Figure 7.12: IMS Client Display Before Registration



The RUCRG IMS client is configured to first prompt the user to register, if they are not already registered before they can perform key IMS procedures like session initiation. A REGISTER request is sent to the home IMS registrar, once the Register menu item has been selected. If the user selects the Exit menu item, the RUCRG IMS client application will be closed.

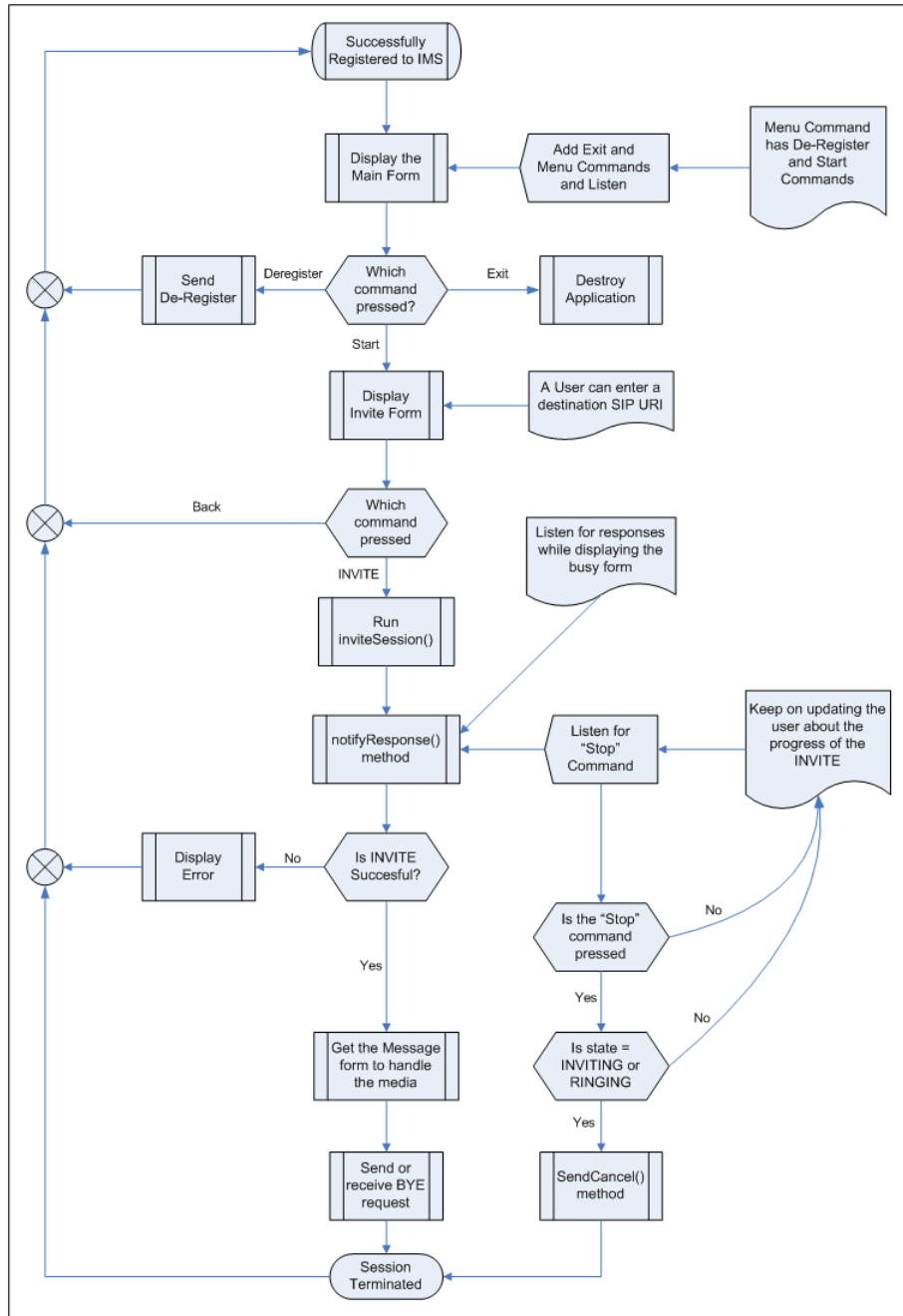
Once the GUI is loaded, the user has to choose whether they want to start a SIP or an IMS session. At that point, the user has to fill in some configuration parameters (server port, his or her own Address-of-Record and the server address in the GUI etc.), and then press the Apply button. This GUI event causes the configured parameters to be saved in the JSAPConfig.xml file in the user directory and updated in the Properties. After doing so, the user can press the Register menu item, causing the client to send a REGISTER request to the identified server. If the request has been sent successfully, the user should see a dialog box that shows two fields requiring the user-name and password. If the correct credentials are provided, the user is successfully bound to the supplied IP address and port.

As stated before, registration is the first step towards accessing IMS services. After a successful registration to the home IMS network, an Invite screen, which allows the user to establish a session, is displayed. After a successful invitation, the call is started. Pressing the End command on the call screen will end the communication.



The main flow diagram of the UAC side of IMS Client is shown in Figure 7.13. The diagram shows the flow of procedures and events to be performed by our IMS client after a successful registration, the creation of the session and the ending of the session.

Figure 7.13: IMS UAC Main Flow Diagram



## 7.8 Summary

In this chapter, we presented how IMS functionality was added to JSAP++ to come up with the RUCRG IMS client. We also explained how some important SIP extensions were used such as *Route*, *P-Preferred-Identity*, *P-Access-Network-Info*.

In sum, this chapter showed how we transformed JSAP from a very basic SIP client with unsatisfactory signalling and media support, into an IMS compliant client with robust and complete signalling and media support. In the next chapter, we will present the tests that were carried out on the RUCRG IMS client to test its conformance and interoperability.

# Chapter 8

## RUCRG IMS Client Testing

Chapters 5, 6 and 7 provided details of the implementation of the functional IMS client prototype called the RUCRG IMS client. This chapter presents the results of the various compatibility tests that we performed between the RUCRG IMS client, SIP application servers and other freely available IMS and/or SIP clients. The chapter also details the tests that were carried out to evaluate the client's conformance with IMS and SIP standards.

### 8.1 Testing

According to ETSI [17], equipment implementing standardised protocols and services can be tested in two related but different ways. These are described below:

#### 8.1.1 Conformance Testing

Conformance testing involves establishing the extent to which a device that has not previously been shown to conform, known as the equipment under test (EUT), complies with the requirements specified by a particular protocol [17]. There are a number of SIP conformance test suites that have been developed. For example, ETSI have developed a SIP test suite in a standardised testing language called Testing and Test Control Notation version 3 (TTCN-3) [16]. Another SIP conformance test tool is the TAHI project, which was developed in Japan. However, we failed to find a test suite for conformance with IMS.

One of the major problems with test suites is that they cover a great amount of test cases making it difficult to choose which tests cases to run. Li *et al* [34] argue that in the ETSI test suite each test case corresponds literally to one or two sentences in the SIP protocol

specification. Additionally, ETSI does not make available the detailed information of the SIP conformance test system for commercial reasons. This also makes choosing test cases more complex. On the other hand, the TAHI test suite is based on Perl, which may lead to problems in understanding and modifying the tests for various purposes. As a result this thesis used a simple protocol monitor to test for conformance.

### 8.1.2 Interoperability Testing

Interoperability testing is aimed at assessing the ability of a device, which has not previously been shown to interoperate (EUT), to support required functionality between itself and the qualified equipment (QE) to which it is connected [17]. The QE is an application/device which implements the same protocol as the EUT, but has already been proven to interoperate with similar applications/devices from other suppliers. Unlike conformance testing, interoperability testing does not seek to verify the protocol requirements. Instead, interoperability tests are performed to ensure correct exchange and use information between products.

During protocol specification, standardisation bodies may not specify how applications should behave in a given scenario. Discretion is left to the developer to decide how they want their application to behave. As such it becomes difficult to test for conformance and this may also result in interoperability problems among applications because various parts of the protocol may be interpreted differently by different programmers.

