MASTER IN COMPUTER SCIENCE

Implementation of a presence and instant messaging system using SIP with mobility support

Le Kim, David

Award date:
2005

Facultés Universitaires Notre-Dame de la Paix, Namur
Institut d'Informatique
Année Académique 2004 - 2005

Implementation of a presence and instant messaging system using SIP with mobility support

David Le Kim

Mémoire présenté en vue de l'obtention du grade de Maître en Informatique

Abstract

This master thesis proposes an implementation of a SIP client offering presence and instant messaging functions, under the assumption that SIP proxies are not presence-enabled.

This implementation relies on SIMPLE, a SIP-based protocol, whose purpose is to resolve interoperability issues among various SIP clients. Our client has also been enhanced with mobility support without continuous connectivity at application-layer level. Throughout this paper, we provide an introduction to SIP, presence, instant messaging and mobility support.

Résumé

Ce mémoire de maîtrise propose une implémentation d'un client SIP offrant des fonctionnalités de présence et de messagerie instantanée avec l'hypothèse selon laquelle les proxy SIP ne possèdent pas de fonctions de présence.

Cette implémentation repose sur SIMPLE, un protocol basé sur SIP, dont le but est de résoudre les problèmes d'interopérabilité parmi les clients SIP. Notre client a également été conçu de manière à supporter la mobilité sans connexion permanente au niveau de la couche applicative. Dans ce mémoire, nous proposons une introduction à SIP, aux fonctionnalités de présence et de messagerie instantanée ainsi qu'au support de la mobilité.

Acknowledgements

First and foremost, I wish to thank my supervisor, Professor Laurent Schumacher, for his guidance and valuable advices throughout the various revisions of this master thesis.

I am also grateful to Professors Jean-Marie Bonnin and Xavier Lagrange for giving me the opportunity to work within the *Réseaux et Services Multimédias* department at *Ecole Nationale Supérieure des Télécommunications de Bretagne* in Rennes, France.

I wish to express my gratitude to Bruno Deniaud for all kinds of support he provided me during my internship.

I acknowledge the help of Emil Ivov for his technical support on Sip Communicator.

I also wish to express my appreciation to all those people who brought a significant contribution to this thesis and the work carried out during my internship.

Finally, I would like to thank my parents for their love, encouragement and confidence in me throughout these past five years.

Contents

List of Figures

List of Tables

List of Abbreviations

3GPP	Third Generation Partnership Project
APEX	Application Exchange
CGI	Common Gateway Interface
CH	Correspondent Host
CN	Correspondent Node
CPIM	Common Profile for Instant Messaging
CPP	Common Profile for Presence
FA	Foreign Agent
GPRS	General Packet Radio Service
GPL	General Public License
GRE	Generic Routing Encapsulation
GUI	Graphical User Interface
HA	Home Agent
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IETF	Internet Engineering Task Force
IM	Instant Messaging
IMPP	Instant Messaging and Presence Protocol
IP	Internet Protocol
IPSec	IP Security
ITU	International Telecommunications Union
JAIN	Java API for Integrated Networks
JMF	Java Media Framework
LAN	Local Area Network
MIME	Multipurpose Internet Mail Extensions
MH	Mobile Host
MN	Mobile Node
MSRP	Message Session Relay Protocol
OS	Operating System
PA	Presence Agent

PIDF	Presence Information Data Format
PRIM	PResence and Instant Messaging
PSTN	Public Switched Telephone Network
PUA	Presence User Agent
QoS	Quality of Service
RFC	Request For Comments
RPID	Rich Presence Information Data Format
RSVP	Resource ReserVation Protocol
RTCP	RTP Control Protocol
RTP	Real-time Transport Protocol
RTSP	Real-Time Streaming Protocol
SAP	Session Announcement Protocol
SCTP	Stream Control Transmission Protocol
SDP	Session Description Protocol
SIMPLE	SIP for Instant Messaging and Leveraging Extensions
SIP	Session Initiation Protocol
SMS	Short Message Service
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier
VoIP	Voice over IP
XML	eXtensible Markup Language
XMPP	eXtensible Messaging and Presence Protocol

Chapter 1

Introduction

Presence and instant messaging, the functions that enable a user to check people's status online and send them real-time messages, has proved to be one of the most popular applications on the Internet. Since the revolution of Internet, presence and instant messaging systems have been gathering hundreds of millions of users who stay connected for very long periods of time. With the introduction of wireless technologies like Wi-Fi and UMTS, we expect a similar phenomenon in such networks. Paradoxically, presence and instant messaging services seem to be one of the most difficult Internet applications to standardize as each of them uses its own protocol.

The purpose of this work is to implement a SIP (Session Initiation Protocol) client which combines presence awareness and real-time instant messaging based on an IETF protocol called SIMPLE (SIP for Instant Messaging and Leveraging Extensions), under the assumption that SIP proxies are not presence-enabled. As its name suggests, SIMPLE is a SIP-based protocol and has been designed to offer interoperability among SIP-based applications. As of today, SIP, to be discussed in Chapter 2, is currently the leading signaling protocol for Voice over IP gradually replacing H.323 in this role.

Our SIP client is meant to be used on mobile terminals. Enabling terminal mobility in IP-based networks can be achieved through a variety of means. Mobile IP, to be discussed in Chapter 5, is one of them. However, due to its

limitations and high complexity, SIP-based solutions coming from university research have been proposed to provide mobility support at application-layer level. In order to support mobility management, our SIP client has been designed from one of those solutions.

This work was carried out within the framework of the project SIPMOB during a four-month internship at *Ecole Nationale Supérieure des Télécommunications de Bretagne* under the supervision of Professor J.M. Bonnin.

This thesis is structured as follows. Chapter 2 will introduce the SIP protocol. This chapter will describe how SIP works, its entities, the structure of its messages, etc. Chapters 3 and 4 will respectively describe the concept of presence and instant messaging and present the existing solutions on Internet. Following these two chapters, Chapter 5 will introduce the mobility support with Mobile IP and SIP. Chapter 6 will present the application analysis and requirements that our application should fulfill. Chapter 7 will describe the design and the implementation of our application. Chapter 8 will introduce the results, discuss the proposed solution and present the future work that can be performed. Finally, Chapter 9 will present the conclusions that can be drawn from this thesis work. Readers familiar with SIP, presence, instant messaging and Mobile IP can safely skip reading chapters 2, 3, 4 and 5.

Chapter 2

SIP - Session Initiation Protocol

Session Initiation Protocol (SIP) is an application-layer signaling protocol for establishing, modifying and terminating multimedia sessions [25]. In other words, it provides a way to establish audio, video and messaging communications between devices over a network, i.e. the Internet. SIP's conception of networks matches that used in the Internet : smart endpoints or devices exchange data with each other over a simple transport infrastructure. This contrasts with the traditional telephone network which uses dumb endpoints over an intelligent network. This difference makes the core network work more efficiently as intelligence is placed where it is needed the most.

