Chapter – 6
IMS Client Development and Deployment

Chapter – 6
IMS Client Development and Deployment

6.1 Introduction

IP Multimedia Subsystem introduced since 2002, since initiation of IMS many research work done for designing and implementing IMS-capable terminals and network elements. [123] Bubley has conducted a research into the evolution of SIP and IMS handset. “While much attention has focused on deployments of IMS network infrastructure and applications, the need for a new class of phones has been largely forgotten”.

Before initiate design own IMS Client, an in depth studied the three open IMS Clients, which were mainly targeted to be implemented within the end user devices that have high processing power and memory (e.g. PDA Phone). These end user devices may include laptops, desktop personal computers or Personal Digital Assistance (PDA). Hence the three open IMS Clients may not be deployed within the current CLCD based handheld mobile devices.

Based on the 3GPP IMS [1], the IMS compliant end user has to provide the necessary SIP signaling support, the service related media codecs for multimedia applications in addition to the basic connectivity support, such as IP [125]. Current open IMS Clients do provide these supports to the end users with higher processing power and memory devices.

In this study, the IMS Client that is developed & deployed under the Java Micro-Edition (J2ME) platform with SIP. Though developed within the J2ME platform, this client should be capable of providing all the IMS supports as specified by the 3GPP and IETF standards and recommendations. As a result, this study aims to developing an IMS Client that can be deployed within the J2ME-compliant handheld mobile devices.

6.2 Other IMS Clients

Currently there are three open IMS Clients available to the research community. Among these three IMS Clients, only one does not have free source code for download, as it is just open for use. The other two are open in the sense that developers can download their source codes and extend them. Although these IMS Clients have not targeted the handheld mobile devices, one of them FOKUS OpenIC Lite [124] can be deployed in some PDAs.

6.2.1 FOKUS Open IMS Client Lite

The FOKUS group has developed an IMS client, which they call Open IMS Client (OpenIC) Lite, shown in Figure 6.1. Work on FOKUS OpenIC Lite is progressing towards a robust IMS client with diverse services for Fixed Mobile convergence [125]. The solutions delivered by the framework align with IETF [5], 3GPP [2] and TISPAN client specifications. The client is available in Java and in .NET and runs on multiple platforms like Windows Mobile, Linux & Windows XP.

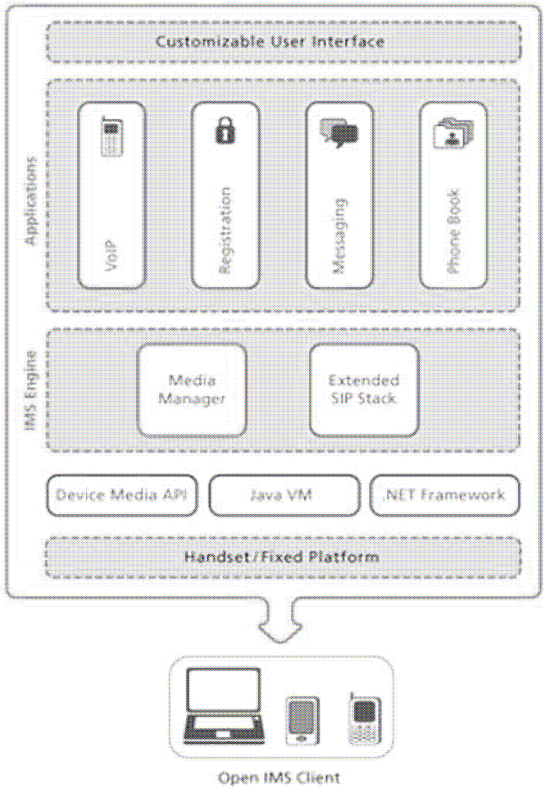

Figure 6.1: FOKUS OpenIC Lite [125]

While OpenIC Lite is free and available for download, the FOKUS group has a commercial Open IMS Client version called OpenIC. "Open" in this sense means extendable therefore it should not be confused with "Open Source" [125].

6.2.2 IMS-Communicator

IMS-Communicator is an open source project that was originally developed by PT Inovafao to support the development and testing of IMS and NGN components for its Service Handling on IP Networks (shipnet®) IMS/NGN architecture [126]. The IMS-Communicator, shown in Figure 2.15, is based on the old version of the S IP-Communicator soft-phone project. It is built on top of the JAIN-SIP reference implementation, and makes use of the Java Media Framework (JMF) API as its media stack.

Figure 6.2: IMS-Communicator Screenshot [126]

According to the IMS-Communicator website, the developers of IMS-Communicator have no plans for porting it to the J2ME Connected Device Configuration (CDC) environment. As a result, this IMS Client cannot be deployed to mobile devices. The Apache Software License and the GNU Lesser General Public License (LGPL) license the IMS-Communicator project. Its hosting is provided by the BerliOS developers.

6.2.3 UCT IMS Client

University of Cape Town (UCT) IMS Client is another open source IMS Client hosted by the BerliOS developers [127]. The communications research group at UCT in South Africa has developed the UCT IMS Client, and it designed to be used in conjunction with the Fraunhofer FOKUS Open IMS Core network testbed. On the UCT website the following is stated about the UCT IMS Client "This project is still in active development and there are many known bugs" [127].

The UCT IMS Client screenshot is shown in Figure 6.3. The UCT IMS Client has been developed in C++ language and is only available for Linux operating systems. It is not available for mobile devices.

The UCT IMS Client [128] was developed as a flexible and extensible tool for the furthermost of research into IMS networks and services. Since its first release it has seen a great deal of improvements and additional features. It has been used extensively worldwide as a tool to understand the basics of IMS as well as a code-base on which new IMS applications are developed.

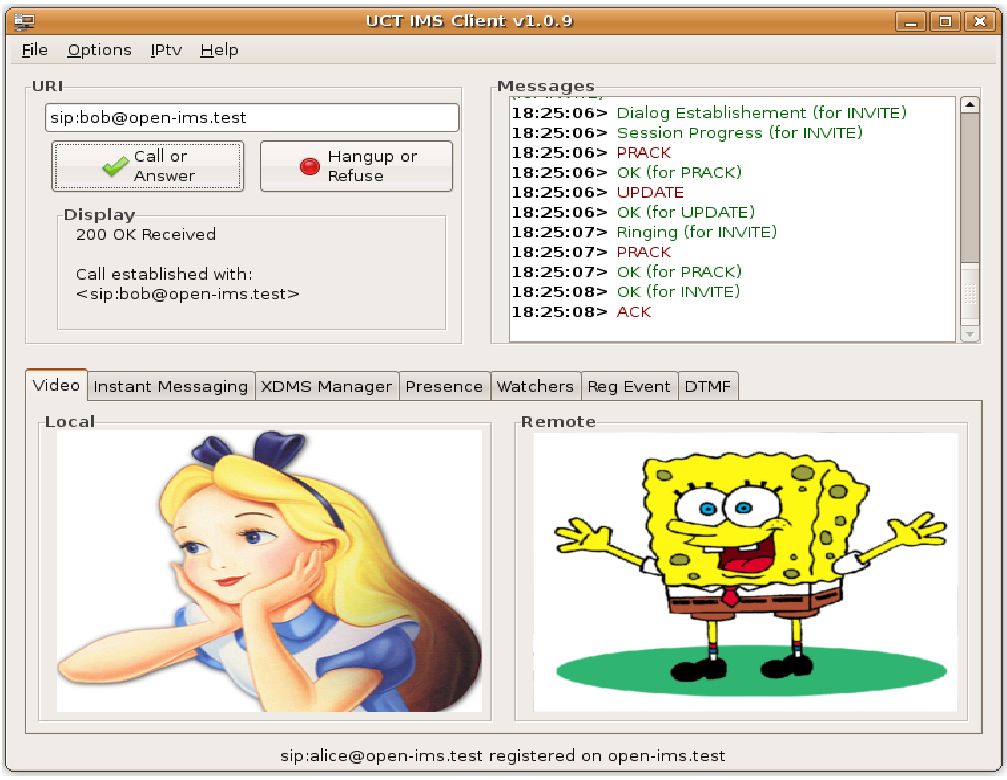

Figure 6.3: UCT IMS Client Screenshot [127]

6.3 Overview of Tools Used For Developing IMS Client

The development of the IMS Client based on Java 2 Micro-Edition (J2ME) platform. The development software and hardware tools used for rich application portable devices are presented further in this chapter, introduces and discusses the J2ME as the chosen development platform for the IMS Client, the Java libraries or Application Programming Interfaces (APIs) used to support the SIP signalling.

6.3.1 Development Tools

Different software and hardware tools and libraries are available for use by developers to develop rich applications for portable devices such as mobile phones, set-top boxes, PDAs and navigation systems, Mao, G.F., Talevski, A. & Chang, E [129] have compared the most popular three mobile development platforms, and presented their results as shown in Table 6.1.

Due to its extensive market penetration and developer community support, J2ME development platform was chosen for the development of the IMS Client.

Table 6.1: Mobile Development Platforms [129]

Functionality	Symbian	J2ME	BREW
Foundation	C++	Java	C++
Learning Curve	Difficult	Average	Difficult
Emulator	Free	Free	Limited
Debuggers Available	Latest version- good	Excellent	Needs payment
Development Tool Cost	Varies	Free	Expensive
Cross-Platform Deployment	Compile per target	Average	CDMA handsets only
Developer Community and Support	Extensive	Extensive	Limited
Market Penetration	Extensive	Extensive	In few countries
SIP Support	Yes	Yes	Unknown
RTP Support	Yes (complicated)	No	Unknown

6.3.1.1 Java Development Kit (JDK)

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 Sun website. For the development of the IMS Client, in this study, JDK 1.5 was used.

6.3.1.2 J2ME Sun Wireless Toolkit

The Sun Wireless Toolkit (WTK) is the reference implementation of J2ME. It has building tools, utilities and a device emulator for creating J2ME Java applications, which are compliant to Mobile Information Device Profile (MIDP). The Sun WTK provides the byte-code pre-verification tool, implementation of API class libraries and a smart-phone device emulator by Goyal [130]. KToolbar is the main Sun WTK component, which is used to build MIDlets, to launch the emulator, and to start the utilities provided by the Sun WTK. Java applications that run in the MIDP environment are called MIDlets.

The IMS Client is developed as a MIDlet and emulated using the Nokia smart-phone emulator provided with the Sun WTK version 2.5. Version 2.5 of the Sun WTK is compliant with Java Technology for the Wireless Industry (JTWI) JSR 185 and Mobile Service Architecture (MSA) JSR 248 specifications for mobile devices, as shown in Figure 6.4.