### 8.1.3 Interoperability with Conformance Monitoring

Combining interoperability with conformance monitoring is one way of dealing with a situation where a conformance suite is not used. In this case a human interpreter analyses the protocol monitor output. According to ETSI [17] "... it is valid to consider using the techniques together to give a combined result." ETSI [17] further argue that, "... some limited conformance testing with extensive interoperability testing ... may be useful in certain situations." We therefore used this approach in this thesis to test our client. The test setup is shown in Figure 8.2 (section 8.3). Consequently, our results were obtained in one single set of experiments.

## 8.2 Testing Requirements

The list of hardware, software and tools used for setting up the experiments are provided in the following subsections.

### 8.2.1 Hardware Requirements

Table 8.1 lists the hardware requirements for conducting the experiments.

Table 8.1: Hardware Specifications

Desktop Personal Computer	
Component	Description
CPU	Intel® Core™ i7-870 2.93 GHz
RAM	4 GB
HDD	500 GB
Network Connection	Gigabit Ethernet
Laptop Personal Computer	
CPU	Intel® Pentium® Processor T4500
RAM	2.5 GB
HDD	250 GB
Network Connection	Gigabit Ethernet

### 8.2.2 Software Requirements

Table 8.2 lists the software that was used for conducting the experiments.

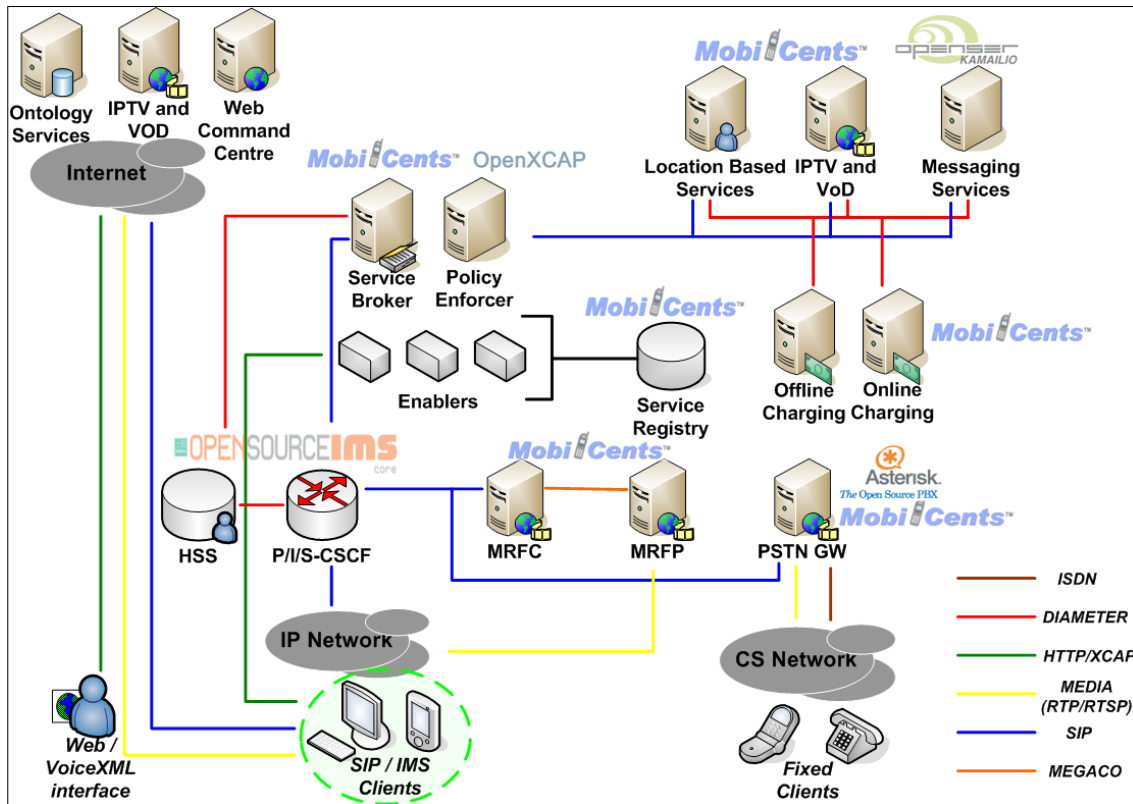
Table 8.2: Software Specifications

Desktop Personal Computer	
Software	Description
Operating System	Linux Ubuntu 9.10
Java	JDK 1.6.0, Java Virtual Machine(JVM)
Netbeans	6.8
Gstreamer	1.4
Laptop Personal Computer	
Operating System	Microsoft Windows Vista Home Edition/Linux Ubuntu 10.10
Java	JDK 1.6.0_23, Java Virtual Machine(JVM)
Netbeans	6.9.1
Gstreamer	1.4
Wireshark	A network protocol analyser used to capture network packets

### 8.2.3 Testbed Specifications

Figure 8.1 captures our testing environment, the RUCRG testbed. The figure also shows the position of the RUCRG IMS client within the testbed as well as the access network.

Figure 8.1: RUCRG IMS Client Position in RUCRG Testbed

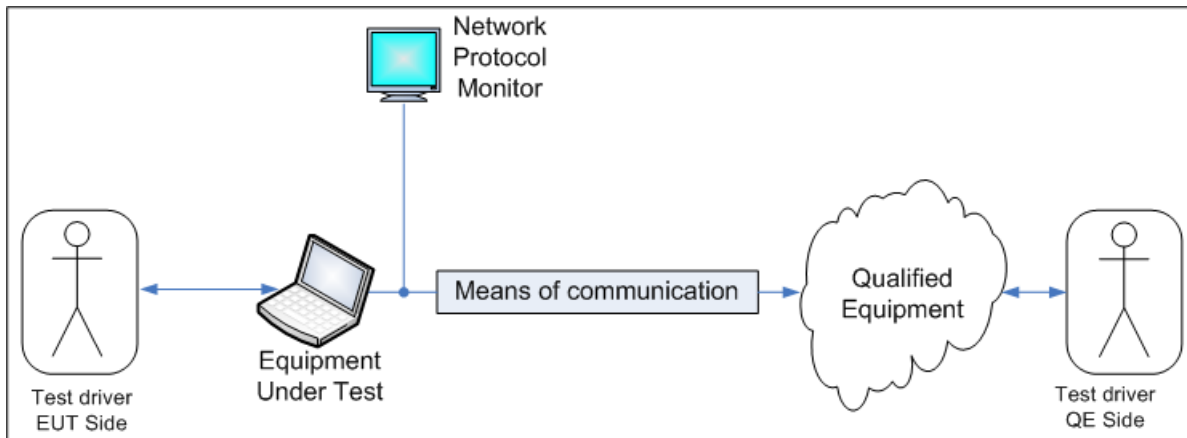


The testbed was originally built for SIP experimentation. A move towards compliance with IMS was then started and work is still in progress to finalise the migration. The IMS part of the testbed comprises the IMS CN, which provides the basic control layer elements: P-CSCF, I-CSCF, S-CSCF and HSS as discussed in chapter 2.

### 8.3 Test Setup

The RUCRG IMS client (EUT) was tested within the controlled environment of the RUCRG testbed described above. Since a registrar was used to locate the users in the RUCRG testbed, the clients involved in any communication had to be registered with either the IMS CN or SIP proxy server. Two private user identities were created to be used within IMS: Chiedza@open-ims.test and Tino@open-ims.test. The clients involved in the test were hosted on two different machines within the same local area network (LAN) as shown in Figure 8.2.

Figure 8.2: Interoperability Testing with Conformance Monitoring (Adapted from ETSI [17])



This approach enabled us to run a network protocol monitor (wireshark) on the machine that our EUT was running on as described earlier. Additionally, this setup enabled us to carry out conformance tests since we were monitoring all the traffic generated and received by the EUT. While this arrangement cannot provide a complete and time efficient proof of conformance, analysis of the protocol monitor output was able to show whether the signalling between the EUT and QE conformed to the appropriate standard(s) throughout the testing. The EUT was initially deployed on the laptop while it was running Windows. The experiments were then repeated with the EUT deployed on the laptop running Linux Ubuntu. The QEs were deployed on the personal computer. Among the QEs used were two IMS clients (Mercurio IMS client and UCT IMS client), four SIP clients (Twinkle, Ekiga, GRANDSTREAM GXV3140 and SJphone), FOKUS IMS Core as the IMS CN and Kamailio as the SIP proxy. After registration, either one of the client applications (QE or EUT) was used to initiate requests, while the other responded to them.