2.1 The Origins of SIP

SIP was originally developed around 1996 from an academic project lead by Henning Schulzrinne, Associate Professor of the Department of Computer Science at Columbia University. His intent within the Multi-Party Multimedia Working Group was to define a mechanism which allows voice, video and data to be integrated over the same network. In 1999, the Internet Engineering Task Force (IETF) issued the first SIP specification, RFC 2543. During the following years, new SIP's functions were rapidly developed and extended for use in instant messaging and presence for example, so a new

set of standards based on RFC 3261 was released in 2002. Nowadays, many companies are offering an increasing number of SIP-based applications and services.

2.2 Overview of SIP Functionalities

Here are the five facets SIP is supporting for establishing and terminating multimedia sessions [25] :

- **User location :** determines the end system to be used for communication.
- **User availability :** determines if the called party wants to engage in communications.
- **User capabilities :** determines the media and media parameters to be used during the session.
- **Session setup :** establishes the session parameters at both called and calling party.
- **Session management :** includes transfer and termination of sessions, modifies sessions parameters and invokes services.

Functionalities of SIP include the following capabilities as described in [10] :

- **Mobility** : no matter where it is, a SIP client can dynamically register to its home location and access services it asks for. Thanks to its unique identifier, which is similar to an email address, all its calls would be forwarded to it. Multiple devices can be associated with this identifier, calls will be routed to them simultaneously or sequentially, according to the user policy.

- **Separation of signaling and media** : with SIP, signaling paths are totally independent from media's ones. Signaling and media may be routed through different locations on different physical networks. SIP does not define the type of session that is being established but rather how it should be managed.
- **Flexible message structure** : SIP messages are text-encoded, present a simple structure thus are easy to read, understand and debug. Developers can easily and quickly create applications using popular programming languages such as Java.
- **Media negotiation** : SIP allows clients to negotiate media parameters and protocols to be used during a session. For example, a SIP client would be able to choose a more suitable media while moving to a network with a smaller bandwidth.
- **Transport layer independent** : SIP uses User Datagram Protocol (UDP) as well as Transmission Control Protocol (TCP) to connect users between them no matter what the underlying infrastructure is. SIP can work both with IPv4 and IPv6.
- **Multi-devices support** : If a SIP device establish an audio/video session, audio can still be transmitted to other SIP endpoints even if they are non-video enabled. Also different media parameters can be used for transmitting and receiving video/audio streams.

2.3 SIP Entities

SIP is based on the client/server transaction model. A SIP client sends out a request. A SIP server responds to that request by generating a response. During a session, a SIP entity can participate in SIP sessions as a client, as a server, or as both.

We can find two types of logical SIP entities :

2.3.1 User Agent

In a SIP network, a *User Agent (UA)* is an endpoint device which initiates and terminates media sessions by exchanging requests and responses. A SIP UA can be a SIP phone as well as a SIP software or even a telephony gateway. As defined in [25], a UA is a logical entity that can act both as a *User Agent Client (UAC)* and *User Agent Server (UAS)*.

- **User Agent Client** : logical entity that creates new request
- **User Agent Server** : logical entity that generates responses to received SIP requests

This is different from the classic client/server model as a SIP entity can play both the role of server and client during the course of a same session.

2.3.2 Server

SIP server is part of the SIP-enabled network. It helps UA to set up sessions and assists them in other functions. There are four types of SIP servers : proxy, redirect server, registrar and UAS which has been previously introduced.

- **Proxy** : logical intermediary entity that plays the role of forwarding SIP messages to another SIP entity as close as possible to the targeted user. A proxy acts both as a client and server, it can rewrite some parts, i.e. headers, of a SIP message before routing it to its destination. A proxy also ensures that a user is allowed to make a call and has the appropriate authentication.
- **Redirect Server** : logical intermediary entity that receives requests from UAC or another proxy and returns a response, redirecting the UAC to another location.
- **Registrar** : logical entity that receives SIP registration requests and updates databases containing the location of all UAs within a domain.

As those servers are logical, a same physical box can play both the role, for example, of proxy and registrar at the same time.

2.4 Encapsulation and Layer Structure

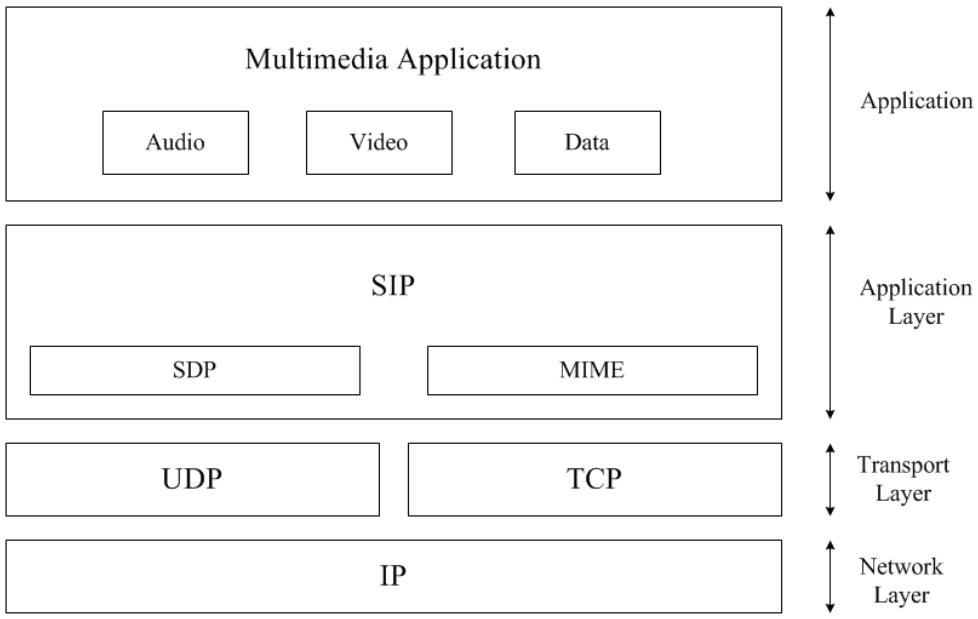

Figure 2.1: Encapsulation and Layer Structure

Figure 2.1 represents SIP messages encapsulation. SIP messages may carry a *Session Description Protocol (SDP)*, which will be fully detailed in Section 2.6.1, or MIME content i.e. text. SIP's body may also contain other content types defined by the IETF. As mentioned in Section 2.2, SIP messages are totally transport layer independent and have their own built-in reliability mechanisms. The application itself determines the appropriate protocol for the communication. As UDP session management is more straightforward compared to any other transport protocols, most of SIP clients and SIP phones use UDP. Therefore TCP is rather used when a longer and more permanent connection is required.