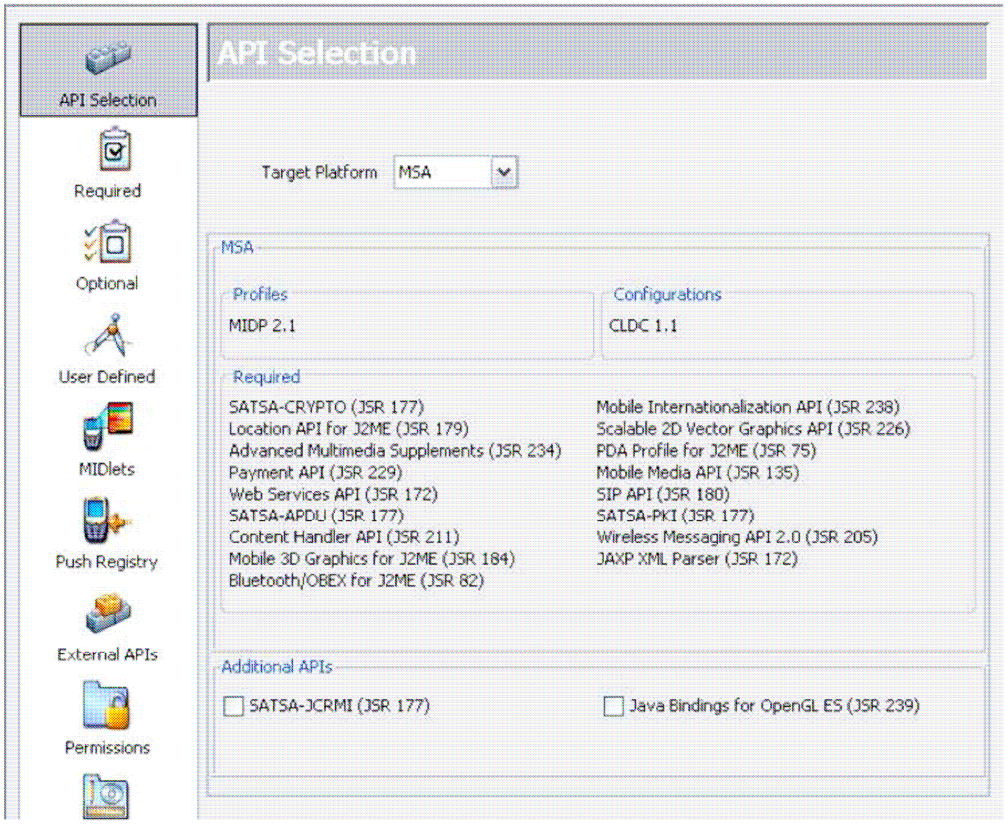

Figure 6.4: Sun Java WTK 2.5 API Selection Window

From the API Selection window, one can decide whether to use the JTWI or MSA target development platform. Many of today's Java enabled wireless mobile devices are mainly based on Connected Limited Device Configuration (CLDC) and MIDP; with the latest mobile devices being compliant with Java MSA API (JCP JSR 248, 2006). Such devices include Nokia N81, N91, N70 series and Sony-Ericsson Z750.

6.3.2 EclipseME

Although the Sun WTK provides a powerful set of J2ME development tools, it does not offer the Integrated Development Environment (IDE) experience. For developers who are used to the convenience of IDE, the Sun WTK lacks key productivity features, such as team collaboration, context-sensitive code editors, and debugging and project management. In order to provide the IDE experience to developers, the Eclipse IDE [195] can be extended with the

EclipseME plug-in to enable the integration between the Eclipse and the Sun WTK.

In order to integrate both the Eclipse IDE and Sun WTK, both tools must be downloaded and installed in the same machine. The EclipseME plug-in makes use of the Sun WTK emulator during runtime, while Eclipse IDE is used as a text editor.

6.3.3 JAVA 2 MICRO-EDITION PLATFORM

Java was introduced in 1995, but its platform capabilities took off with the advent of servlets. Servlets are Java programs that run on a server, offering a modular and efficient replacement for Common Gateway Interface (CGI). The first revolution of Java occurred when it expands into the server side with Java 2 Platform, Enterprise Edition (J2EE). J2ME is the second revolution in Java that targets small, mobile devices.

Today, the Java language defines three platforms: Java 2 Platform, Standard Edition (J2SE), J2EE and J2ME. J2SE is designed for desktop computers. Most often it runs on Linux, Solaris or Microsoft Windows operating systems. J2EE is a comprehensive platform for multi-user, enterprise-wide applications. It is based on J2SE and incorporates APIs for server-side computing. J2ME is a set of technologies and specifications developed for small devices like pagers, mobile phones and set-top boxes. J2ME uses subsets of J2SE components, such as smaller virtual machines and leaner APIs. J2ME is divided into configurations and profiles, as well as optional APIs, which provide specific information about APIs and different families of devices. In the following subsections the structure of J2ME is discussed.

6.3.4 Configurations

A configuration is designed for a specific kind of device based on memory constraints and processor power. It specifies a Java Virtual Machine (JVM) that can be easily ported to devices supporting the configuration. Device manufactures are responsible for porting a specific configuration to their devices. Currently there are two configurations, Connected Device Configuration (CDC) and the Connected, Limited Device Configuration (CLDC). The configurations and profiles of J2ME are generally described in terms of their memory capacity [131]. The minimum amounts of volatile and non-volatile memory are specified.

i) Connected Device Configuration (CDC)

A CDC device has a minimum Read-Only Memory (ROM) of 512 KB, Random Access Memory (RAM) of 256 KB and some kind of network connection (Li & Knudsen, 2005). It is designed for devices like TV set-top boxes, car navigation systems and high-end PDAs. The CDC specifies that a full JVM must be supported.

ii) Connected, Limited Device Configuration (CLDC)

CLDC is designed for smaller devices than those targeted by the CDC. Such devices include mobile phones, pagers and PDAs. The CLDC devices have limited display size, limited memory, limited CPU power, and limited input, limited battery life and limited network connection. CLDC devices have 160 KB to 512 KB of total memory, including a minimum of 160 KB of ROM and 32 KB of RAM available for the Java platform [131].

The reference implementation of the CLDC is based on a small JVM called the KVM. The name derives from the fact that it is a JVM of which the size is measured in kilobytes rather than megabytes [131]. The KVM cannot do everything a JVM does in the J2SE world because of it is too small.

6.3.5 Profiles

A Profile is based on a configuration and provides additional APIs, such as user interface, persistent storage and whatever is necessary to develop running applications for the device. It is layered on top of a configuration, adding the APIs and specifications needed to develop applications for a specific family of devices discussed by Li & Knudsen [131].

The Foundation Profile is a specification for devices that can support a rich networked J2ME environment, as depicted in Figure 6.5. It does not support a user interface. Other profiles can be layered on top of it to add user interface support and other functionality.

The PDA Profile (PDAP) is designed for palmtop devices with a minimum of 512 KB combined ROM and RAM, and a maximum of 16 MB. It includes an application model based on MIDlets, but uses a subset of the J2SE Abstract Windowing Toolkit (AWT) for graphic user interface (GUI).

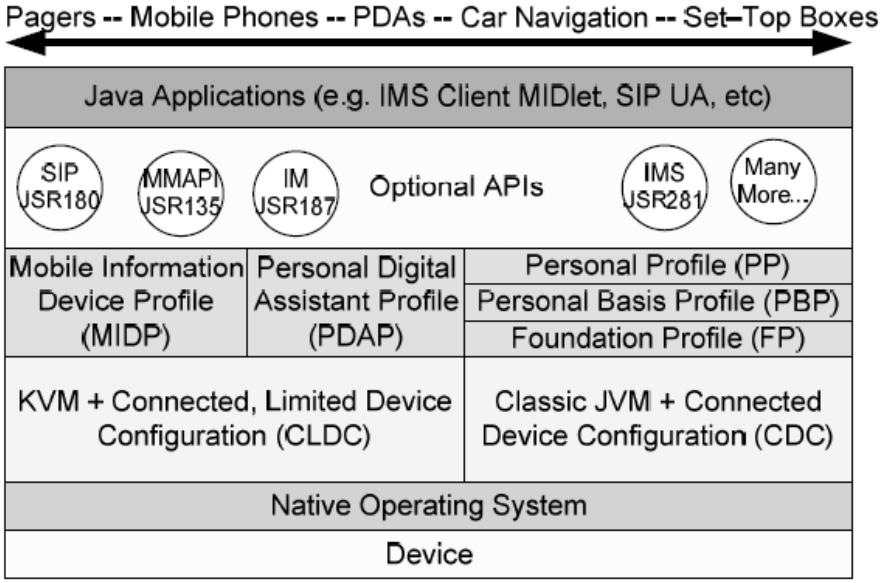

Figure 6.5: J2ME Stack [131]

MIDP complements the CLDC *by* minimizing both the memory and power required for limited devices. It also provides the basic API that is used for creating applications for these devices. The Java Community Project (JCP) has released two versions of MIDP JSR so far (MIDP 1.0 as JSR 37 and MIDP 2.0 as JSR 118). The JSR-248 MSA is a J2ME platform umbrella specification that defines the next step in the Java platform evolution for mobile handsets, based on the CLDC [132].

6.4 JAVA Libraries For SIP Programming

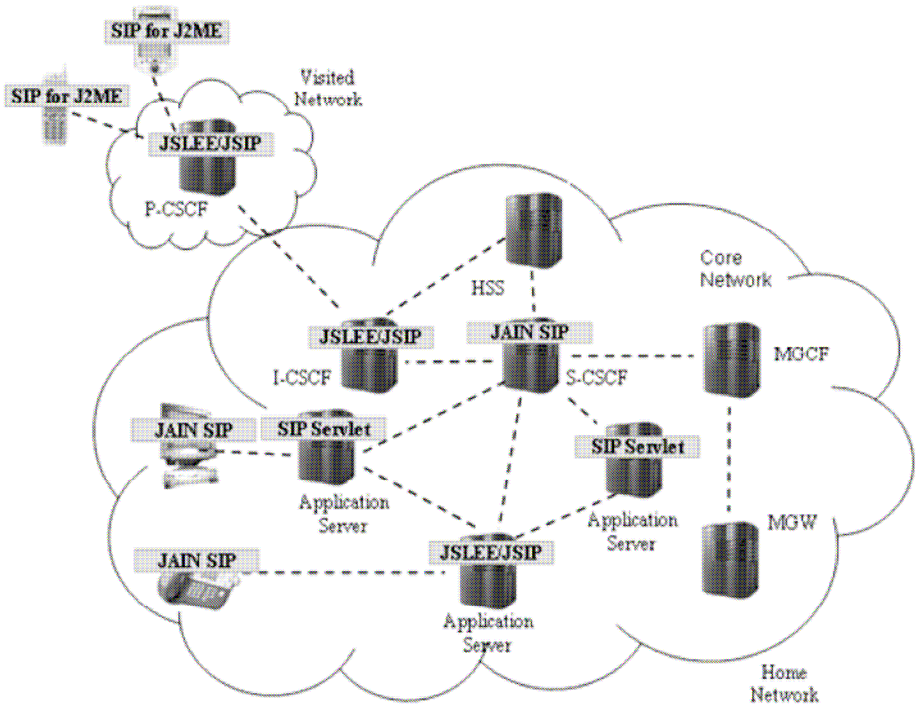

Figure 6.6: IMS and the applicability of the SIP specifications [133]

The SIP APIs being standardized for the Java platform may be viewed as overlapping in functionality with regard to the type of applications that can be implemented within each specification [133], as seen in Figure 6.6 for rapid development of services, programmers and application developers need APIs.

The selection of IETF SIP to become the core signalling protocol for the 3GPP IMS has resulted in the extensive use of the SIP APIs as the interface towards IMS. SIP API for J2ME has been extensively used in the development of the IMS Client for this project, therefore is briefly described in the following subsection.

6.4.1 SIP API for J2ME

Nokia Corporation is the specification lead within the JCP in the development of the SIP API for J2ME, which was given the JSR number 180 [134]. JSR 180 API was designed to be a compact and generic SIP API, which provides SIP functionality in transaction level. The API is integrated into the Generic Connection Framework (GCF) defined in CLDC. It is designed as an optional package that can be used with many J2ME Profiles. The minimum platform required by this API is the J2ME CLDC version 1.0; The API can also be used with the J2ME CDC [134].

The JSR 180 API inherited the Connector and Connection Java interfaces, as shown on the class diagram of the API in Figure 6.7.