## 8.4 System Testing

We have already alluded to the fact that both conformance and interoperability are important approaches of testing standardised protocol implementations. In this section, we describe in detail the basic conformance and interoperability tests that were carried out between the RUCRG IMS client and freely available: SIP proxy servers (Kamailio), SIP clients (Twinkle, Ekiga), IMS core networks (FOKUS IMS Core) and IMS clients (Mercurio IMS client and UCT IMS client). We defined test cases for IMS/SIP registration and session initiation, media support, presence support and IM support. These



cases were based on end-to-end systems testing, that is, higher level functionality, rather than on specific protocol requirements. Each experiment is described using the following structure:

- Entities involved.
- Test purpose.
- Preconditions.
- Result.

The test cases that we defined were not meant to provide exhaustive testing of all facets of SIP protocol operation. Instead, we chose scenarios that provided coverage of the functionality that we implemented in the EUT.

## 8.4.1 Endpoint Registration with a Registrar

### 8.4.1.1 Entities Involved

UE (EUT), IMS CN (P-CSCF, S-CSCF, I-CSCF and HSS), SIP proxy.

### 8.4.1.2 Test Purpose

To evaluate registration capabilities of the EUT with Kamailio (acting as the SIP registrar) as well as with FOKUS IMS Core (acting as the IMS registration server).

### 8.4.1.3 Preconditions

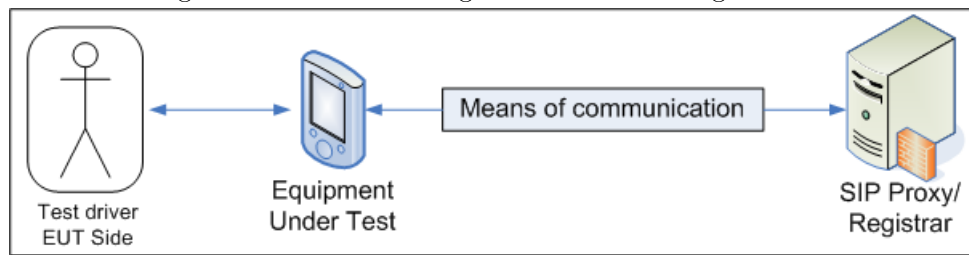
EUT is pre-configured with the proxy server it is supposed to send registration requests.

### 8.4.1.4 Results

#### **SIP Registration**

Since the JSAP client already supported ordinary SIP registration, the aim of this test was to make sure the client could still successfully register with any ordinary SIP proxy server after modifications had been made to the code.

Figure 8.3: Test Arrangement for SIP Registration



The results of these tests are presented in Table 8.3.

Table 8.3: SIP Registration Test Results

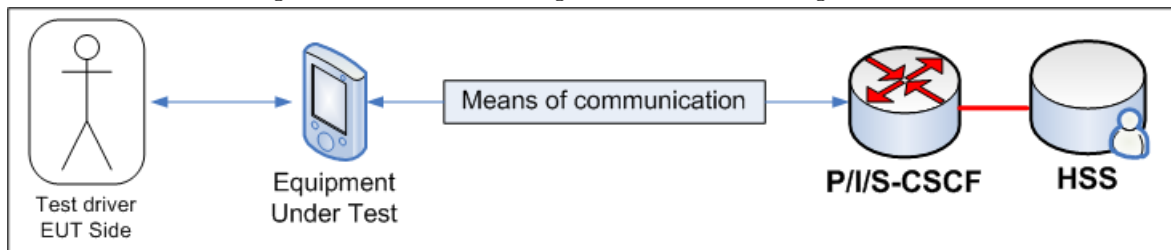
Test Case	Test				Result
EUT registration with the SIP registrar (Kamailio)	EUT successfully sends a REGISTER request to the Registrar	EUT successfully processes "401 Unauthorized" received from the Registrar	EUT successfully sends a second REGISTER request with credentials to the Registrar	EUT successfully processes "200 OK" received in response to the REGISTER request	Pass
EUT refreshes contact address with the SIP registrar (Kamailio)	EUT successfully sends a REGISTER request with Expiry set to 60 seconds to the Registrar	EUT successfully processes "401 Unauthorized" received from the Registrar	EUT successfully sends a second REGISTER request with credentials to the Registrar	EUT receives "200 OK" to the REGISTER request	EUT successfully sends another REGISTER request 30 seconds after the first in order to refresh the registration (NOTE 1). Pass
EUT de-registers contact address with the SIP registrar (Kamailio)	EUT successfully sends a REGISTER request with Expiry set to 0 seconds to the Registrar	EUT receives "401 Unauthorized" from the Registrar	EUT successfully sends a REGISTER request with credentials and Expiry set to 0 seconds to the Registrar	EUT successfully processes "200 OK" message received from the QE	Pass
EUT tries to register with non existing SIP registrar	EUT successfully times out after 60 seconds				Pass

As one can see, the EUT can successfully register, de-register and refresh registrations with a SIP registrar.

### IMS Registration

This test was used to evaluate whether the client could register successfully with FOKUS IMS Core.

Figure 8.4: Test Arrangement for IMS Registration



The results are presented in Table 8.4. This test was also passed.

Table 8.4: IMS Registration Test Results

Test Case	Test				Result
EUT registration with the IMS network (FOKUS IMS core)	EUT successfully sends a REGISTER request to the Registrar	EUT successfully process "401 Unauthorized" received from the Registrar	EUT successfully sends a second REGISTER request with credentials to the Registrar	EUT receives "200 OK" to the REGISTER request	Pass
EUT Refreshes contact address with the IMS network (FOKUS IMS core)	EUT successfully sends a REGISTER request with Expiry set to 60 seconds to the Registrar	EUT successfully processes "401 Unauthorized" received from the Registrar	EUT successfully sends a second REGISTER request with credentials to the Registrar	EUT receives "200 OK" to the REGISTER request	EUT successfully sends another REGISTER request 30 seconds after the first in order to refresh the registration (NOTE 1). Pass
EUT de-registers contact address with the IMS network (FOKUS IMS core)	EUT successfully sends a REGISTER request with Expiry set to 0 seconds to the Registrar	EUT receives "401 Unauthorized" from the Registrar	EUT successfully sends a REGISTER request with credentials and Expiry set to 0 seconds to the Registrar	EUT successfully processes "200 OK" message received from the QE	Pass
EUT tries to register with non existing IMS network	EUT successfully times out after 60 seconds				Pass

NOTE 1: 3GPP TS 24.229 [6] mandates that a UE should re-register an already registered public user identity either 600 seconds before expiration time if the previous registration was for a period greater than 1200 seconds, or when half of the registration time has expired if the previous registration was for 1200 seconds or less.

## 8.4.2 Point-to-point Audio/Visual call using Proxy/IMS

### 8.4.2.1 Entities Involved

UEs (QEs and EUT), IMS CN (P-CSCF, S-CSCF, I-CSCF and HSS), SIP proxy.

### 8.4.2.2 Test Purpose

To verify that a voice and video communication can be successfully established from EUT to the QE and vice versa. The EUT and the QE both assumed the “originating” and “terminating” roles with respect to initiating the call in successive tests.