As a signaling protocol, SIP does not manage multimedia sessions neither exercise control on media transmitted. However, SIP works in conjunction with RTP/RTCP (Real-time Transport Protocol/RTP Control Protocol)

for transporting real-time data, RSVP (Resource ReSerVation Protocol) for reserving resources, RTSP (Real-Time Streaming Protocol) for controlling streams delivery and SAP (Session Announcement Protocol) for announcing multimedia sessions [36]. All these protocols are application layer protocols [26].

2.4.1 RTP/RTCP

The Real-time Transport Protocol (RTP) was designed to support end-to-end transport of real-time media. Common formats like PCM, GSM and MP3 for sound and MPEG and H.263 for video can be conveyed using RTP. RTP is also often used in conjunction with Internet telephony standards such as SIP and H.323. RTP's header has four main fields : sequence number, timestamp, synchronization source identifier (SSRC) and payload type.

The sequence number allows receiver to detect packet loss and reorder packets on arrival. Each packet is identified by a unique sequence number, so if there is a gap in the RTP stream between sequence numbers 15 and 18, the receiver understands that packet number 16 and 17 are missing.

Due to random queuing delays in the routers, the time between the generation of a packet at its source and the reception of this packet at the sink may strongly vary from packet to packet. This phenomenon is known as jitter. The timestamp field aims at removing packet jitter by using buffers at the destination location and allows the stream to be synchronously played out.

The synchronization source identifier (SSRC) purpose is to identify the source of the packet stream. Each stream in a RTP session is associated with only one SSRC.

Each RTP stream may be encoded following different audio and video formats. The payload type specifies the type of audio or video encoding used

during the RTP session. Thanks to this field, the sender may change the encoding during a same session to increase the audio/video quality or decrease the stream bit rate according to the bandwidth availability. We should emphasize that RTP does not guarantee any Quality of Service (QoS) at all, that is the reason why other protocols such as RTCP and RSVP should work in conjunction with [26]. Table 2.1 and 2.2 shows some audio and video payload types supported by RTP.

Payload Type Number	Audio Format
0	PCM μ-law
3	GSM
9	G.722
14	MPEG Audio

Table 2.1: Some Audio Payload Types Supported by RTP

Payload Type Number	Video Format
26	JPEG
31	H.261
32	MPEG1 Video
33	MPEG2 Video

Table 2.2: Some Video Payload Types Supported by RTP

The RTP Control Protocol (RTCP) provides feedback information, for example, to modify the transmission rate, to all the participants of a RTP session. The feedback information is about the quality of the data transmission or long-term statistical data. RTCP can also perform fault diagnosis to determine whether problems are local, regional, or global [26].

2.4.2 RSVP

The purpose of the Resource ReSerVation Protocol (RSVP) is to reserve network resource[1] for transmitting real-time data flows. RSVP is considered

[1]We assume here that the word *resource* is synonymous to bandwidth.

rather as a signaling protocol than a routing protocol due to the fact that it does not specify the paths where reservations should be made. Applications will send RSVP requests to routers which provide the reserved resource to data streams [26].

2.4.3 RTSP

The Real-Time Streaming Protocol (RTSP) allows users to replay various media across the Internet. Users will be able to control playback as they wish i.e. rewinding, forwarding, pausing, ... RTSP provides a framework to control audio/video streams, it does not deliver the media streams itself. RTSP is likely to be used with RTP but they can also work independently [26].

2.4.4 SAP

Session Announcement Protocol (SAP) is a multicast session announcement protocol which consists in advertising multicast conferences and communicating specific multicast address and time information to prospective participants. SAP carries a payload that describes the session i.e. Session Description Protocol (SDP) explained in Section 2.6.1 [8].

2.5 SIP Messages

2.5.1 Requests and Response Codes

A SIP request is a SIP message sent from a client to a server. A SIP response is a SIP message sent from a server to a client. SIP is based on a request/response transaction model analogous to HTTP[2]. A transaction is a two-part process : a request invoking a method on the server and at least one response.

[2]Although SIP's message and header field are similar to HTTP, SIP is not an extension of HTTP.

METHOD	DESCRIPTION
INVITE	Session setup
ACK	Acknowledgment of final response to INVITE
BYE	Session termination
CANCEL	Pending session cancellation
REGISTER	Registration of users
SUBSCRIBE	Request notification of an event
UNSUBSCRIBE	Cancel notification of an event
NOTIFY	Transport of subscribed event notification
MESSAGE	Transport of an instant message body
UPDATE	Update of the session parameters
REFER	Transfer user to a URL
INFO	Midcall signaling transport

Table 2.3: Main SIP methods

Table 2.3 shows the main SIP methods. The five first methods are the basic ones described in RFC 3261 [25], the others are SIP extensions : the INFO method (RFC 2976 [12]), Event Notification Framework (RFC 3265 [41]), the UPDATE method (RFC 3311 [42]), the MESSAGE method (RFC 3428 [5]) defined for instant messaging and the REFER method (RFC 3515 [46]). In order to add new functionalities to the SIP protocol, many new methods are frequently developed and proposed as Internet drafts.

CLASS	DESCRIPTION
1xx	Provisional or informational : Request in progress but not complete
2xx	Success : Request has completed successfully
3xx	Redirection : Request should be tried at another location
4xx	Client error : Request not completed due to error in request, can be retried when corrected
5xx	Server error : Request not completed due to error in request, can be retried at another location
6xx	Global failure : Request has failed and should not be retried

Table 2.4: SIP Response Code Classes

SIP responses are identified by a Status-Code. A Status-Code is a three-digit integer result code as shown in Table 2.4. Many of them come from HTTP, i.e. `404 Not Found`. The first digit defines the class which the response belongs to. Thus, any response with Status-Code ranging from 100 to 199 is referred to as a 1xx response.

2.5.2 SIP Address Format

Each SIP user is uniquely identified by a *Uniform Resource Identifier (URI)*. A SIP URI can also identify a communication resource such as a mailbox on a messaging system, a PSTN number at a gateway service, a group in an organization, etc. It has a similar format to e-mail address and can be placed on web pages as a hyperlink or in e-mail messages. A SIP URI has the following format :

```
sip:dlekim@enst-bretagne.fr
sip:dlekim@134.138.228.102
```

where `dlekim` is the username and `enst-bretagne.fr` the hostname. It can also take a phone number format as shown here :

```
sip:+32-484-21-19-50@enst-bretagne.fr;user=phone
```

`user=phone` specifies that the user is using a SIP phone for establishing his call. SIP messages may be secured and encrypted (namely TLS[3]) during transport from the caller domain to the callee's one, this can be achieved with SIPS URI.

```
sips:dlekim@enst-bretagne.fr
```

2.5.3 Message Structure

Here is an example of a SIP `INVITE` request (Table 2.5) followed by a line by line description (Table 2.6).