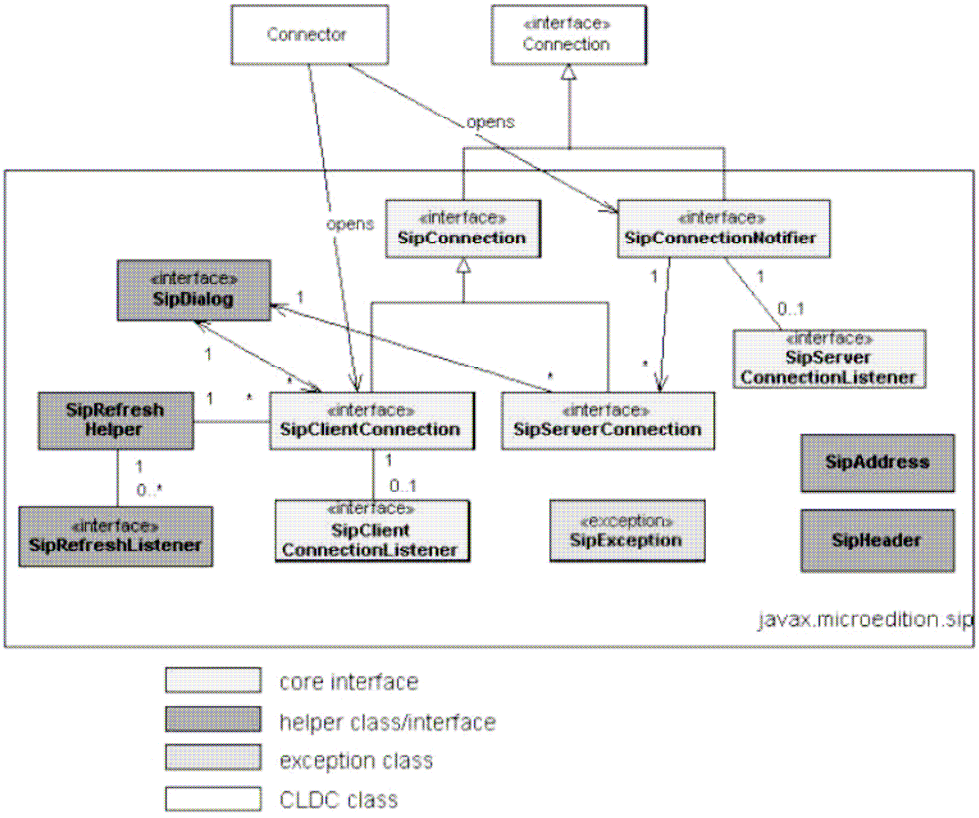

Figure 6.7: Simplified class diagram of SIP API for J2ME [134]

The SIP API is used *by* applications to implement SIP UA functionality. This includes both UAC and UAS functionalities. A single SIP UA uses both the client and server connection functionality for SIP signalling and communication.

i) Javax.microedition.sip Package

This is the package defined by the JSR 180 API. It is also the main package used to implement this project's IMS Client. In this subsection, some of the

major components of this package will be defined, but the full specification of the package can be found on the Java Community Process (JCP) website.

- ***SipConnection*** - this is the base interface for SIP connections. It holds the common properties and methods for the following sub-interfaces: SipClientConnection and SipServerConnection
- ***SipClientConnection*** - this interface represents SIP client transaction. Applications can create a new SipClientConnection with Connector or SipDialog object. This interface is also used to initialise and send responses.
- ***SipSeverConnection*** - this interface represents SIP server transaction. The SipConnectionNotifier creates the transaction when a new request is received.
- ***SipConnectionNotifier*** - this interface defines a SIP server connection notifier, which is opened with Connector.openQ using a SIP URI string with the host and user omitted. This interface queues the received messages. To receive incoming requests, application calls the acceptAndOpenQ method, which returns a SipServerConnection instance.
- ***SipClientConnectionListener*** - this is the listener interface for incoming SIP responses.
- ***SipServerConnectionListener*** - this is the listener interface for incoming SIP requests.

As shown in Figure 6.8, the process of sending a message is twofold. Firstly the message is the prepared and sent. Secondly, the response is processed.

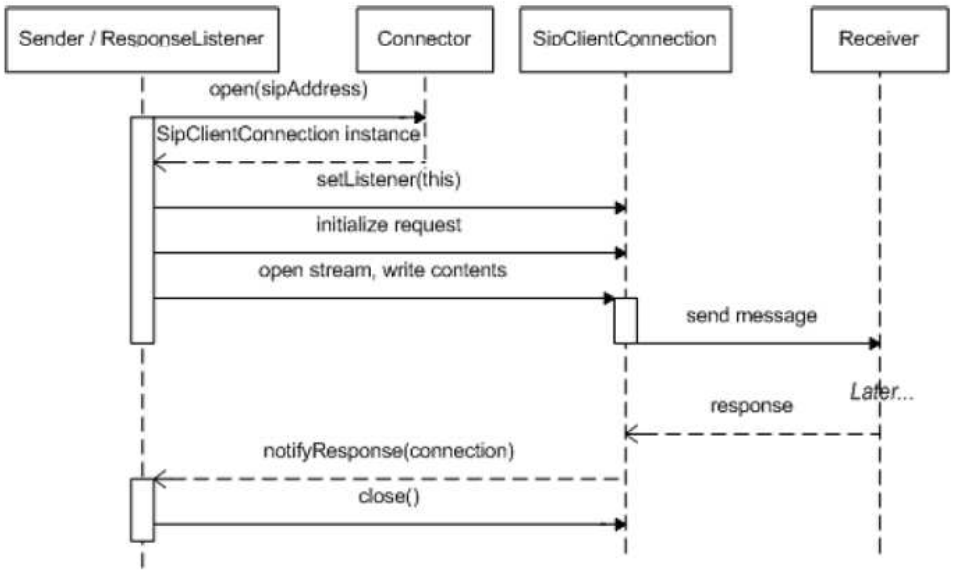

Figure 6.8: Sending SIP request [134]

There are two ways to process an incoming request using the JSR 180 API. The first method uses the synchronous approach, which involves blocking the current thread to wait for one to arrive. This method works well if it is known that the request will be received at a certain time. The second method is to open a permanent server connection and be notified when a message arrives asynchronously. This is sometimes referred to as the callback loop, as shown in Figure 6.9.

Due to the flexibility of SIP, the javax.microedition.sip package was extended in this project to allow other IMS based headers and methods to be exchanged. 3GPP and IETF collaborated to ensure that the core SIP (RFC 3261) is extended to comply with the needs of the IMS. Such SIP extensions were released in the form of RFC documents, and by making use of these RFCs, the traditional SIP APE was enhanced to perform most of the IMS session control.

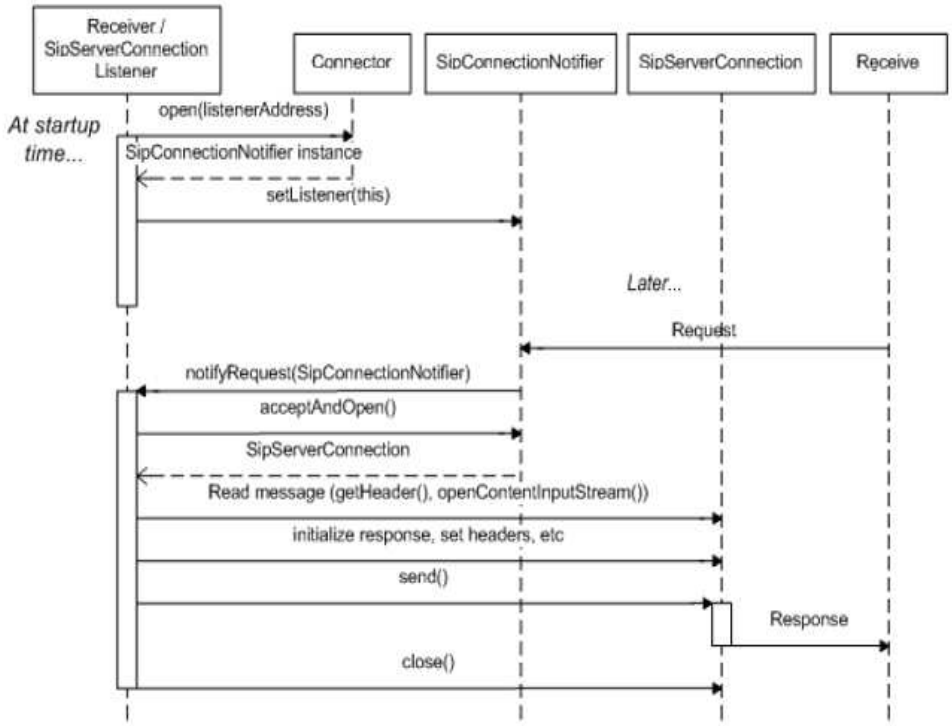

Figure 6.9: Receiving a SIP request [134]

6.5 IMS Client Development

A complete IMS Client is made up of the User Agent Client (UAC) side and the User Agent Server (UAS) side, and its stack is shown in Figure 6.10. While UAC side generates the requests and processes the responses, the UAS side processes the requests and then generates the responses.

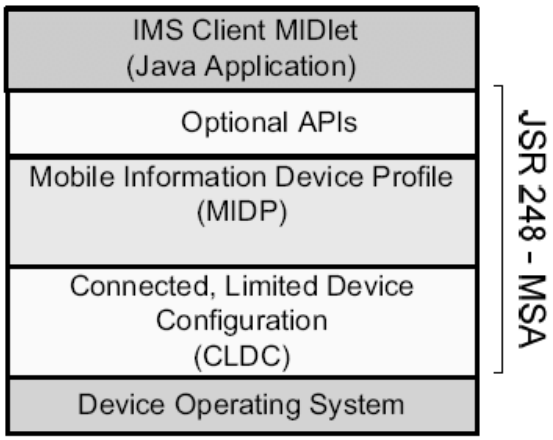

Figure 6.10: IMS Client Stack (Based on [131])

Depicted in Figure 6.11 is the position of a proposed IMS Client on the existing IMS core network and the access network. Access network can be any IPv4-connected network such as mobile 3G networks or wireline Asynchronized Digital Subscriber Loop (ADSL) link.

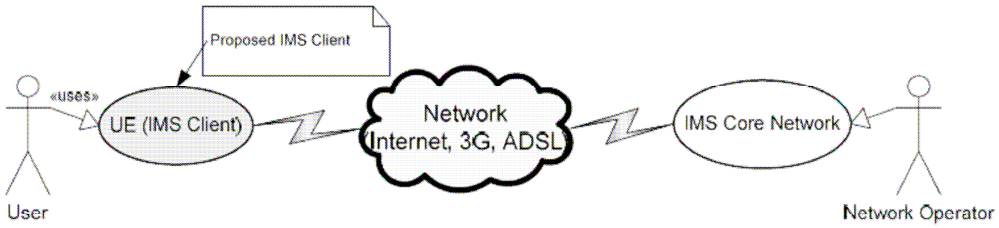

Figure 6.11: IMS Client position in relation to IMS CN

The main flow diagram of the UAC (User Agent Client) side of IMS Client is shown in Figure 6.12. The diagram shows the flow of procedures and events to be performed by the IMS Client after a successful registration, the creation of the session and the ending of the session.

The main flow diagram of the UAC side of IMS Client is shown in Figure 6.12. The diagram shows the flow of procedures and events to be performed by the IMS Client after a successful registration, the creation of the session and the ending of the session.