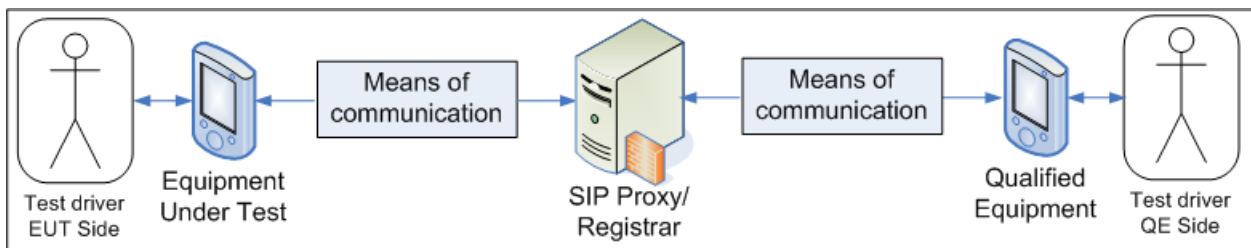
### 8.4.2.3 Preconditions

Communicating clients are registered.

### 8.4.2.4 Results

#### SIP Session Setup

Figure 8.5: Test Arrangement for SIP Session Setup



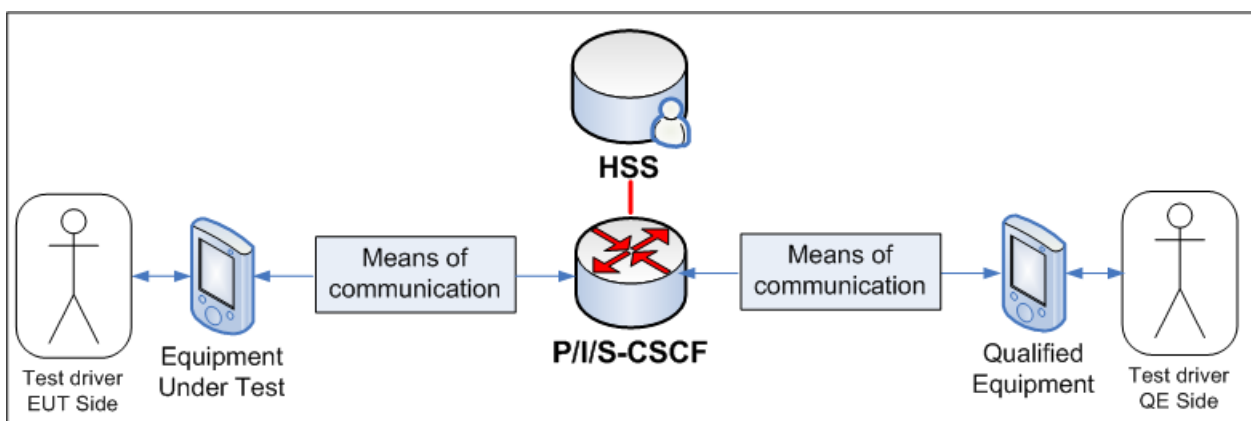
The results are presented in Table 8.5 and show that the EUT is capable of setting up a media session with a QE using SIP.

Table 8.5: SIP Media Session Setup Results

Test Case	Test			Result
Call establishment from QE to EUT through SIP proxy with authentication	EUT successfully processes INVITE received from QE and sends back "180 Ringing"	EUT successfully sends "200 OK" message to QE	EUT successfully processes ACK received from QE	Pass
Call establishment from EUT to QE through SIP proxy with authentication	EUT successfully sends INVITE to the QE	EUT successfully processes the "180 Ringing" received from the QE	EUT successfully processes "200 OK" received from the QE and sends back an ACK	Pass

### IMS Session Setup

Figure 8.6: Test Arrangement for IMS Session Setup



The results presented in Table 8.6 show that our client can successfully setup an IMS media session with another IMS client through the FOKUS IMS Core.

Table 8.6: IMS Media Session Setup Results

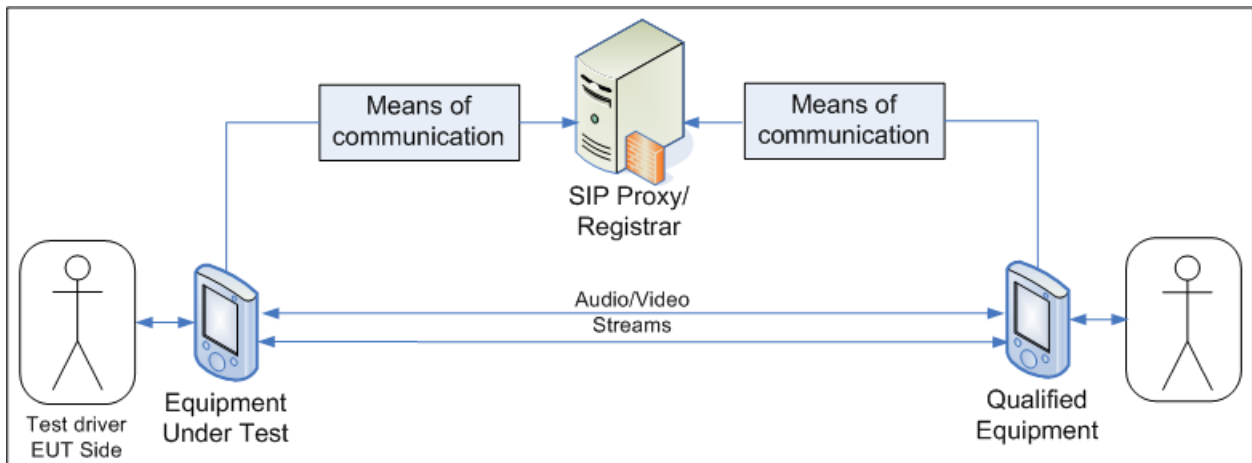
Test Case	Test					Result	
	EUT successfully processes INVITE received from QE and sends back "183 Session Progress"	EUT successfully processes PRACK (response to "183 Session Progress") received from QE and sends back "200 OK"	EUT successfully sends "180 Ringing" to QE	EUT successfully processes PRACK (response to "180 Ringing") received from QE and sends back "200 OK"	EUT successfully sends "200 OK" (in response to the INVITE) to QE		EUT successfully processes ACK received from QE
Call establishment from QE to EUT through the IMS with authentication	EUT successfully processes INVITE received from QE and sends back "183 Session Progress"	EUT successfully processes PRACK (response to "183 Session Progress") received from QE and sends back "200 OK"	EUT successfully sends "180 Ringing" to QE	EUT successfully processes PRACK (response to "180 Ringing") received from QE and sends back "200 OK"	EUT successfully sends "200 OK" (in response to the INVITE) to QE	EUT successfully processes ACK received from QE	Pass
Call establishment from EUT to QE through the IMS with authentication	EUT successfully sends INVITE to QE	EUT successfully processes "183 Session Progress" received from QE and sends back PRACK	EUT successfully processes "200 OK" (to "183 Session Progress" PRACK) received from QE	EUT successfully processes "180 Ringing" received from QE and sends back PRACK	EUT successfully processes "200 OK" (to "180 Ringing" PRACK) received from QE	EUT successfully processes "200 OK" (to INVITE) and sends back ACK	Pass