[3]TLS, short for Transport Layer Security, is a protocol providing transport layer security mechanisms between two communicating applications. Refer to Section 2.7.4 for more information.

```
INVITE sip:bob@info.be SIP/2.0
Via:  SIP/2.0/UDP 192.108.119.243:5060
To:  bob <sip:bob@info.be>
From:  alice <sip:alice@info.be>
Call-ID: 123456789
CSeq:  1 INVITE
Contact:  <sip:alice@192.108.119.210>
Content-Length:  408
```

Table 2.5: Example of SIP `INVITE` request

This is the minimum required set of headers a SIP application needs to handle a SIP message. Many other headers may be added for the use of other functionalities which will be stated in further sections.

LINE	DESCRIPTION
`INVITE sip:bob@info.be SIP/2.0`	The request line starts with the name of the method, followed by the destination SIP URI and SIP version.
`Via: SIP/2.0/UDP 192.108.119.243:5060`	The `Via` header contains SIP version, the transport protocol, the sender's IP address and the port number (5060 is SIP's dedicated port number). SIP servers will add to the `Via` header their own address before forwarding SIP messages.
`To: alice <sip:bob@info.be>`	The `To` header contains the display name followed by the SIP URI of the destination of the SIP message.
`From: bob <sip:alice@info.be>`	The `From` header contains the display name followed by the SIP URI of the originator of the SIP message.
`Call-ID: 123456789`	The `Call-ID` header contains a unique identifier for this session. It is made up of a random identifier sometimes followed by "@" and the hostname or IP address. All requests and responses within this same session must use the same `Call-ID`.
`CSeq: 1 INVITE`	The `CSeq` (Command Sequence number) header contains an integer followed by the request method. This is incremented for each request with the same method within the same session. Both caller and callee parties maintain their own `CSeq` counts.
`Contact: <sip:alice@192.108.119.210>`	The `Contact` header contains sender's return address.

Table 2.6: SIP `INVITE` request and line by line description

Table 2.7 shows an example of a SIP `INVITE` response matching the previous request. The description is analogous to the request's one. Most of the time, SIP `INVITE` requests and responses carry a SDP body. This will be introduced in the following section.

```
SIP/2.0 200 OK
Via:  SIP/2.0/UDP 192.108.119.243:5060
To:  bob <sip:bob@info.be>
From:  alice <sip:alice@info.be>
Call-ID: 123456789
CSeq:  1 INVITE
Contact:  <sip:bob@136.146.128.9>
```

Table 2.7: Example of SIP `INVITE` response

2.6 Entity Interaction

This section describes different interactions between SIP entities.

2.6.1 Session Establishment

Session setup is the primary function of SIP. A user agent client (UAC) sends an `INVITE` request to a user agent server (UAS) to set up a session. The `INVITE` message may contain a body which is a description of the type of the session the user agent client wishes to establish. A SIP user agent initializes the `To, From` and `Call-ID` headers at the start of the session. Tags may be added to the `To` and `From` headers otherwise these headers (`To, From` and `Call-ID`) are never modified during a session and are used to uniquely identify the session referred to as a SIP call leg[4]. All SIP messages which are sent within the same call leg follow the same path. This set of headers and other media description are the minimum amount of call state that a user agent must maintain.

Figure 2.2 shows a successful SIP session establishment. The SIP session establishment is a three-way handshake. The UAC sends an `INVITE` request, receives a `200 OK` response, then sends an `ACK` request. A request failure will result in a `REQUEST/4xx` or `5xx` or `6xxx/ACK` message exchange. Zero or more provisional `1xx` responses can be sent prior to a final response `200 OK`. Once a session is set up, a media session goes on indefinitely without

[4]Another name for a SIP call leg is SIP dialog

requiring further SIP signaling message exchange. The media session is an end-to-end communication between the two UAs without involving the proxy server. Session termination happens when one of the user agents sends a `BYE` referencing a call leg and receives an `ACK`.

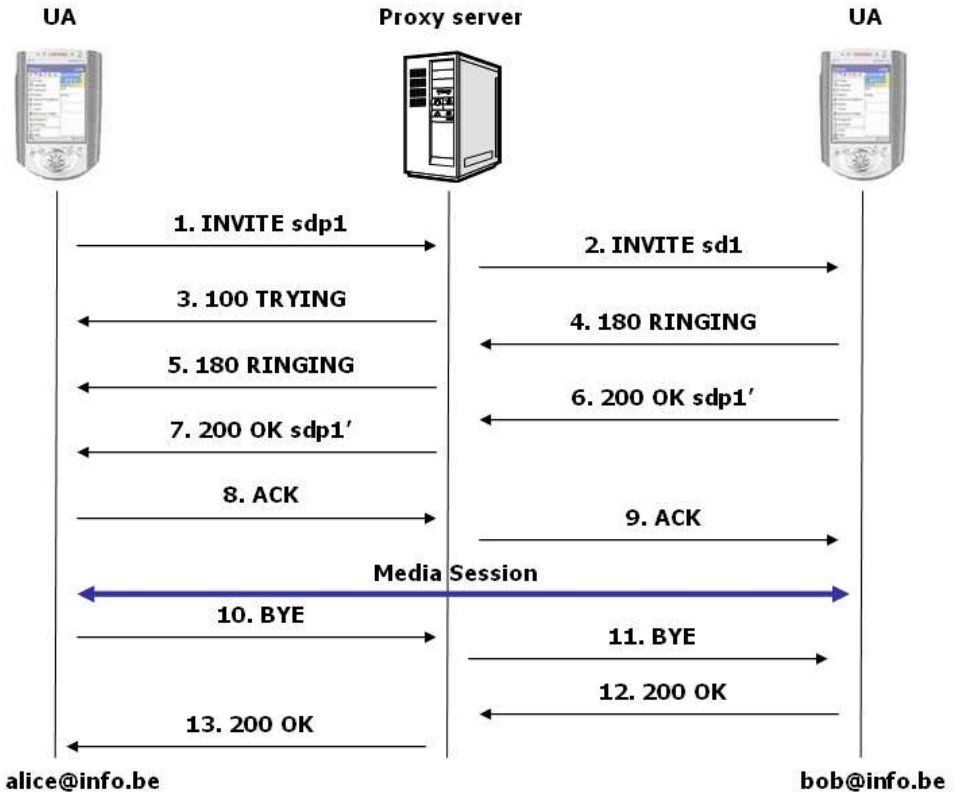

Figure 2.2: Successful SIP Session Establishment

SDP - Session Description Protocol

SIP itself does not provide media negotiation but makes it possible between two endpoints by using *Session Description Protocol (SDP)*. SDP is not considered as a true protocol but rather a text-based description language, which is described in RFC 2327 [30]. The negotiation is an offer-response model in which the caller proposes one or more media types and the other user agent accepts or declines each media session in a response. The offer is