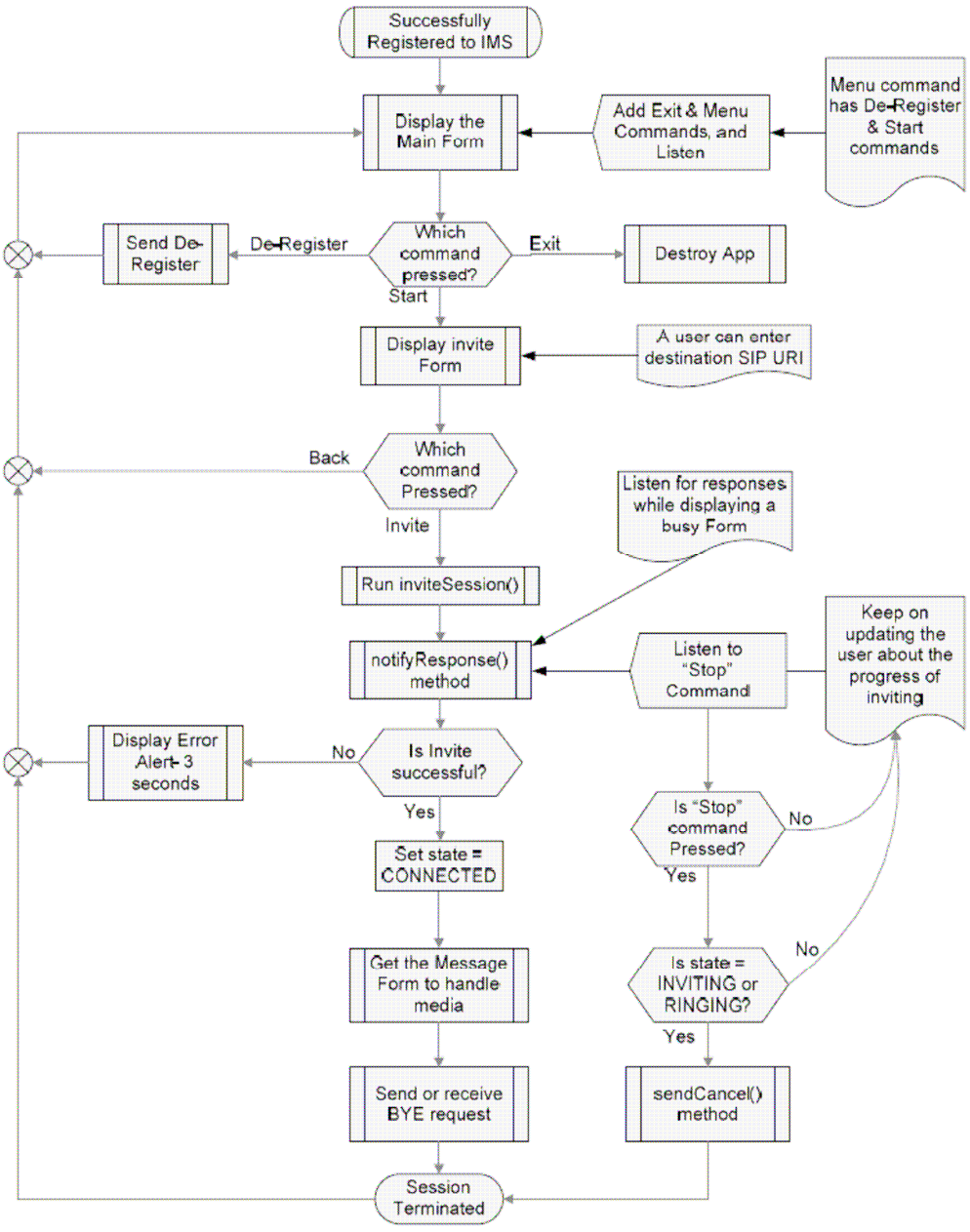

Figure 6.12: Session Establishment Main Flow diagram

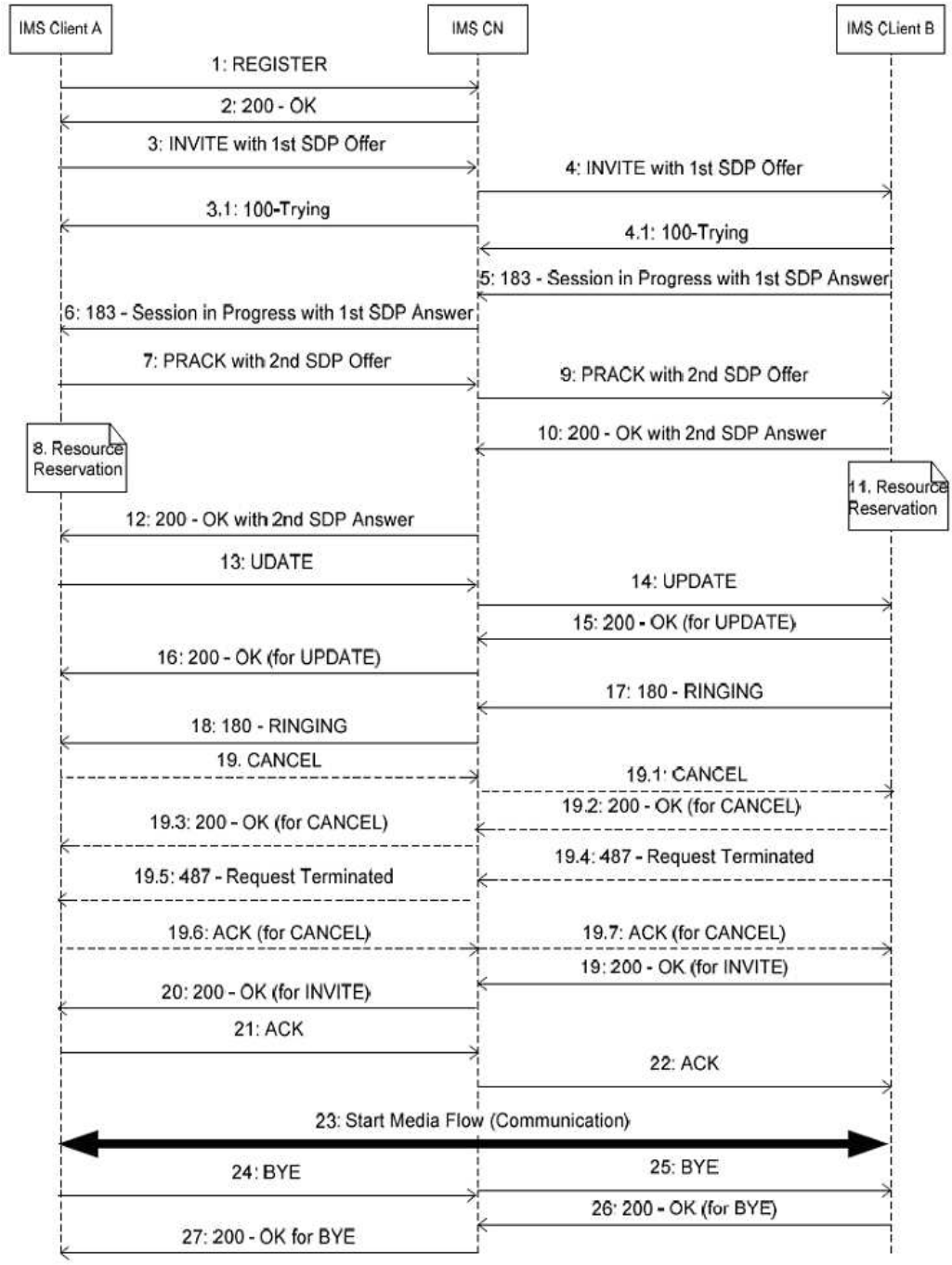

Figure 6.13: IMS Client Sequence Diagram [135]

The system is designed to perform the following functions, as shown in Figure 6.13:

- Registering on the IMS Core Network
- Establishing a session using INVITE method
- Negotiating media codex and QoS through SDP offer/answer

mechanism

- Terminating a session
- Exchanging text messages (Instant Messaging)
- Cancelling the early session using CANCEL method
- Ending the session using BYE request

6.5.1 SIP Registration

Each IMS subscriber needs to be registered to his/her home IMS CN to access the IMS services. Making use of the SIP API (JSR 180), a SIP REGISTER request is initiated with the important headers, and the connection is opened to the home domain registrar's URI or IP address:

```
private SipClientConnection connection = null;
s_cscf = "sip:scscfSregistrar.domain:5060;transport=udp";
connection = (SipClientConnection)
Connector.open(s_cscf);
connection.initRequest("REGISTER", null);
connection.setHeader("Route", p_cscfAddress);
connection.setHeader("Require", "see-agree");
connection.send();
```

Due to the configuration of the IMS testbed at I2IT Pune lab, a single REGISTER request is adequate to perform registration. Authentication mechanisms have not yet been implemented within this testbed. Before a user register to the home IMS CN (S-CSCF), a user account must be created in the IMS HSS database (i.e. the allocation of public and private user identities). If the domain does not know the user, such a user will unable to register. After having sent the REGISTER request, the client has to wait for the final response. A 200-OK response means the user has registered successfully. A sequence diagram of the registration procedure is shown in Appendix G.1.

Figure 6.14 depicts the IMS Client display before the user can register to the home IMS CN. IMS Client is configured to first prompt the user to register. The SIP REGISTER request is the sent to the home IMS registrar, once the "Register" command has been pressed. If the user presses the "Exit" command, the IMS Client application will be closed and destroyed from the Sun WTK smart-phone emulator.

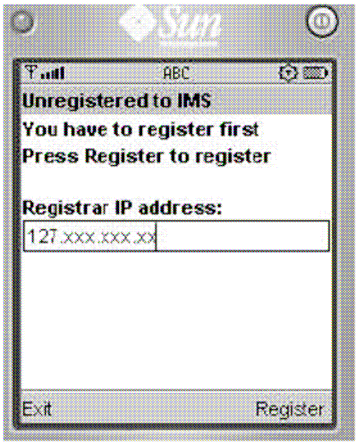

Figure 6.14: IMS Client Display before Registration

The Java code to create the above display is provided in Appendix H.

In order to avoid multiple registrations, the IMS Client displays a progress screen, as shown in Figure 6.15. The Client displays a "Please wait" message while it communicates with the IMS network entities. Though Figure 6.15 shows a busy display during registration, the same display is used when establishing the session.

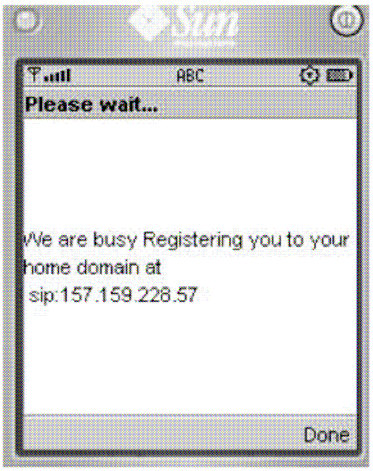

Figure 6.15: IMS Client Busy Display

If the IMS registrar or the whole IMS CN is not up and running, the client will timeout after 30 seconds, and allow the user to try again.

6.5.2 Session Establishment

After having successfully registered to the home IMS network (i.e. after receiving a 200-OK response), the IMS Client will display a different screen,

to allow the user to establish a session or to de-register from the IMS network. The sequence diagram for session establishment is shown in Appendix G.2. Figure 6.16 depicts the IMS Client display after a successful registration to IMS network.

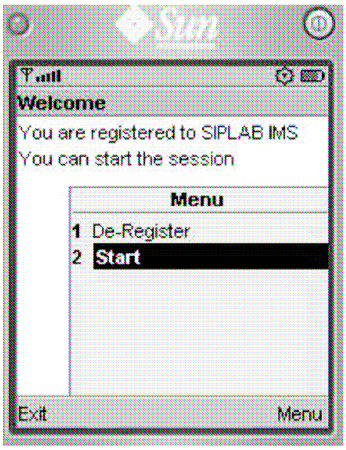

Figure 6.16: IMS Client Display after Registration

Again, if the user decides to close the IMS Client MIDlet, he presses the "Exit" command. The "Menu" command consists of two subcommands: the first one, "De-Register", allows the user to de-register from the IMS network, while the second one; "Start" command moves the display into the invite screen, shown in Figure 6.17.

If the user follows the "De-Register" command, a SIP REGISTER request will be sent to the IMS CN. The SIP REGISTER request that is sent for de-registration is similar to the one sent for registration, with the only exception being the value of the Expires header, which must be set to zero.