### Decoding and Display of Multimedia Streams

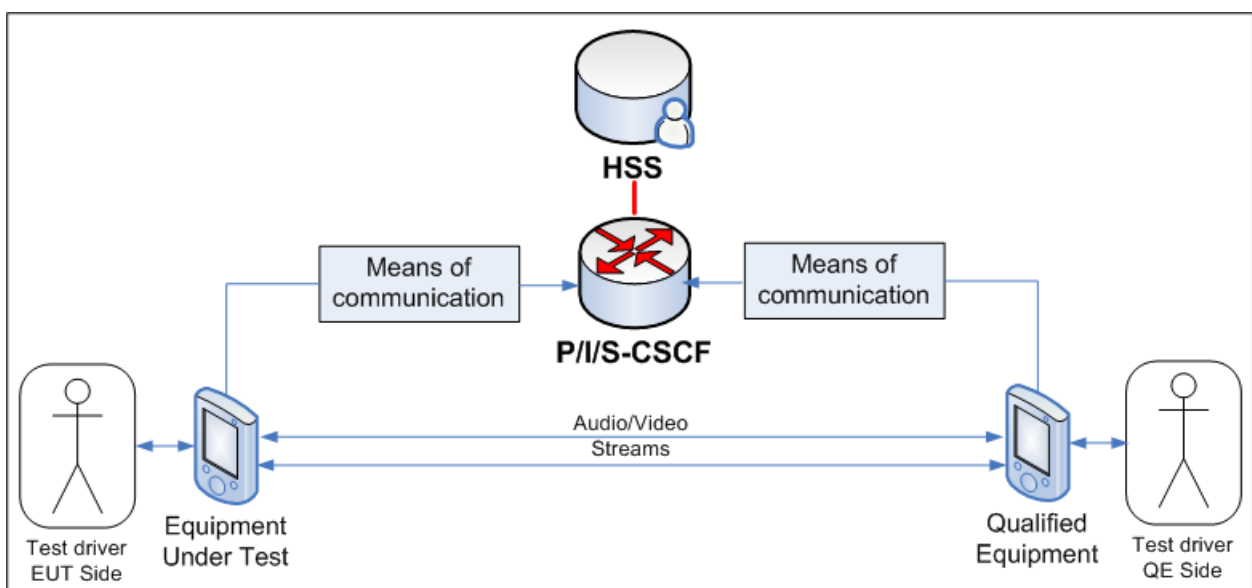
To evaluate media support, audio and video sessions were established between the EUT and QE through a SIP proxy server. The audio and video streams were analysed by a human user at the EUT and the QE. Figure 8.7 illustrates the test arrangement:

Figure 8.7: Test Arrangement for SIP Media Test Case



The same set of tests was repeated with the call being routed through the IMS network. Figure 8.8 shows the test arrangement:

Figure 8.8: Test Arrangement for IMS Media Test Case



The tests were framed by the following questions:

- Can speech from EUT be heard and understood at QE?
- Can video from EUT be seen and understood at QE?
- Can speech from QE be heard and understood at EUT?
- Can video from QE be seen and understood at EUT?

All the tests were successful and video and audio streams could be decoded from either side for both cases: calls established through a SIP proxy and those established through the IMS network.

### **SIP Session Termination and Call Clearing**

The aim of this test was to ensure that the client could successfully terminate/clear a call session with another SIP client through a SIP proxy server. The test setup for SIP session termination and call clearing is similar to that for SIP session setup in Figure 8.5. The results are presented in Table 8.7.

Table 8.7: SIP Call Clearing and Call Rejection Results

Test Case	Test		Result
Clearing of an active call/BYE from EUT to QE through a SIP proxy	EUT successfully sends a BYE request to QE	EUT begins disconnecting by releasing RTP session	EUT successfully processes "200 OK" received from QE
Clearing of an active call/BYE from QE to EUT through a SIP proxy	EUT successfully processes a BYE request from QE and sends back "200 OK"	EUT successfully releases RTP session and completely disconnects	Pass
Call clearing before destination answers/CANCEL (EUT to QE) through a SIP proxy	EUT successfully sends a CANCEL request to QE	EUT successfully processes "200 OK" received from QE	Pass
Call clearing before destination answers/CANCEL (QE to EUT) through a SIP proxy	EUT successfully processes CANCEL request received from QE and sends back a "200 OK"	EUT successfully sends a "487 Request Terminated" to QE	Pass
Rejection of incoming call/BYE originating from EUT through a SIP proxy	EUT successfully sends a BYE request to QE		Pass
Rejection of incoming call/BYE originating from QE through a SIP proxy	EUT successfully processes a BYE request received from QE and sends back a "200 OK"		Pass

The results show that the EUT is capable of successfully terminating/clearing established SIP sessions with a QE.

### **IMS Session Termination and Call Clearing**

The aim of this test was to ensure that the client could successfully terminate/clear a call session with another IMS client through the IMS network. The test setup for IMS session termination and call clearing is similar to that for IMS session setup in Figure 8.6. The results are presented in Table 8.8.

Table 8.8: IMS Call Clearing and Call Rejection Results

Test Case	Test			Result
Clearing of an active call/BYE from EUT to QE through an IMS network	EUT successfully sends a BYE request to QE	EUT begins disconnecting by releasing RTP session	EUT successfully processes "200 OK" received from QE	Pass
Clearing of an active call/BYE from QE to EUT through an IMS network	EUT successfully processes a BYE request received from QE and sends back a "200 OK" message	EUT successfully releases RTP session and completely disconnects		Pass
Call clearing before destination answers/CANCEL (EUT to QE) through an IMS network	EUT successfully sends a CANCEL request to QE	EUT successfully processes "200 OK" received from QE		Pass
Call clearing before destination answers/CANCEL (QE to EUT) through an IMS network	EUT successfully processes CANCEL request received from QE and sends back "200 OK"	EUT successfully sends a "487 Request Terminated" to QE		Pass
Rejection of incoming call/BYE originating from EUT through an IMS network	EUT successfully sends a BYE request to QE			Pass
Rejection of incoming call/BYE originating from QE through an IMS network	EUT successfully processes a BYE request received from QE and sends back a "200 OK"			Pass

The results show that the EUT is capable of terminating/clearing an established IMS session with a QE.

### 8.4.3 Presence

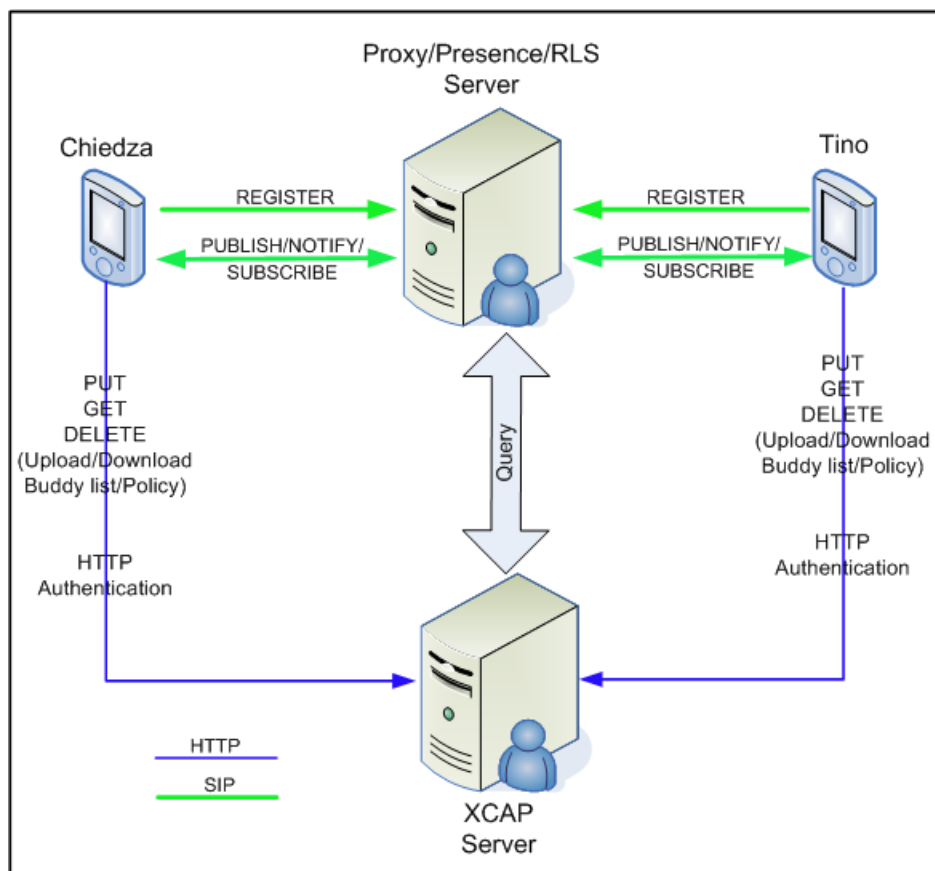
#### 8.4.3.1 Entities Involved

UE (EUT and QEs), XCAP Server, SIP proxy.

#### 8.4.3.2 Test Purpose

Testing was carried out to determine whether the client was capable of performing relevant XCAP functions.