made in the initial `INVITE` message and the response is received in the `200 OK`. The callee party may make his offer inside the `200 OK` response then the caller responds by sending an `ACK`. The SDP content encapsulated in the SIP body message contains the media type, IP address, port and codec to use with each media streams. More than one codec can be specified for each media type. For each of these codecs proposed, the user agent should be prepared to receive media for the duration of the session.

Here is an example of SDP content (Table 2.8) followed by a line by line description (Table 2.9) :

```
v=0
o=dlekim 0 0 IN IP4 192.108.119.210
s=
c=IN IP4 192.108.119.210
t=
m=video 22222 RTP/AVP 26 20 31
m=audio 22224 RTP/AVP 0 3 4 5 6 8 15 18
a=rtpmap:26 JPEG/9000
a=rtpmap:0 PCMU/8000
```

Table 2.8: Example of SDP Content

A set of headers are added to SIP messages to provide information about the body such as :

```
Content-Type :  application/sdp
Content-Length :  358
```

The `Content-Type` header indicates the type of body and the `Content-Length` header contains the length of the SIP's message body in bytes. A `Content-Length` of 0 means there is no message body. We should stress here that SIP is totally session description format independent. Although SDP is the most common format for session description, SIP may use other formats as well.

LINE	DESCRIPTION
`v=0`	Current version number of SDP - Not used by SIP
`o=dlekim 0 0 IN IP4 192.108.119.210`	Origin - Not used by SIP
`s=`	Subject - Not used by SIP
`c=IN IP4 192.108.119.210`	Connection - network, address type and address
`t=`	Time - start and stop time - Not used by SIP
`m=video 22222 RTP/AVP 26 20 31`	Media - media type (video), port, type and payload type number (Section 2.4.1)
`m=audio 22224 RTP/AVP 0 3 4 5 6 8 15 18`	Media - media type (audio), port, type and payload type number (Section 2.4.1)
`a=rtpmap:26 JPEG/9000`	Attribute - rtpmap lists attribute of RTP/AVP video profile including codec and sampling rate
`a=rtpmap:0 PCMU/8000`	Attribute - rtpmap lists attribute of RTP/AVP audio profile including codec and sampling rate

Table 2.9: SDP Content and Line By Line Description

2.6.2 Registration

In order to call a UAS from its SIP URI, a UAC need to register first to a registrar server of its domain. Its role is to maintain mapping between SIP URI and IP address provided by the `Contact` field. This information is kept within a location server which updates its database in the case of a new registration or in the case of a SIP URI that has been mapped with a new IP address (Figure 2.3). Thus, any SIP user can always contact any other registered SIP user wherever he is.

Table 2.10 shows a SIP `REGISTER` request. The first line contains the domain of the registrar the user wish to register to. `Max-Forwards` indicates the maximum number of hops a request can pass through to reach its final destination. The `Expires` header contains the time in seconds after which the message expires.

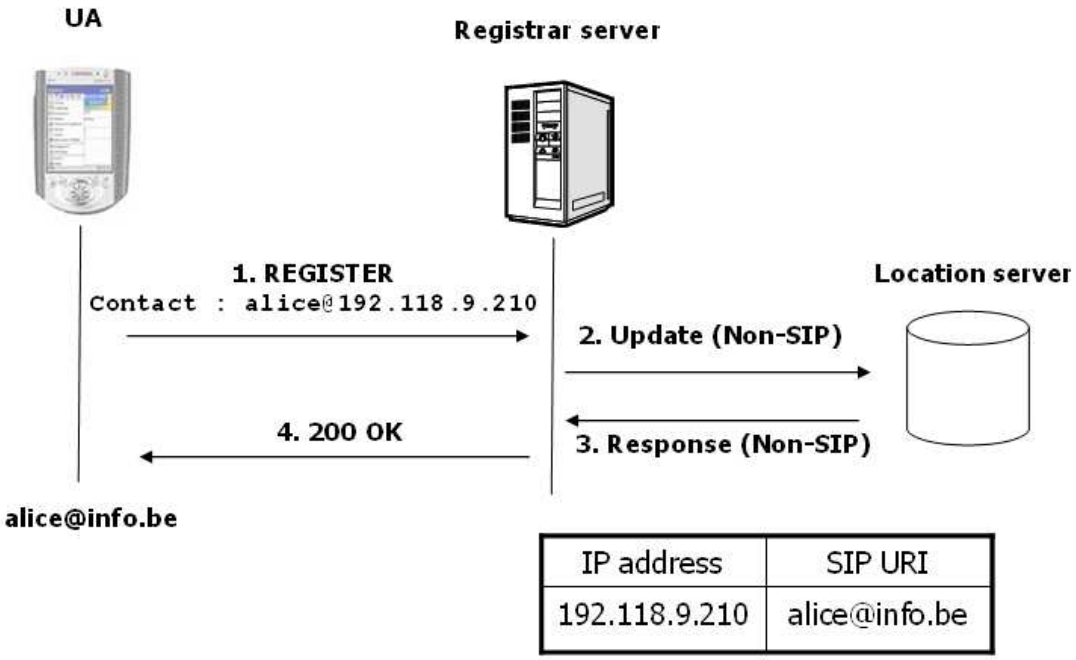

Figure 2.3: SIP Registration

```
REGISTER sip:info.be SIP/2.0
Via:  SIP/2.0/UDP 192.108.119.243:5060
Max-Forwards:  70
From:  alice <sip:alice@info.be>
To:  alice <sip:alice@info.be>
Call-ID: 123456789
CSeq:  1 REGISTER
Contact:  <sip:alice@192.108.119.210>
Expires:  3600
Content-Length:  0
```

Table 2.10: Example of SIP `REGISTER` request

2.7 Security

SIP's use of intermediaries and its end-to-end user operation make it difficult to secure. In order to meet these needs, SIP provides many different security mechanisms and hop-by-hop encryption. Instead of inventing new security

mechanisms specific to SIP, SIP reuses existing security measures taken from HTTP and SMTP. Here are some of these security measures.