Figure 6.17: IMS Client Invite Display

From the invite display, the user can establish a session by entering the

destination SIP URI, and pressing an "Invite" command or return to the main display be pressing the "Back" command. When an "Invite" command is pressed, a SIP INVITE request is constructed and sent to the destination SIP URI, through the IMS network elements. IPv4 was used for the communication and registration in this study. A captured SIP trace for an INVITE request with SDP offer that was sent from IMS Client A (the user is User1) to IMS Client B (the user is User2) is shown in Table 6.2.

Table 6.2: INVITE Request Trace

```
INVITE sip:User2@157.159.103.57:5070 SIP/2.0
Call-ID: edb74ffbfflcad!8788cb6339b906ec7@ 157.159.229.253
CSeq: 1 INVITE
Via:SIP/2.0/UDP157.xxx.xxx.xxx;branch=z9hG4bK2ea5.2b263c91.0;i=8
Via: SIP/2.0/TCP 157.xxx.xxx.xxx;branch=0;i=e
Via: SIP/2.0/TCP 157.159.xxx.xxx;branch=0
Via:SIP/2.0/UDP
157.159.xxx.xxx:5070;branch=z9hG4bKdd69421d967dd7febbc60d55a43b05
87
Max-Forwards: 66
P-Preferred-Identity: User1<sip:User1@imstestbed.net>
P-Called-Party-ID: <sip:User2@imstestbed.net>
P-Visited-Network-ID: home
P-Access-Network-Info: siplab
From: User1 <sip:User1@imstestbed.net>;tag=1426653256
To: <sip:User2@imstestbed.net>
Require: lOOrel preconditions ,sec-agree
Supported: lOOrel
Contact: <sip:User1@ 157.159.229.253:5070>
Allow: INVITE,ACK,CANCEL,BYE,PRACK,UPDATE,REFER,MESSAGE
Content-Type: application/sdp
Content-Length: 230
v=0
o=User1 10022007 10022007 IN IP4 157.xxx.xxx.xxx
s=-
c=IN IP4 157.xxx.xxx.xxx
t=00
m=message 3150 SIP/UDP rfc3428
a=curr:qos local none
a=curr:qos remote none
```

```
a=des:qos mandatory local sendrecv
a=des:qos none remote sendrecv
```

6.5.3 Session Description Protocol (SDP) Offer/Answer

When establishing the session, i.e. sending an INVITE request, the SDP [136] first offer is also sent. This SDP [137] offer is attached to the INVITE request. The presence of the SDP payload on an INVITE request is indicated on the "Content-Type" header. The Content-Type header must contain the "SDP/application" value.

Because the IMS Client currently supports the Instant Messaging [138] session, the SDP offer contains the "text/plain" as the media type, as shown in this Java code:

```
HostSdp=
      "v=0\n"+
      "o=User1 23102007 23102007 IN IP4 " + mylP +"\n"+
      "s=-\n"+
      "c=IN IP4 " + mylP +"\n"+
      "t=0 0\n"+
      "m=message 9090 SIP/UDP rfc3428\n"+
      "a=curr:qos local none\n" +
      "a=curr:qos remote none\n" +
      "a=des:qos mandatory local sendrecv\n"+
      "a=des:qos none remote sendrecv\n";
```

As shown in Appendix G.2, SDP offer/answer is performed from the first INVITE request until the last 200-OK response of the UPDATE request.

6.5.4 Communication Mode

The IMS Client developed in this study focuses on session control and session management. To demonstrate the use of the client, communication is achieved through Instant Messaging (IM). The IETF has released an RFC 3428 [138], which is an extension of the core SIP (RFC 3261) [116] that allows the transfer of Instant Messages. In this RFC 3428 [118], a MESSAGE method is defined.

A Java snapshot for sending an EM is shown below:

```
1) public void sendMessage(){
2) SipClientConnection client_connection = null;
3) OutputStream output = null;
4) client_connection= dialog.getNewClientConnection("MESSAGE");
```

```
5) client_connection.setHeader("Content-Type","text/plain");
6) client_connection.setHeader("Content-Length",
Integer.toString(message.length()));
7) output = client_connection.openContentOutputStream();
8) output.write(message.getBytes());
9) output.close();
10)}
```

The SIP MESSAGE method is sent within an existing dialog as shown on line 4 of the Java code above. When sending a request within an existing dialog, "getNewClientConnetionQ" method is called with an intended method as a parameter.

6.5.5 Cancelling an Early Session

Cancelling a SIP session is performed using a SIP CANCEL method as specified in RFC 3261 [116]. An IMS Client user can cancel a request by pressing a "Stop" command during session establishment. A session will be cancelled only if the final 200-OK for an INVITE request has not been sent yet.

6.5.6 Ending Session

Either user pressing an “End” command can end a session. When a button is pressed, a UAC side of an IMS Client will process and send a BYE request. Based on RFC 3261 [116], an IMS Client may not send a BYE request outside of a dialog. The caller's IMS Client may send a BYE request for either confirmed or early dialogs and the caller's IMS Client may send a BYE request on confirmed dialogs, but may not send a BYE request on early dialogs.

Figure 6.18: An IMS Client Chatting Display

Figure 6.18 depicts the commands appearing on an IMS Client display during IM communication. A "Send" command is used to send the typed text, while an "End" command is used to send a BYE request. Once the UAC on an IMS send a BYE request, it will wait for a 200-OK final response to ensure that a session has ended.

6.6 System Implementation

In this section, an implementation of the IMS Client within the IMS testbed is discussed. The implementation was carried out using the development tools and libraries described in section 6.3, 6.4. Sub-section 6.6.1 discusses the IMS testbed and the implementation environment for study, while sub-section 6.6.2 provides the system simulation.

6.6.1 IMS Network Testbed

This study was carried out at the Wireless Network Laboratory I2IT, Pune (during September 2008 to March 2009). Figure 6.19 shows the whole IMS Testbed.

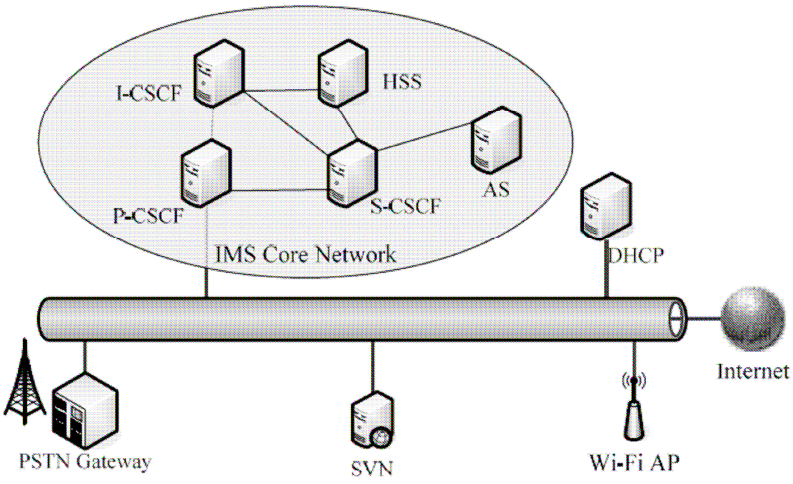

Figure 6.19: IMS Testbed

To perform the experiments on this testbed, the user credentials first have to be created within the HSS. The private user identity was created as: User1.imstestbed.net. The IMS Testbed can be accessed through either the LAN or Wi-Fi technologies within the Test bed network infrastructure.

6.6.2 System Simulation

Session setup, control and management are at the core of this study. The IMS Client uses SIP and SDP to manage and perform these actions. To ensure a functional system, Instant Messaging was used to communicate between

two users. The discussion of the implementation of the IMS Client is presented in this subsection.

i. Creating the IMS Client

A project was created using Sun WTK KToolbar (shown in Figure 6.20 (A)) and it was named "User1_IMS". If the Sun WTK has been installed on the C drive, on Windows machine, the project folder, User1_IMS, is created in the Sun WTK application directory as follows:

- **C:\WTK25\apps\User1_IMS**

When creating a project using the Sun WTK, the following folders are automatically created:

- src - stores the source code
- bin - stores the JAR and Java Application Descriptor (JAD) files
- classes - stores the class files of the compiled application
- res - stores the resource files, like pictures
- lib - stores the library files

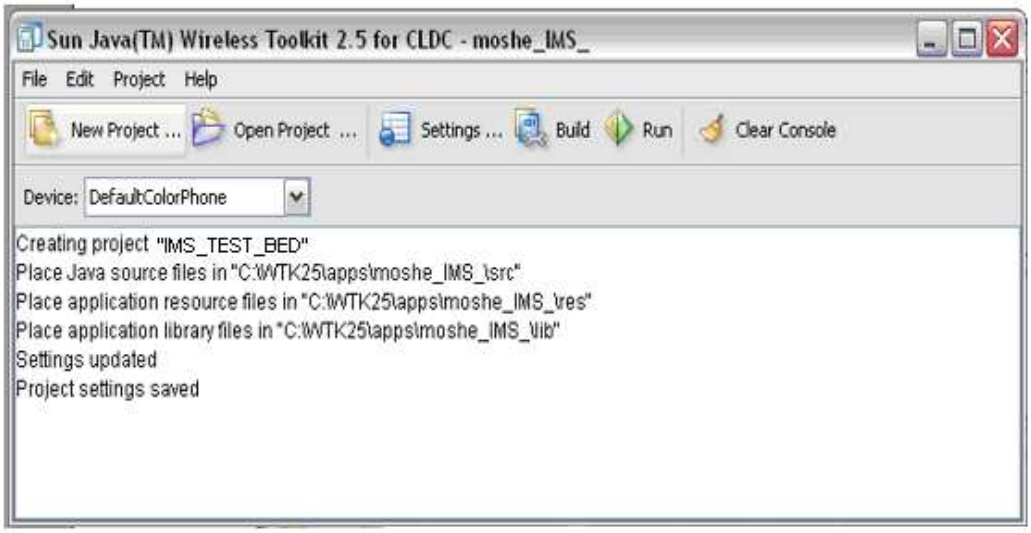

Figure 6.20 (A): Sun Wireless Toolkit - KToolbar

The Java source code was edited in EclipseMe IDE and stored in the src folder. The source code was compiled and pre-verified by building it with the Sun WTK. After the project has been built, the compiled files are stored in the Classes folder.

The next step is to package the project. Packaging is done using the Sun WTK Package submenu (this is under Project -> Package -> Create Package). By packaging the project, the MIDlet suite JAR and JAD files are created and stored in the bin folder. This MIDlet suite is ready to be deployed

into a mobile device emulator or a real device. The JAR and JAD files are the important files used to deploy the MIDlet into the emulated or real devices.

For this study, a Nokia phone emulator, provided with the Sun WTK was used to emulate the real mobile device. Sun WTK version 2.5 supports the MSA features, which include the SIP API for J2ME.

ii. Running the IMS Client

The "Run" command on the Sun WTK KToolbar is used to run the MIDlet. When the MIDlet is started, the registration screen is displayed. 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 chatting screen is displayed. The chatting screen allows the user to type text and send to the other remote user. Pressing the “End” command on the chatting screen can end communication.

6.7 EXPERIMENTAL SETUP

The IMS Client was tested in an experimental setup within the Wireless Network Lab at I2IT, Pune. The experimental setup is shown on Figure 6.20. The IMS Testbed comprises the IMS CN, which provides the basic control layer elements such as: P-CSCF, E-CSCF, S-CSCF and HSS. The Dynamic Host Configuration Protocol (DHCP) and the Application Server (AS) also form part of the IMS Testbed. Each IMS network element has its own EPv4 address. The VMware software is used to interact, from the same screen, with the P-CSCF, I-CSCF and S-CSCF, where each network element can be restarted or stopped. The HSS uses a separate machine, and with its own EP address and display.