Figure 8.9: Test Arrangement for Presence Test Case



#### 8.4.3.3 Preconditions

The user has an account configured on the XCAP server.

#### **8.4.3.4 Results**

These tests were conducted to verify the ability of the client to perform the XCAP functions implemented in chapter 6. The results are presented in Table 8.9.

Table 8.9: Summary of XCAP Results

Test Case	Test	Result
Create/Replace XCAP Document	EUT successfully performs an HTTP PUT request with the request URI set to the location where the document is to be placed	Pass
Create/Replace an element in an XCAP Document (add/modify users)	EUT successfully performs an HTTP PUT request with the request URI set to the location of the document to be modified and the node selector set to the element to be created/replaced	Pass
Create/Replace an attribute in an XCAP Document (add/modify user attributes)	EUT successfully performs an HTTP PUT request with the request URI set to the location of the document to be modified and the node selector set to the attribute to be created/replaced	Pass
Fetch an element in an XCAP Document	EUT successfully performs an HTTP GET request with the request URI set to the location of the document to be queried and the node selector set to the element to be retrieved	Pass EUT successfully processes "200 OK"
Fetch an attribute in an XCAP Document	EUT successfully performs an HTTP GET request with the request URI set to the location of the document to be queried and the node selector set to the attribute to be retrieved	Pass EUT successfully processes "200 OK"
Fetch an XCAP Document	EUT successfully performs an HTTP GET request with the request URI set to the location where the document is to be retrieved	Pass EUT successfully processes "200 OK"
Delete an element (delete user)	EUT successfully performs an HTTP DELETE request with the request URI set to the location of the document to be modified and the node selector set to the element to be deleted	Pass
Delete an attribute (delete user attribute)	EUT successfully performs an HTTP DELETE request with the request URI set to the location of the document to be modified and the node selector set to the attribute to be deleted	Pass
Delete XCAP Document	EUT successfully performs an HTTP DELETE request on a URI that references the document to be deleted	Pass



---

The EUT managed to automatically retrieve its buddy list from the XCAP server after successfully logging into either the IMS network or a SIP network. It then used the list to send subscriptions to all the buddies listed. Similarly, the EUT managed to upload the latest copy of the buddy list onto the XCAP server when the user logged out and cancelled all subscriptions.

## 8.5 Summary

In this chapter, the setup that was used for testing the RUCRG IMS client within the RUCRG testbed was presented. The equipment for conducting these tests and the requirements that needed to be satisfied were also provided. The discussion of results was based on the SIP signalling messages exchanged between the RUCRG IMS client and the IMS network (IMS testbed) for the registration and establishment of the session. The SDP content was also carried within these signalling SIP messages, and used to negotiate the type of media used as well as the preconditions for the QoS agreement. The results presented show that the RUCRG IMS client is compliant with the 3GPP technical specifications and IETF SIP recommendations.

# Chapter 9

## Conclusion

SIP used as a signalling protocol provides capabilities to develop real-time multimedia applications over the Internet. The introduction of the IMS has resulted in the enhancement of existing SIP services such as voice/video calls, instant messaging (IM) and presence. It has enabled new multimedia oriented communication services through the integration of telecommunication and data on an access independent IP network. This co-occurrence of data, voice and video has increased the demand for services with new presentation characteristics.

The RUCRG decided to reinforce and upgrade the JAIN SIP Applet Phone (JSAP) to be IMS compliant and create a single client that researchers (RUCRG) can easily adapt to suit their needs as they develop new services. We therefore build our own comprehensive IMS client (easily modifiable and open source) that could be integrated into the infrastructure and services of the RUCRG testbed. This thesis presents the process followed to create such a client (the RUCRG IMS client). Also described in this thesis is the architecture of this client and its current development status and integration with RUCRG testbed.

### 9.1 Synopsis

This thesis outlined the development of a Java based, IMS compliant client called RUCRG IMS client, which was developed using the JSAP as its foundation. JSAP is an open source project which one of the RUCRG members helped to develop. It possesses some of the basic features required in a SIP/IMS compliant client, such as voice and text IM and is used extensively to test SIP applications by researchers in the RUCRG. Unfortunately,

the client only supports SIP applications. The mandate of this research was to upgrade the JSAP into an IMS capable user agent.

Because our IMS client development was based on an existing SIP client, it was necessary to perform an extensive assessment of it. On the one hand, this was done to reinforce understanding of how JSAP worked, particularly because there was very little documentation. On the other hand, this exercise was carried out to validate and verify the already implemented SIP features in the JSAP. After careful analysis we found that the JSAP:

- Uses JAIN SIP (a low level Java API for SIP signalling).
- Supports core SIP signalling.
- Uses JAIN SDP in the manipulation of session descriptions.

Furthermore, the analysis of the JSAP also revealed various errors which we remedied. The main PRACK mechanism was also added to the client in order to support advanced SIP functions, such as playing announcements during early Dialogs.

Having consolidated the SIP functions in JSAP, we needed to completely overhaul the media portion, because of the problems that we identified with JMF (the framework is no longer being supported and some parts of the framework do not work on some OS platforms). Alternative, open source media APIs to the JMF were investigated and Gstreamer (accessed through Gstreamer-Java wrapper) was chosen to replace JMF. Voice and video support was integrated into JSAP using Gstreamer media API. At this point, the client could reliably perform SIP functions as well as receive and stream audio and video using the Gstreamer media library. After these enhancements, we named the client JSAP+.

Presence is important so we overhauled it before integrating IMS functionality. Given that JSAP+ supported some form of SIP based presence which was integrated with IM, we started by reviewing the way JSAP+ handled presence. We found that JSAP+ lacked the mechanisms to store user data in a central repository. User information was stored on the client: JSAP+ lacked the ability to save user data after the program was executed or when the user moved from one device to another. So, we needed the client to support network-based storage. This was achieved by extending the JSAP+ client to make use of XCAP, in the storage and retrieval of user data. With this extension, we renamed the client JSAP++.

Then, we began our work on making the client IMS compliant. In chapter 7, we detailed how JSAP++ was transformed from a basic SIP client to an IMS compliant client, to be

called the RUCRG IMS client. In that chapter, we provided details of the enhancements that we made to JSAP++ to ensure that it could:

- Register with an IMS network.
- Establish IMS sessions.
- Negotiate media codecs during IMS session establishment using the SDP offer/answer mechanism.
- Cancel early IMS sessions using the CANCEL method.
- Exchange voice and/ video with other IMS clients.
- Terminate IMS sessions using the BYE method.

Additionally we detailed how some important IMS headers (SIP extensions for IMS) such as *Route*, *P-Preferred-Identity* and *P-Access-Network-Info* headers were integrated into JSAP++.

Finally we tested the system to demonstrate the functionality of the client. We used a hybrid approach to the testing, in which we combined conformance monitoring and interoperability testing. Communication was established between our client (EUT) and various QEs to verify compatibility, while a network protocol analyser was used to validate that the messages that were exchanged conformed to the standards.

The final outcome of this thesis has been the upgrade of JSAP from a basic SIP client to a SIP and IMS compliant client. The client is now capable of interacting with the FOKUS IMS Core for the setup, control and termination of IMS services and is interoperable with Mercurio IMS client and UCT IMS client. The client also supports ordinary SIP functions and can work with non-IMS SIP proxy servers. It is also interoperable with the following SIP clients: Twinkle, Ekiga, GRANDSTREAM GXV3140 and SJphone. The client can also receive and stream audio and video using the Gstreamer media API. Lastly, the RUCRG IMS client supports network based storage of user data via XCAP using the Mobicents XCAP API. Modularity and intuitiveness were built into the client throughout the development: the client needed to remain simple to extend to allow integration of new features and updating of existing ones as RFCs and standards evolve.