2.7.1 Hop-by-Hop encryption

We know that end-to-end encryption message is the best way to preserve confidentiality and integrity. However, we cannot apply this scheme to SIP as SIP headers should be read and even modified by proxy servers. Therefore endpoints should first require authentication of proxy servers before sending their requests. Thus low-layer security mechanisms for SIP are recommended, which will be detailed in Section 2.7.4.

2.7.2 Authentication and Authorization

Before processing any SIP requests, a proxy server may require the initiator to authenticate itself and vice versa. This authentication procedure is also applicable when a UAS receives a request from a UAC. Once the initiator provides assurance of its identity, the recipient of the request may check whether it has the authorization for the services it requests for. SIP defines a header field called `Authorization` which contains authentication credentials of a user agent. SIP also defines the `Proxy-Authorization` header field allowing a user agent to identify itself to a proxy [25].

2.7.3 Privacy, Integrity and Confidentiality

MIME bodies are carried within SIP messages. The MIME standard provides securing mechanisms called S/MIME. S/MIME provides the following cryptographic security services for electronic messaging applications : authentication, message integrity and non-repudiation of origin using digital signature, and privacy and data security using encryption [7]. S/MIME can provide some degree of end-to-end security through tunneling. The SIP tunneling consists in encapsulating the whole SIP message within a MIME body and then apply the MIME security mechanism as the same way as typical SIP bodies [25]. This would give supplementary processing for proxy servers which have to decapsulate S/MIME bodies before routing them.

2.7.4 Transport and Network Layer Security

Transport Layer Security (TSL) and IP Security (IPSec) protocols provide respectively transport and network layer security guaranteeing message confidentiality and integrity.

TLS is a protocol providing security mechanisms between two communicating applications. TLS is mainly used over TCP and is specified as the desired protocol within the SIP `Via` header or through the `sips` URI. TLS is designed for hop-by-hop application model. Hence, if a user agent sends a request using TLS to a proxy server, it will have no guarantee that its request will be sent end-to-end [25].

IPSec is a set of protocols that provides security at the network layer protocol. SIP specification on security mechanisms does not give IPSec profile to be used in SIP-based network but just recommend IPSec for securing the network layer.

2.8 SIP vs H.323

H.323 is a popular standard defined by ITU (International Telecommunications Union) whereas SIP is defined by IETF. H.323 is not considered as a protocol by itself but rather specifies how to use components, protocols and procedures to implement multimedia conferences on a Local Area Network (LAN) [1]. H.323 and SIP are direct competitors even if H.323 has a larger piece in the current VoIP market due to its longer presence. We highlight here the most important differences between SIP and H.323 :

- H.323 has a higher complexity level than SIP : H.323 specifies hundreds of elements while SIP defines only 37 headers, with each a small number of values and parameters.

- H.323 is vertically integrated : H.323 provides a complete set of protocols for multimedia conferencing such as signaling, registration, admission control, transport and codecs while SIP only provides session initiation and management, and should rely on other protocols to offer services.
- H.323 is not very scalable : As H.323 has been designed to work within a single LAN, it has scalability problems for large numbers of domains.
- Message format : H.323 uses binary representation for its messages while SIP uses simple text encoded messages which are much easier to implement and debug.

Because of its manageability, reliability and interoperability with PSTN, H.323 is still widely used in the enterprise market and will continue to exist for some time. Many serious implementers propose a SIP to H.323 interworking function/gateway as a solution to the interoperability problem.

Chapter 3

Presence Service

Presence service basically known as ”buddy list” indicating a user's status is continuously widely used in instant messaging (IM) softwares like AOL, MSN Messenger, Yahoo!, ICQ,... and will reach more than 229 million registered users in the world by 2005 (Source : IDC). The use of presence service will continue to increase with the Internet growth and is likely to become omnipresent in the future. On the one hand, presence service allows a set of users to receive a customized amount of information. On the other hand, it allows third-party to exploit the presence information and customize the service according to the user's needs and preferences defined in the presence information [15]. Nowadays, most of these technologies are based on proprietary protocols and are unable to interoperate between different service providers.

Thanks to its design, SIP is particularly well suited as a presence protocol. Indeed, presence information is already contained in SIP location services in the form of registrations and SIP messages can be routed from any user in the network to the server which handles the registration state of a particular user [43]. For those reasons, a new SIP extension known as SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) has been specified by the IETF in order to provide a common standard among instant messaging and presence services [13].

3.1 Standards and Protocols

Since it became obvious that presence and instant messaging solutions based on proprietary protocols would be unable to communicate with each other, the IETF has been working on a standardized solution. We present here the standards resulting from different working groups [44] :

3.1.1 IMPP

In 1999, a new working group called *Instant Messaging and Presence Protocol (IMPP)* was formed by the IETF. Its purpose was to define requirements and a general framework for presence and instant messaging systems. The IMPP working group met many difficulties in defining a common standard and therefore, preferred to let the market decide. Three new working groups were formed with each their own protocol : Application Exchange (APEX), Presence and Instant Messaging Protocol (PRIM) and SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions). PRIM was an early proposal and was discontinued because it was obsolete. APEX was a XML-based protocol but is no longer used today. Most of the proponents of APEX give their support on eXtensible Messaging and Presence Protocol (XMPP) which is also based on XML technology. SIMPLE and XMPP remain the two protocols being developed within IETF for presence and instant messaging applications.

Although IMPP working group failed to reach a consensus, it kept working on standard solutions to enable interoperability between instant messaging systems. IMPP released several RFCs :

- RFC 3859 : A common profile for presence (CPP)
- RFC 3860 : A common profile for instant messaging (CPIM)
- RFC 3862 : A common extensible instant message format (message/cpim)
- RFC 3863 : A common extensible presence information data format (PIDF)

The Common Profiles for Instant Messaging (CPIM) and Presence (CPP) specifications define a set of operations and parameters to achieve interoperability between different Instant Messaging and Presence protocols which meet Instant Messaging / Presence Protocol Requirements [27]. We outline that CPIM and CPP specify the semantics and not the syntax. The MIME content-type message/cpim is the common message format for CPIM-compliant messaging protocols while PIDF is the common message format for CPP-compliant presence protocols. An application is IMPP-compliant if it meets both CPIM and CPP requirements defined in [27].

3.1.2 SIMPLE

As the name indicates, SIMPLE (SIP for Instant Messaging and Presence Leveraging Extensions) is a SIP-based protocol. The SIMPLE working group aims at developing an open standard and IMPP-compliant protocol which adds presence and instant messaging functionalities to SIP. SIMPLE propose the usage of SIP as a presence protocol, this can be accomplished through an instantiation of the general event notification framework defined for SIP [41]. Today, many industry leaders, such as Microsoft and IBM, have strongly embraced both SIP and SIMPLE, using them in respectively Windows Messenger and Lotus IM. The 3GPP has also decided to use SIMPLE to give mobile devices presence capabilities.