The experiment was conducted using two identical IMS Clients, hosted from two different machines, but in the same Local Area Network (LAN), as shown in Figure 6.20 (B), from the IMS Testbed, two public and private user identities were created, one for User1 and the other one for User2.

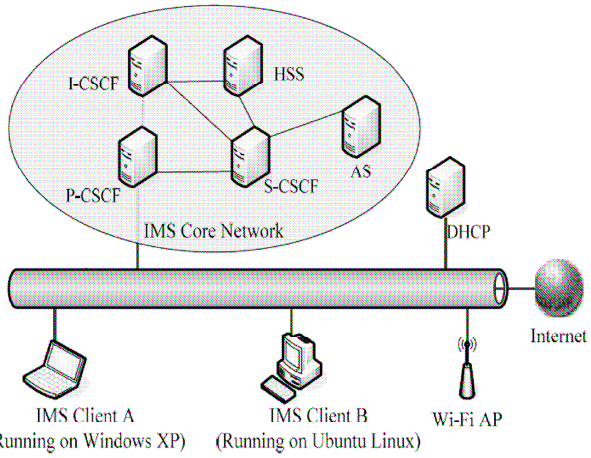

Figure 6.20 (B): Experimental Setup of IMS Testbed

6.7.1 Testing Requirements

The list of hardware and software requirements, and tools used for setting up the experiments are provided in the following subsections. The aim of this project was not to develop the IMS testbed; hence only the specifications for this testbed as outlined in subsection 6.7.1.3 were provided.

6.7.1.1 Hardware Requirements

Table 6.3 lists the hardware requirements for conducting the experiment using two IMS Clients and an IMS network (testbed).

Table 6.3: Hardware Specifications

Name	Description
Personal Computer (PC)	
CPU	Intel Pentium 4, 2.66 GHz
RAM	512MB
Hard Drive	80GB
Network Connection	Intel PRO/100 VE
Laptop	
CPU	Intel Centrino Duo 1.66 GHz
RAM	1GB RAM
Hard Drive	80GB
Network Connection	Intel PRO/100 VE and Intel PRO/Wireless 3945ABG

6.7.1.2 Software Requirements

Table 6.4 lists the software requirements for conducting the experiment using two IMS Clients and an Open IMS network (testbed).

Table 6.4: Software Specifications

Name	**Description**
Personal Computer (PC)	
Operating System	Linux Ubuntu 6
Java	Jdk 1.5.1, Java Virtual Machine(JVM)
J2ME platform	Linux version of Sun Wireless Toolkit 2.5.2
EclipseME	Source code editor. A plug-in to Eclipse
Phone Emulator	Provided within Sun WTK
Laptop	
Operating System	Microsoft Windows XP Home Edition
Java	Jdkl.5.0_09JVM
J2ME platform	Windows version of Sun WTK 2.5.1
EclipseME	A source code editor. A plug-in to Eclipse 3.2
Phone Emulator	Provided within Sun WTK
Other Software	
Wire shark	A network protocol analyzer used to capture network packets (formally known as Ethereal).

6.7.1.3 Testbed Specifications

The specifications provided in this subsection were provided based on Open IMS Core testbed.

Table 6.5: IMS Testbed Specifications

Name	**Description**
Hosts	
CPU	Pentium 4 @3.40GHz
RAM	2.0GB
Server	Windows Server 2003 Enterprise Edition
	VMWare Server 1.0.2
Guests	
CPU	Shared CPU
RAM	256MB
GNU/Linux Debian	Core 2.6.8 arch i686
SIP	SIP Express Router 0.8.14
Database	MySQL server 5.0.20
Network	
Server	Windows DHCP server for clients
Router	Cisco 400 Series (10 Mbps interfaces)
Switch	MRV MR2228-S2C (100 Mbps Interfaces)
Infrastructure Type	Bus Type

6.7.2 Experimental Demonstration

The IMS Client was tested on the setup shown in Figure 6.3. Both Clients first had to register to the IMS network. After registration, IMS Client A established a session by sending an INVITE request to IMS Client B. Figure 6.4 shows the testing environment at IMS testbed.

The P-CSCF, E-CSCF and the S-CSCF are hosted within the same machine, while the HSS is on a separate machine. The IMS Client application or MEDlet was deployed within two separate computers. IMS Client A was deployed within the laptop, while IMS Client B was deployed within the personal computer.

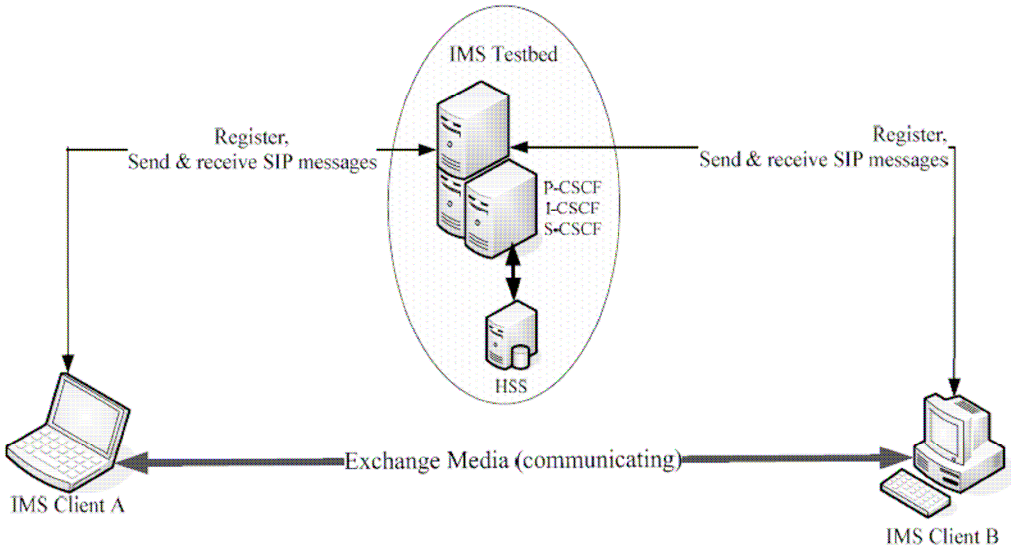

Figure 6.21: IMS Testing Environment

During registration and session establishment, the signalling path was used. All the signalling messages traversed the IMS network, while the payload media used a different channel. Media was exchanged directly between the two IMS Clients, without passing through the IMS network.

Figure 6.22 shows one of the IMS Client screenshots from the start until the end of the session. At the beginning, the user is given a screen with an option to register to the IMS network. After having registered successfully, the user can decide to establish the session or to de-register from the home IMS network. To get the invite screen, the users use the "Start" command. Within the invite screen, a user is requested to enter the destination SIP address, and then press an "Invite" command to send a SIP INVITE request.

An INVITE request will follow the same path used during registration (i.e. go through the home IMS network towards the destination user). The home IMS network will forward an INVITE request to the destination user provided such a user is registered. If the user is not registered, the IMS network will return the final response to the originating user, indicating that the other user is not available.

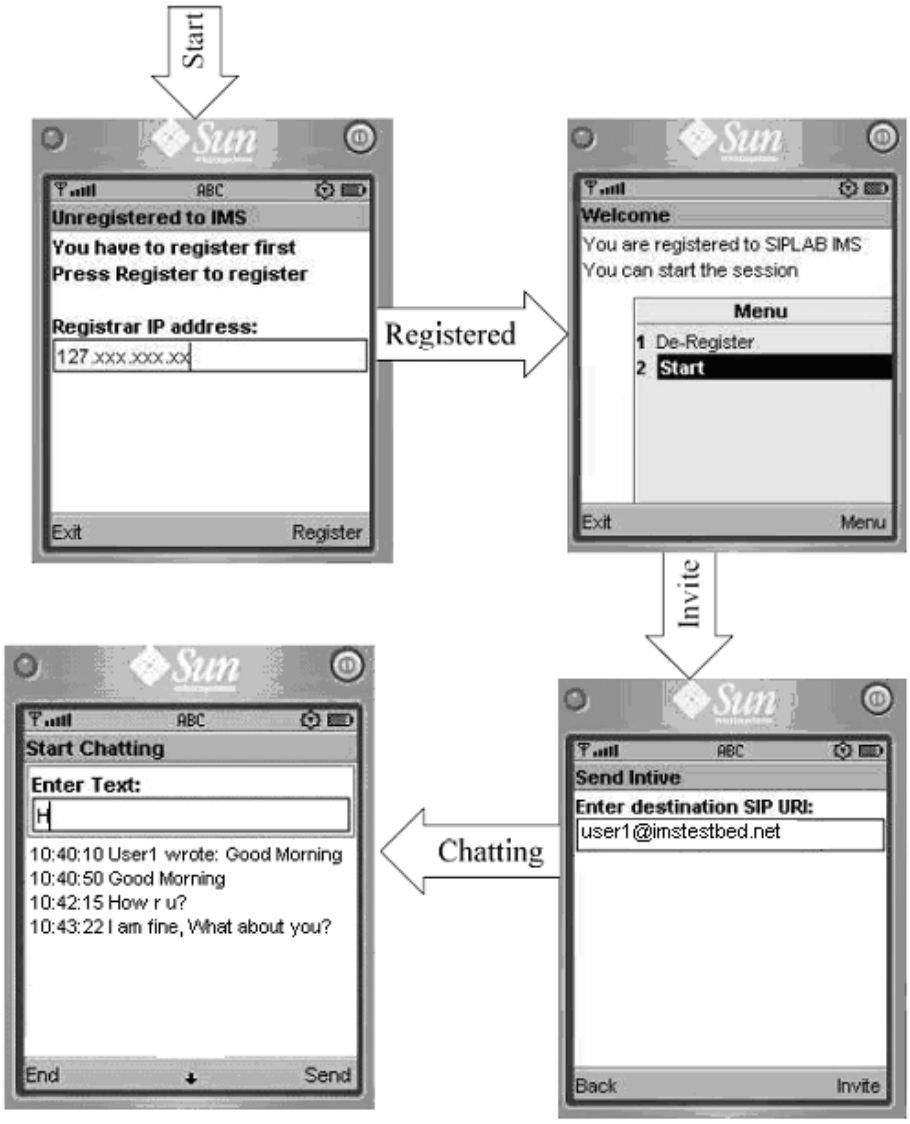

Figure 6.22: Demonstration of How to Establish a Session

6.8 Results Discussion

In this section the evaluation and discussion of the results based on the experimental demonstration carried out on the previous section will be done. En the next section, the results as extracted from the Wireshark Internet protocol analyzer [94] will be discussed. The aim is to show and discuss the SIP messages (requests and responses) exchanged between the two IMS Clients and the IMS testbed. The SDP content will also form the basis of the results discussion.

Poikselka at el [139] indicated that one of the most important issues of the IMS design was to separate the control plane from the user plane. But there was a need to find the mechanism to authorize and control the use of the bearer traffic intended for IMS media traffic. This mechanism that was

found is based on SDP parameters negotiated at the IMS session.

6.8.1 Registration Request

Figure 6.23 shows the SIP REGESTER request sent from User2 IMS Client to the home IMS network (which is the IMS testbed). The registering IMS Client's SIP URI is placed under the "To:" and "From:" headers. The "To:" header does not have any tag. Because this is the original request, only the "From:" header can have a tag number.