## 9.2 Discussion

The main objective of this thesis was to reinforce the existing SIP functions in the JSAP and upgrade it to be IMS compliant so that it can be used for both SIP and IMS application testing by the RUCRG. The goal was to produce our own client that provides native IMS functionality, supports re-usability of client code, enables service composition/aggregation, and allows easy modification by the RUCRG researchers. These goals were identified by working closely with the RUCRG researchers, gathering a comprehensive list of requirements and incrementally adding functionality to the client. The methodology adopted for the development of the client described in this thesis was progressive prototyping, which involves incrementally adding functionality as required. This client was developed to be compliant with 3GPP, European Telecommunications Standards Institute (ETSI), Telecoms and Internet converged Services and Protocols for Advanced Network (TISPAN) and the Internet Engineering Task Force (IETF) recommendations and specifications. This was done to ensure that the client is interoperable with other clients and servers that follow the same standards. Care was taken not to limit the use of the client to the IMS platform thus making the client backward compatible with legacy SIP servers and applications. Lastly, the client had to be free in terms of cost (use of freely available libraries in development) as well as open source.

### 9.2.1 Achieved Goals

This research was undertaken to develop a unified, cross-platform SIP/IMS client to be used for testing communication services being developed by the RUCRG, while adhering to the specifications and recommendations of the major standardisation bodies.

The developed client is indeed cross-platform. We successfully deployed it on two completely unrelated operating system platforms (Windows Vista and Linux Ubuntu) without the need to modify the source code.

The results of the testing that was done show that the RUCRG IMS client is capable of registering with the IMS network using AKAv1-MD5 as well as register with SIP proxies using MD5. The RUCRG IMS client is now fully offer/answer capable, that is, it is able to establish and terminate SIP/IMS based multimedia sessions and negotiate media codecs (both static and dynamic) using the SDP answer/offer mechanism. Furthermore, the client now has full multimedia capability, that is, it is able to establish voice/video sessions with both SIP/IMS clients using the Gstreamer media API which replaced JMF.

The structure of the client has been improved and the client has been documented allowing for faster and easier modifications.

Finally, the client now uses a network based storage mechanism for storing user data using XCAP. Thus all the objectives of this thesis were met.

### 9.2.2 Challenges

It was not an easy task to harmonise and make sense of the large set of standards that our client needed to be compliant with.

At times, it was also difficult to decide on the subset of features to implement or exclude. We hope that the choices were correct, but naturally we expect other researchers to add the features they need, if they are not implemented.

One important goal was to ensure support to major operating systems and platforms available today. Due to poor multimedia support for Java, this was not an easy task.

Lastly, the client was undocumented which made it difficult to improve/upgrade the client. This resulted in a lot of time being spent on studying the structure of the client.

### 9.2.3 Limitations

The IMS client developed in this study was built according to 3GPP and IETF specifications and recommendations. Although IMS may be considered a mature technology, it is still evolving. Consequently, some of the implemented features may not work in other IMS testbeds (which are still work in progress) running applications based on different 3GPP specifications releases. The RUCRG IMS client was mainly targeted to be used within the RUCRG testbed running the FOKUS IMS Core. The features we tested were functional and compatible with the RUCRG testbed settings, but modifications may be required when using the client in other testbeds.

As one could expect, the study did not evaluate interoperability with all available SIP/IMS clients.

## 9.3 Future Work

There is need to integrate the SIP Event Notification mechanism for subscribing to a homogeneous list of resources as described in RFC 4662 [49] instead of sending individual

SUBSCRIBE requests for each resource. The watcher (subscriber) can then be able to subscribe to an entire list and receive notifications when the state of any of the resources in the list changes.

The Mobicents XCAP client API that was used to add XCAP support in JSAP is an incomplete implementation of the XCAP protocol as highlighted in chapter 6. Future work should complete and standardise this section.

Our client only uses presence information together with IM. A third aspect that can be enhanced, at the interface level, is to integrate presence with other forms of communication services such as voice, video, and file sharing.

## 9.4 Summary

This chapter provided a summary of the work done in this thesis. It reported how JSAP was modified to consolidate SIP functions and integrate XCAP to support presence. The chapter also details the work done to transition JSAP from a basic SIP client to an IMS compliant client called RUCRG IMS client.

At the end of this journey the Rhodes University Convergence Research Group now has an advanced, robust and easily modifiable IMS compliant client.

# Bibliography

- [1] 3GPP. TS 27.060 v3.8.0: Technical Specification Group Core Network; Packet Domain; Mobile Station (MS) supporting Packet Switched Services (Release 1999) . Third Generation Partnership Project, June 2003.
- [2] 3GPP. TS 24.228: Signalling flows for the IP multimedia call control based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3. Third Generation Partnership Project, October 2006.
- [3] 3GPP. TS 33.203: 3G security; Access security for IP-based services layer security. Third Generation Partnership Project, December 2010.
- [4] 3GPP. TS 23.218: IP Multimedia (IM) session handling; IM call model; Stage 2. Third Generation Partnership Project, December 2011.
- [5] 3GPP. TS 23.228: IP Multimedia Subsystem (IMS); Stage 2. Third Generation Partnership Project, December 2011.
- [6] 3GPP. TS 24.229: IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3. Third Generation Partnership Project, June 2011.
- [7] 3GPP. TS 33.210: 3G security; Network Domain Security (NDS); IP network layer security. Third Generation Partnership Project, December 2011.
- [8] Arup Acharya, Nilanjan Banerjee, Dipanjan Chakraborty, and Shachi Sharma. Presentials: a flexible middleware for presence-enabled applications. In *Proceedings of the 5th International Conference on Principles, Systems and Applications of IP Telecommunications*, IPTcomm '11, pages 15:1–15:12, New York, NY, USA, 2011. ACM.
- [9] Andreas Bachmann, Alice Motanga, and Thomas Magedanz. Requirements for an extendible ims client framework. In *Proceedings of the 1st international conference*



- on *MOBILE Wireless MiddleWARE, Operating Systems, and Applications*, MOBIL-WARE '08, pages 19:1–19:6, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).
- [10] G. Camarillo. *SIP Demystified*. McGraw-Hill Companies Inc, first edition, 2002.
- [11] G. Camarillo and M.A. Garcia-Martin. *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*. John Wiley and Sons Ltd, third edition, 2008.
- [12] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session Initiation Protocol (SIP) Extension for Instant Messaging. RFC 3428 (Proposed Standard), December 2002.
- [13] M. Day, J. Rosenberg, and H. Sugano. A Model for Presence and Instant Messaging. RFC 2778 (Informational), February 2000.
- [14] M. Diop. Inside IMS/NGN. Website, 2009. <http://betelco.blogspot.com/2009/02/new-version-of-mercuro-ims-client.html>.
- [15] L. Etiemble and M. Diop. Mercuro IMS Client. Website, August 2010. <http://imsclient.blogspot.com/>.
- [16] ETSI. The Global Testing Language. Available Online, 2006. [etsi.org/WebSite/document/Technologies/LEAFLETS/TheGlobalTestingLanguage%28TTCN3%29.pdf](http://etsi.org/WebSite/document/Technologies/LEAFLETS/TheGlobalTestingLanguage%28TTCN3%29.pdf).
- [17] ETSI. ETSI EG 202 237: Methods for Testing and Specification (MTS); Internet Protocol Testing (IPT); Generic approach to interoperability testing. Available Online, April 2007. [http://www.etsi.org/deliver/etsi\\_eg/202200\\_202299/202237/01.01.02\\_60/eg\\_202237v010102p.pdf](http://www.etsi.org/deliver/etsi_eg/202200_202299/202237/01.01.02_60/eg_202237v010102p.pdf).
- [18] FFMPEG. Ffmpeg. Website, March 2010. <http://www.ffmpeg.org/>.
- [19] FFMPEG-Java. Getting Started with FFMPEG-Java - FMJ. Website. [http://fmj-sf.net/ffmpeg-java/getting\\_started.php](http://fmj-sf.net/ffmpeg-java/getting_started.php).
- [20] FMJ. Community News - Freedom for Media in Java (FMJ) Project Releases 0.1 Version. Internet, June 2006. <http://www.artima.com/forums/flat.jsp?forum=276&thread=164831>.