3.1.3 XMPP

The eXtensible Messaging and Presence Protocol (XMPP) is chartered by IETF since 2002. According to [39], *"XMPP core is a protocol for streaming eXtensible Markup Language (XML) elements in order to exchange structured information in close to real time between any two network endpoints. While XMPP provides a generalized, extensible framework for exchanging XML data, it is used mainly for the purpose of building instant messaging and presence applications that meet the requirements of [27]."*

XMPP has been originally designed from the Jabber protocol which is an open and standard XML-based protocol designed by the Jabber Software Foundation. Its goal is to produce a protocol based on XML technologies that enables any two entities on the Internet to exchange messages, presence, and other structured information in close to real time [14].

Today, softwares based on Jabber protocol are widely spread over thousands of Internet servers and are mainly used in information systems within big-name companies, such as Hewlett-Packard and Intel, and administrations for data exchange between applications. There is clearly no doubt that XMPP is today the strongest SIMPLE's competitor. But, according to industry analysts, SIMPLE has yet to hit the ground with large-scale, real-world deployments [33].

3.2 Concepts and Model

RFC 2778 [28] was produced by the IMPP to provide a common vocabulary and present an abstract model for presence and instant messaging. This model aims at developing an open standard, interoperable and protocol independent. We will introduce the IMPP presence service model here and the instant messaging service model will be explained in the following chapter.

Presence is the service that allows a user to be informed about the reachability, availability, and willingness of communication of another user [15].

In other words, the presence service allows users to indicate their current status such as online, busy, away, ... but also allows users to provide information about their communication means, for example, whether they can use audio, video or instant messaging capabilities. Figure 3.1 shows an overview of a presence service.

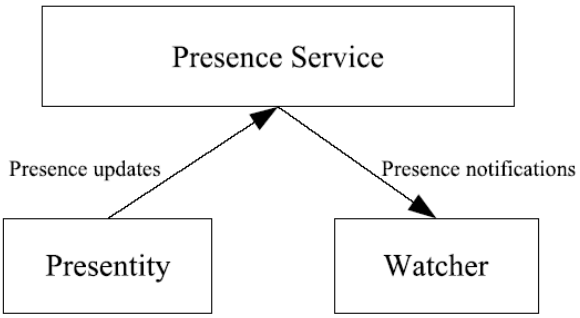

Figure 3.1: Overview of Presence Service

The presence service model is composed of two types of logical entities :

- **Presentity (Presence entity)** : logical entity that provides presence information called notification to the presence service
- **Watcher** : logical entity that receives presence information about presentities from the presence service

We emphasize that these two types of logical entities and their functionalities may be combined in an implementation but should be managed separately. The presence service's role is to store and distribute presence information to the presentities or watchers that request it. Watchers are divided into two distinct sets as displayed in Figure 3.2, called the fetchers and subscribers. A fetcher asks for presence information, but has not subscribed to the presence service. Thus, the presence service does not send notification to fetchers, presence information are only sent when requested by the fetcher. A particular type of fetcher is a poller. A poller asks for presence information to the presence service on a regular basis. A subscriber is a watcher that asks the presence service to be notified of all current and future changes about one or more presentities [20].

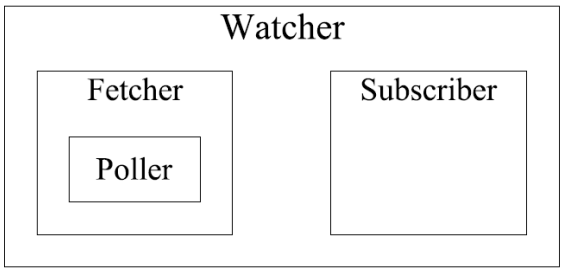

Figure 3.2: Type of Watchers

3.3 Presence with SIP

We will develop and explain the presence part of SIMPLE in this section referred to as *A Presence Event Package for the Session Initiation Protocol [43]*.

3.3.1 Architecture

Before describing the presence architecture, we may define two new concepts : *Presence User Agent (PUA)* and *Presence Agent (PA)*.

- **Presence User Agent (PUA)** : A PUA provides presence information about a presentity. This can be done implicitly through a side effect such as a `REGISTER` request or explicitly through the publication of presence information. For a given presentity, there may be several PUAs assigned to it. Thus, a user may have multiple devices (such as PDA and laptop), each of them supplying a part of the overall presence information for a presentity. A PUA provides presence information to a presence system but remains outside of it, therefore, a PUA does not subscribe neither send notifications to a presence service.

- **Presence Agent (PA)** : A PA is a SIP User Agent able to receive subscriptions, respond to them and generate notifications containing presence information. A PA must have access to presence information manipulated by PUAs for the presentity. This can be done by co-locating the PA with the proxy/registrar or by co-locating a PA with

the PUA of the presentity. However, there are no recommendations about the location of PA functions since these specifications are one of the possibilities among many others.

Figure 3.3 presents an example of SIP presence architecture. There is a presentity named Alice. Three PUAs (PDA, laptop and desktop) provide a part of the presence information about Alice. For example, the laptop as well as the desktop know if Alice is logged or not. And if Alice is logged, we know what type of communication she is using. Each of these three PUAs send their part of information to the PA which collects it and obtains a global view about Alice's presence. Bob and Cynthia are watchers thus, they request presence information from the PA about a presentity or either another watcher. The watcher should first subscribe to the PA via a SIP `SUBSCRIBE` method and receive a notification containing the requested information from the PA via a SIP `NOTIFY` method.