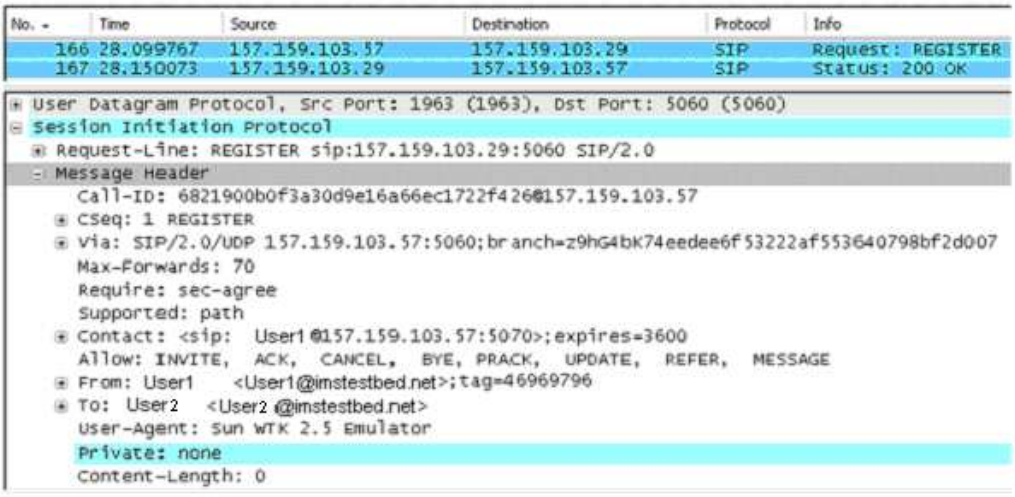

Figure 6.23: REGISTER Request Trace

The Contact header that is very important & mandatory in the SIP REGISTER request. It must include the host device's IP address, port number and the expiry period. The IMS Client will be registered to the home IMS once it has received a 200-OK response.

6.8.2 Invite Request

Figure 6.24 depicts the SIP INVITE request, originated by User2 IMS Client and sent to User1 IMS Client. The INVITE request contains the SIP headers and the SDP message. The QoS in IMS is guaranteed during a SIP session setup or session modification procedure when an IMS Client negotiates its capabilities and expresses its QoS requirements [139]. The attribute lines or a-lines under the m-line of the SDP form a block that describes in more detail the media of the m-line under which they are placed. To understand these SDP attribute tags, a brief explanation is given in Table 6.6.

Table 6.6: SDP Attributes

Attribute Tags	Description
Curr (Current)	Indicates the extent to which the set preconditions have already been fulfilled.
Des (Desired)	Makes it possible to set desired preconditions for the local and the remote user.
Qos (Quality of Service)	The precondition is set due to certain quality requirements of the related media stream.
Local	The local user, who originates the session.
Remote	Indicates the destination.
None	Indicates that no parameter was set
Mandatory	Indicates a mandatory precondition.
Sendrecv (Send and Receive)	Tools should start in send and receive mode

The IETF has released a RFC 4566 document that supports the following media types. These are: "audio", "video", "text", "application", and "message". This list may be extended in the future. In this study, the IMS Client will support only the "message" as the media type, as shown in all the SDP messages.

Note: in Figure 6.24 draws the attention to the desired QoS attribute from the local or calling subscriber. The local user indicates the desired (des) precondition to start a session in send and receive mode (sendrecv), as indicated by the "mandatory" tag.

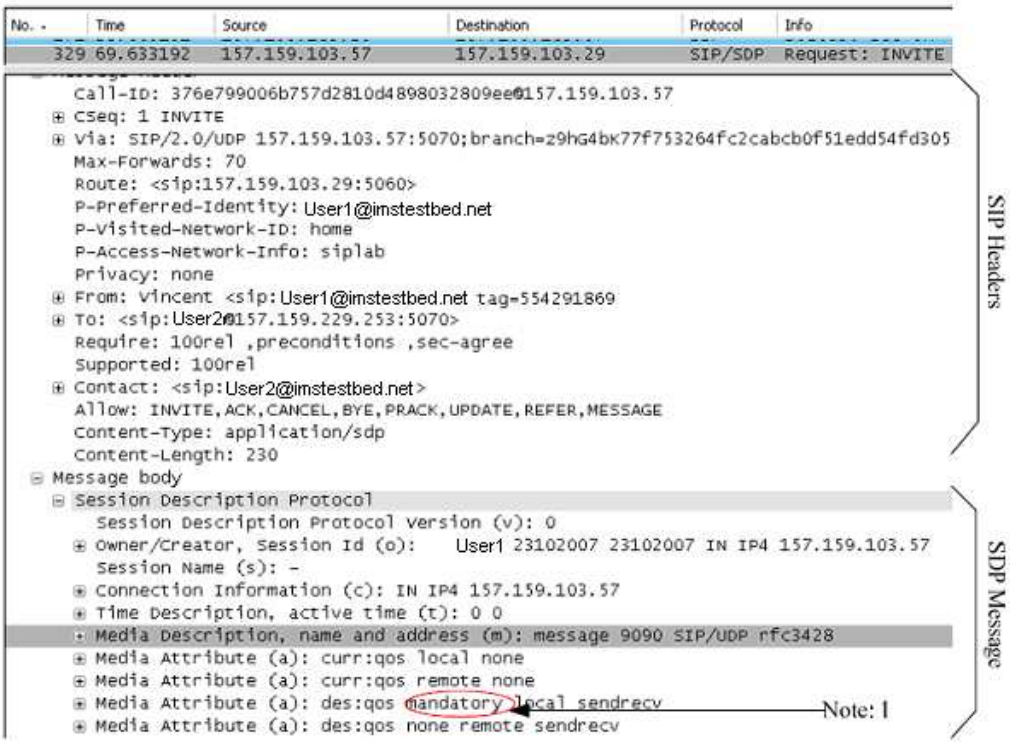

Figure 6.24: INVITE Request Trace

6.8.3 Trying Response

To avoid frequent re-transmission of the INVITE request from the UE, the P-CSCF sends back a 100-Trying response after it has received the INVITE, to indicate that from here on the P-CSCF will take care of the re-transmissions.

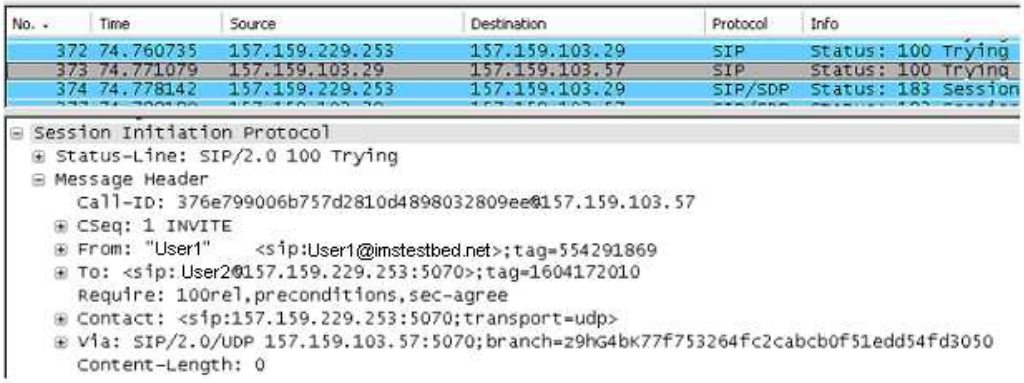

Figure 6.25: 100-Trying Response Trace

The 100-Trying response is sent by all call-state full SIP proxies on the route, and it is always stopped at the SIP proxy that is the last to take over responsibility for re-transmission, as shown in Figure 6.25.

6.8.4 Session Progress Response

The 183-Session-in-Progress response is sent by the remote client to indicate that session establishment procedures have been started, but the called user has not been informed yet. It is sent immediately on receipt of the initial INVITE request. It also contains the SDP first answer, as shown in Figure 6.26. The remote user will indicate the desire to start a session in the send and receive mode as indicated by Note 2 in Figure 6.26.

Going back to Figure 6.24, the remote sendrecv tag was set to "none". And now it is set to "mandatory". The media type is also set to "message", but with different port number. This means that both IMS Clients support the same, single media type, which is "message".

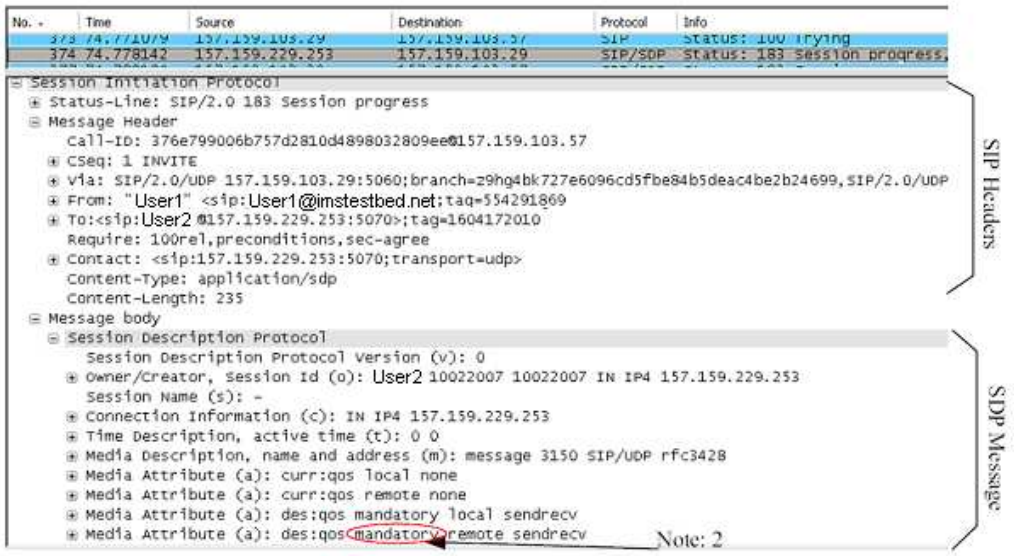

Figure 6.26: 183-Session Progress Response Trace

6.8.5 PRACK Request

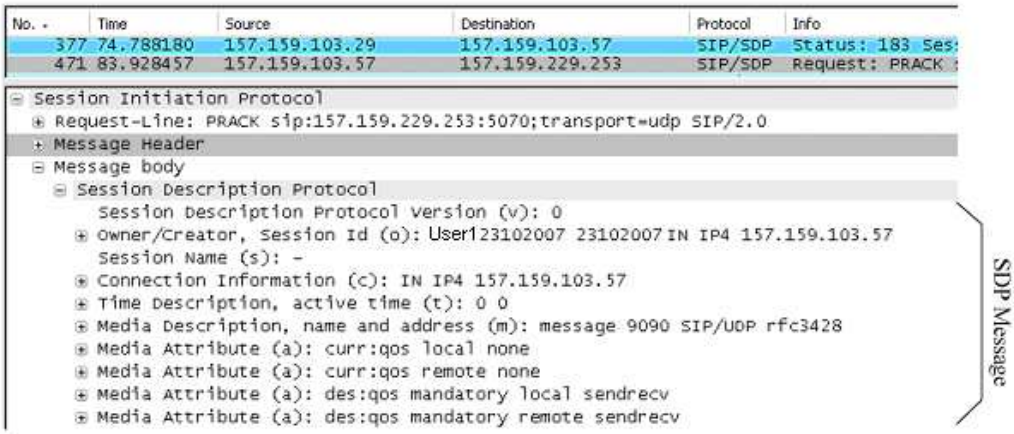

Figure 6.27: PRACK Request Trace

The local client sends back *a* PRACK request to acknowledge the received 183-Session in Progress response. This PRACK also carries the second SDP offer. The final decision is on media is done in this request. Because IMS Clients involved in this session support the same media type, a decision is made and the PRACK is sent to confirm the final decision with the remote client, as shown in Figure 6.27.

The remote client responds to the PRACK request with a 200-OK response as shown in Figure 6.28.