- 
- [21] FMJ. Home FMJ. Internet, October 2007. <http://fmj-sf.net/>.
- [22] M. Garcia-Martin, E. Henrikson, and D. Mills. Private Header (P-Header) Extensions to the Session Initiation Protocol (SIP) for the 3rd-Generation Partnership Project (3GPP). RFC 3455 (Informational), January 2003.
- [23] P. Gregory. *SIP Communications for Dummies*. Wiley Publishing, Inc, second edition, 2006.
- [24] Gstreamer-Java. Gstreamer-java Java Interface to the Gstreamer Framework. Website, May 2010. <http://code.google.com/p/gstreamer-java/>.
- [25] M. Handley and V. Jacobson. SDP: Session Description Protocol. RFC 2327 (Proposed Standard), April 1998. Obsoleted by RFC 4566, updated by RFC 3266.
- [26] M. Handley, V. Jacobson, and C. Perkins. SDP: Session Description Protocol. RFC 4566 (Proposed Standard), July 2006.
- [27] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session Initiation Protocol. RFC 2543 (Proposed Standard), March 1999. Obsoleted by RFCs 3261, 3262, 3263, 3264, 3265.
- [28] E. Ivov. SIP Communicator. Slides, 2007. [jres.org/planning/slides/132.pdf](http://jres.org/planning/slides/132.pdf).
- [29] java.net. Jain-sip-applet-phone. Website, 2010. <https://java.net/projects/jain-sip-applet-phone>.
- [30] java.net. JAIN SIP API. Website, 2011. <http://jsip.java.net/>.
- [31] C. Kaufman. Internet Key Exchange (IKEv2) Protocol. RFC 4306 (Proposed Standard), December 2005. Obsoleted by RFC 5996, updated by RFC 5282.
- [32] K. I. Lakhtaria. Chapter - 6 IMS Client Development and Deployment. Online PDF, 2010. [shodhganga.inflibnet.ac.in/bitstream/10603/734/11/11\\_chapter6.pdf](http://shodhganga.inflibnet.ac.in/bitstream/10603/734/11/11_chapter6.pdf).
- [33] Z. Lei and P. Coulton. IMS Based Mobile Presence Service. In *Future Mobile Experiences: next generation mobile interaction and contextualization, Workshop at NordiCHI*, 2008.
- [34] Tian Li, Zhiliang Wang, and Xia Yin. Sip conformance testing based on ttcn-2. *Tsinghua Science & Technology*, 12, Supplement 1(0):223 – 228, 2007.
- [35] LTI-CIVIL. LTI - CIVIL. Internet, October 2007. <http://lti-civil.org/>.

- [36] L. Luo. Software Testing Techniques Technology Maturation and Research Strategy Class. Technical report, Institute for Software Research International Carnegie Mellon University Pittsburgh, PA15232 USA. <http://www.cs.cmu.edu/~luo/Courses/17939Report.pdf>.
- [37] A. Mankin, S. Bradner, R. Mahy, D. Willis, J. Ott, and B. Rosen. Change Process for the Session Initiation Protocol (SIP). RFC 3427 (Best Current Practice), December 2002. Obsoleted by RFC 5727, updated by RFCs 3968, 3969.
- [38] R. Marston. Multimedia Content Adaptation for IPTV Services in IMS. Master's thesis, University of Cape town, 2008.
- [39] M.T. Masonta. Development of Light-Weight IP Multimedia Subsystem (IMS) Client for Mobile Devices. Master's thesis, Tshwane University of Technology, May 2008.
- [40] A. Niemi, J. Arkko, and V. Torvinen. Hypertext Transfer Protocol (HTTP) Digest Authentication Using Authentication and Key Agreement (AKA). RFC 3310 (Informational), September 2002.
- [41] NMSCcommunications. SIP/IMS Client Applications for Operators, Terminal Vendors, and Equipment Vendors. Whitepaper, May 2010. <http://www.nmscommunications.com/DevPlatforms/WhitePapers/default.htm>.
- [42] Nokia. 3GPP2 X32-20050926-0aa: Call flow updates. Available Online, September 2005. 3GPP2 meeting 2005 September, Vancouver.
- [43] E. Oguejiofor, P. Bazot, B. Georges, R. Huber, C. Jackson, J. Kappel, M. Cameron, and Abhijit Sur. Bala, S. Subramanian. *Developing SIP and IP Multimedia Subsystem IMS Applications*. IBM.
- [44] Oracle. Oracle press release, January 2010. <http://www.oracle.com/us/corporate/press/044428>.
- [45] R.M. Perea. *Internet Multimedia Communications Using SIP: A Modern Approach Including Java Practice*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2008.
- [46] M. Poikselka and G. Mayer. *The IMS: IP Multimedia Concepts and Services*. John Wiley and Sons Ltd, third edition, 2009.
- [47] Harald Psailer. A Java-Based Streaming Media Server. Master's thesis, Vienna University of Technology, 2005.

- 
- [48] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265 (Proposed Standard), June 2002. Updated by RFCs 5367, 5727.
- [49] A. B. Roach, B. Campbell, and J. Rosenberg. A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists. RFC 4662 (Proposed Standard), August 2006.
- [50] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856 (Proposed Standard), August 2004.
- [51] J. Rosenberg. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). RFC 4825 (Proposed Standard), May 2007.
- [52] J. Rosenberg and H. Schulzrinne. An Offer/Answer Model with Session Description Protocol (SDP). RFC 3264 (Proposed Standard), June 2002.
- [53] J. Rosenberg and H. Schulzrinne. Reliability of Provisional Responses in Session Initiation Protocol (SIP). RFC 3262 (Proposed Standard), June 2002.
- [54] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), June 2002. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621, 5626, 5630, 5922, 5954, 6026, 6141.
- [55] H. Schulzrinne and S. Casner. RTP Profile for Audio and Video Conferences with Minimal Control. RFC 3551 (Standard), July 2003. Updated by RFC 5761.
- [56] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: A Transport Protocol for Real-Time Applications. RFC 3550 (Standard), July 2003. Updated by RFCs 5506, 5761, 6051, 6222.
- [57] H. Schulzrinne and B. Volz. Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers. RFC 3319 (Proposed Standard), July 2003.
- [58] Sourceforge. Fobs: C++ wrapper for ffmpeg. Website. <http://fobs.sourceforge.net/>.
- [59] Sun Microsystems. Java Media Framework. Website. <http://java.sun.com/products/java-media/jmf/>.
- [60] W. Taymans, S. Baker, A. Wingo, R. Bultje, and S. Kost. GStreamer Application Development Manual(0.10.25.3). Article, June 2010. <http://gstreamer.freedesktop.org/documentation/>.

- 
- [61] UCT. UCT IMS Client. Website, June 2009. <http://uctimsclient.berlios.de/>.
- [62] D. Waiting, R. Good, R. Spiers, and N. Ventura. The UCT IMS Client. IEEE, 2009.
- [63] Wikipedia. Java Media Framework. Website. [http://en.wikipedia.org/wiki/Java\\_Media\\_Framework](http://en.wikipedia.org/wiki/Java_Media_Framework).
- [64] Wikipedia. libavcodec. Website. <http://en.wikipedia.org/wiki/Libavcodec>.
- [65] D. Willis and B. Hoeneisen. Session Initiation Protocol (SIP) Extension Header Field for Service Route Discovery During Registration. RFC 3608 (Proposed Standard), October 2003. Updated by RFC 5630.

# Appendix A

## Accompanying CD-ROM

The following are contained within the accompanying CD-ROM:

- Thesis document
- Client source code (includes all the required supporting libraries to run the client in the `gov.nist.applet.phone.libraries` package)
- Gstreamer for Windows

# Appendix B

## Deployment Guide

The client comes as a Netbeans project and can be run directly if the following conditions are met:

1. The Java SDK is installed.
2. When running the client on Linux Ubuntu the following Gstreamer packages need to be installed:
  - Gstreamer ffmpeg video plugin
  - Gstreamer extra plugins
  - Gstreamer plugins for mms, wavpack, quicktime and musepack
  - Gstreamer plugins for aac, xvid, mpeg2 and faad
3. When running the client on Windows, Gstreamer WinBuild package (included in the CD-ROM) needs to be installed.