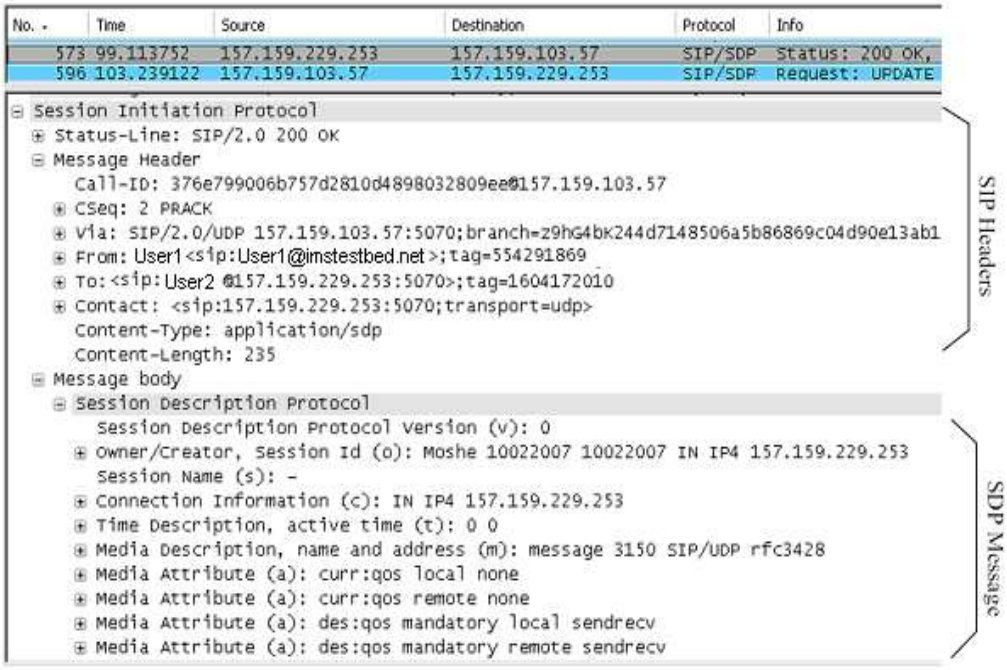

Figure 6.28: 200-OK Response for PRACK Trace

6.8.6 UPDATE Request

The local client sends the UPDATE request on receipt of the 200-OK (PRACK) response. Now the codec and resources have been reserved for the session; and the local user indicates its current QoS to the remote client as indicated by Note 3 in Figure 6.29.

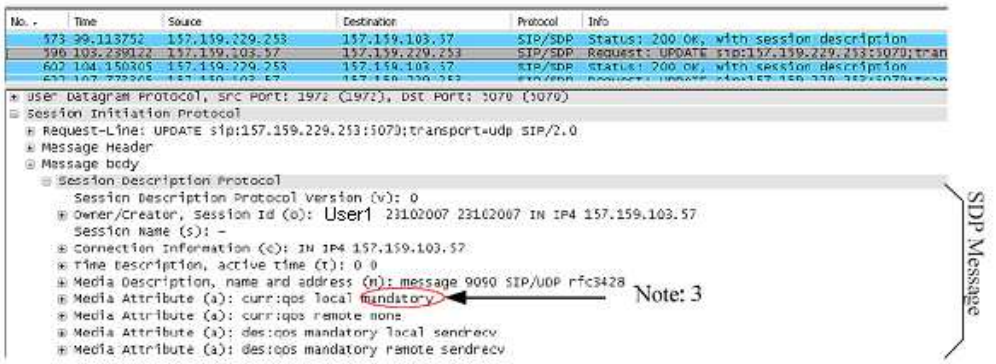

Figure 6.29: UPDATE Request Trace

The remote client, on receipt of the UPDATE request, will respond with a 200-OK response containing the SDP final answer. The remote client will also indicate its current (curr) qos preconditions *by* changing the parameter from "none" to "mandatory" as indicated by Note 4 in Figure 6.30.

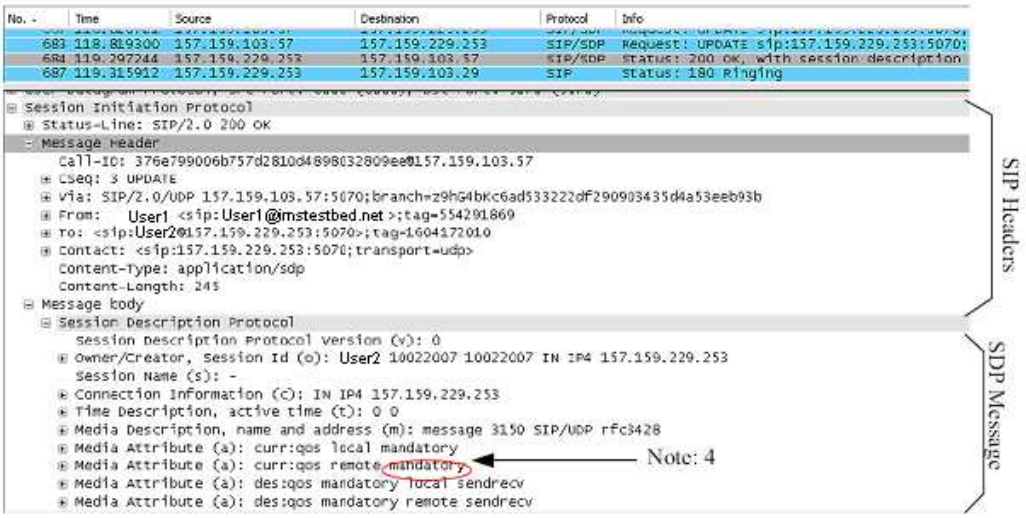

Figure 6.30: 200-OK Response for UPDATE Trace

6.8.7 ACK Request

The local client generates and sends an ACK request for a 200-OK INVITE response to the remote client. The header fields of the ACK are constructed in the same way as that for any request sent within a dialog as shown in Figure 6.31. The sequence number of the CSeq header field must be the same as the INVITE being acknowledged, but the CSeq method must be ACK. The ACK contains the same credentials as the INVITE.

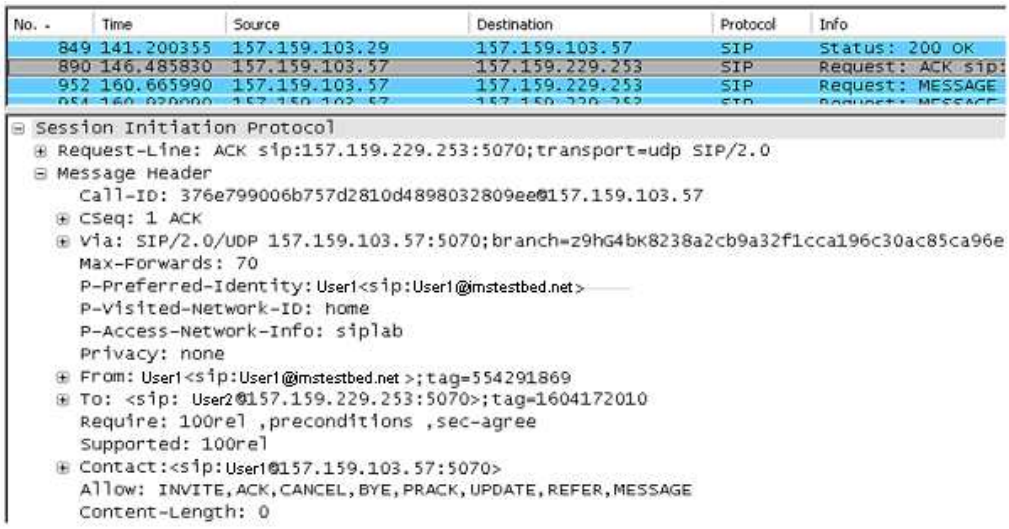

Figure 6.31: ACK Request Trace

6.8.8 Communication Mode

Communication between the two Light-Weight IMS Clients was achieved making use of the SIP MESSAGE method. This method carries plain text as the content type, as shown in Figure 6.32. For every message sent from one client to the other, a 200-OK response should also be sent. Media was exchanged between two clients, without being routed through the IMS network.

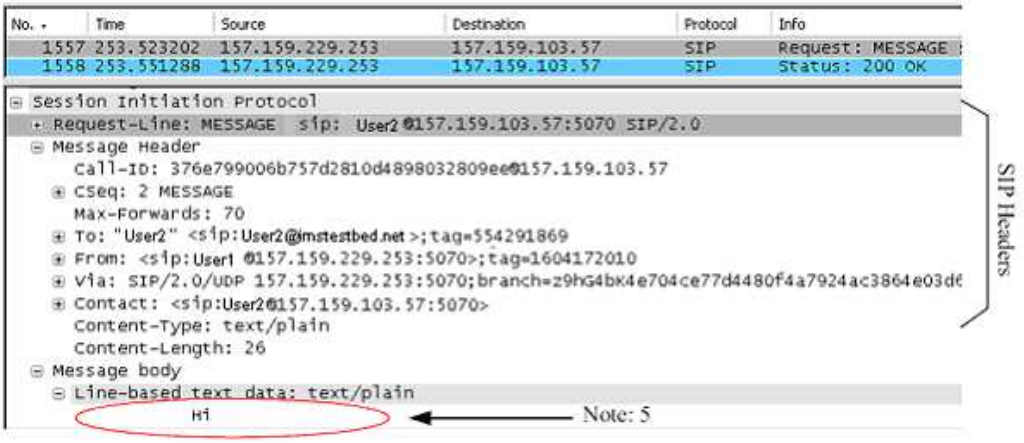

Figure 6.32: MESSAGE Request Trace

The above results, shown in Figure 6.23 to Figure 6.32, indicate that the developed IMS Client is capable of performing the IMS functionalities, which makes it compliant with 3GPP and IETF IMS specifications.

Though this Light-Weight IMS Client was configured to prompt the home

network S-CSCF IP address before registering, registering can be performed automatically without prompting the user. This can be achieved by pre-configuring the P-CSCF or S-CSCF IP address within the IMS Client.

A session was established using an INVITE request. Establishing a session also initiated the SDP offer/answer mechanism to allow the two IMS Clients to negotiate and agree on the media type and resource reservation for QoS. It can be seen that the media attributes at the beginning of session establishment shown in Figure 6.23 were updated regularly until the session had been established successfully.

6.9 Low size & Open IMS Client Implementations

The implementation functionalities of the Light-Weight IMS Client and other available open IMS Clients are provided in Table 6.7.

Table 6.7: Comparison of Low-sized and Open IMS Client Functionalities

Functionality	Low-sized IMS Client	FOKUS Open IMS Client Lite	IMS-Communicator
Development Platform	J2ME (CLDC)	Java (JSE) and .NET	Java (JSE)
Targeted End-User Device	CLDC Compliant Mobile Phones	Pocket PC, Laptops and Desktops	Desktops and Laptops
Memory Footprint	About 37 KB (Excluding Libraries)	About 1.3 MB (Excluding Libraries)	About 1.5 MB (Excluding Libraries)
Authentication Support	In Future	Digest - AKA	Digest - AKA (Work in progress)
Client Availability	Not Available online	Available online (No source code)	Available online (With source code)
Standard Compliance	3GPP and IETF	3GPP, IETF and TISPAN	3GPP, IETF and TISPAN
SIP Support	Yes	Yes	Yes
RTP Support	In the Future	Yes	In the Future

6.10 Summary of Research

In this chapter, the experimental setup for the testing of two identical Light-Weight IMS Clients within the IMS testbed was presented. The equipment and requirements for conducting the demonstration were also provided. The

discussion of results was based on the SIP signalling messages exchanged between the IMS Clients 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 shows that the IMS Client is compliant with the 3GPP technical specifications and IETF SIP recommendations. The IMS Client memory size is about 37 Kilo Bytes (KB) as compared to 6 305 KB of Fokus Open IMS Client [83] and 86 953 KB of UCT IMS Client version 1.0.4 [127]. As opposed to these available IMS Clients, the Light-Weight IMS Client presented in this project is light-weight, and can be deployed within the CLDC mobile devices which are J2ME compliant.