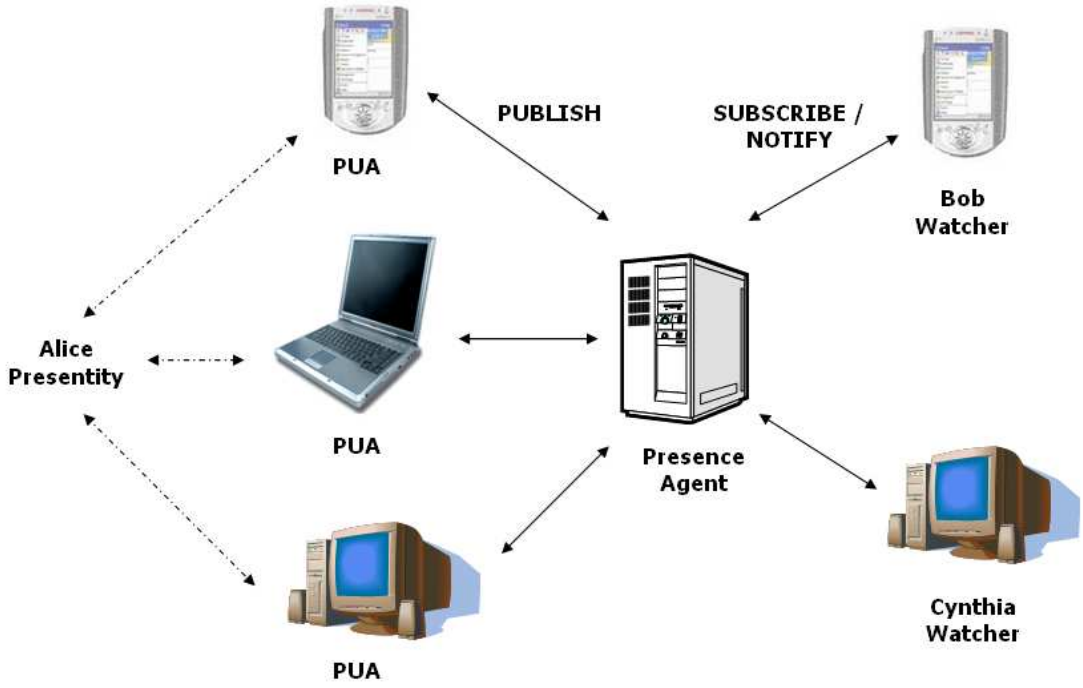

Figure 3.3: Example of SIP Presence Architecture. Source : [15]

3.3.2 PIDF

The *Presence Information Data Format (PIDF)* is a common presence data format for CPP-compliant presence protocols specified by IETF in [21]. In other words, it allows presence information to be transferred among applications based on CPP-compliant presence protocols without any modification. In that way, PIDF has been designed from a minimal model for a maximum interoperability but PIDF can also be extended to obtain a more accurate information about the presence information of a presentity. PIDF is a XML-encoded data format that can be encapsulated within SIP `PUBLISH` request and SIP presence `NOTIFY` requests. The PIDF defines a new MIME media type `application/pidf+xml` indicating the media and encoding [15].

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          entity="pres:someone@example.com">
      <tuple id="sg89ae">
          <status>
              <basic>open</basic>
          </status>
          <contact priority="0.8">tel:+09012345678</contact>
      </tuple>
</presence>
```

Table 3.1: Example of PIDF content

Table 3.1 represents the presence information about a presentity identified as `pres:someone@example.com`. Table 3.2 shows a line by line description of the PIDF content.

As PIDF provides a minimal set of presence information about a presentity, other extensions have been developed in order to supply more complete information. For example, the *Rich Presence Information Data Format (RPID)* aims at providing richer and more detailed presence information [19]. Another extension called *User agent capability extension to PIDF*

TAG	DESCRIPTION
`?xml`	Contains the version of XML and encoding charset
`presence`	Contains a XML namespace and its URI associated
`tuple`	Contains a number of elements representing the presence information
`status`	Contains one optional basic element
`basic`	Indicates the user's availability. Its values are `open` or `closed`, meaning online and offline respectively. Other statuses such as `away`, `busy`, `on the phone`, etc. may be defined
`contact`	Contains a URL of the contact address with an optional priority attribute. Its value indicates the priority level of this contact address over the others. The value of the attribute must be a decimal number between 0 and 1

Table 3.2: Line by Line Description of PIDF Content

provides information about the capabilities supported by a user agent. For instance, if Alice knows that Bob's device does not support text but well audio/video media, Alice will initiate an audio/video session instead of an instant messaging session [31].

3.3.3 Presence Publication

We know that the `REGISTER` method can implicitly supply presence information. For instance, when a user is registered to a SIP registrar, the PA sets its presence to "online" and when a user is not registered, the PA sets its presence to "offline". As described in Section 2.3, `REGISTER` method's role is to maintain a mapping between the SIP URI and the IP address. Therefore, using `REGISTER` method for presence publication is not appropriate. The IETF solved this problem by defining a new SIP method called `PUBLISH` [2]. The purpose of this new extension to SIP is to publish event state used within the SIP specific event notification framework [41]. Thus, the `PUBLISH` mechanism can be used for supporting publication of any event for which it exists an appropriate event package. However, the first application of this extension is for the publication of presence information.

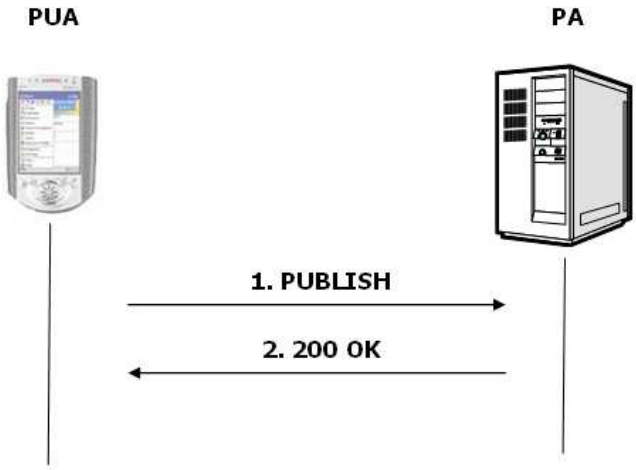

Figure 3.4: Publication of Presence Information

Figure 3.4 shows a typical flow used to publish presence information. The PUA sends a `PUBLISH` request to the PA containing a PIDF payload in order to update it with new presence information and the PA acknowledges with a `200 OK` response. Table 3.3 shows an example of SIP `PUBLISH` request. The `Expires` header field indicates the suggested duration for this event.

```
PUBLISH sip:presentity@example.com SIP/2.0
Via:  SIP/2.0/UDP pua.example.com
To:  <sip:presentity@example.com>
From:  <sip:presentity@example.com>
Call-ID: 98798798@pua.example.com
CSeq:  1 PUBLISH
Max-Forwards:  70
Expires:  3600
Event:  presence
Content-Length:  646
```

Table 3.3: Example of SIP `PUBLISH` request

3.3.4 Presence Subscription and Notification

As mentioned in Section 3.2, a watcher can fetch the presence information which means the watcher only asks for the current presence information and does not want to receive further notification. A watcher can also subscribe to the presence information, in this case the watcher will be informed of any change of the presentity's presence information. In both cases, it is implemented with a SIP `SUBSCRIBE` request and notifications are made through a SIP `NOTIFY` request. A `SUBSCRIBE` request contains an `Expires` header. The `expires` value indicates the duration of the subscription. At any time a subscription expires, another `SUBSCRIBE` message is renewed prior to its expiration.

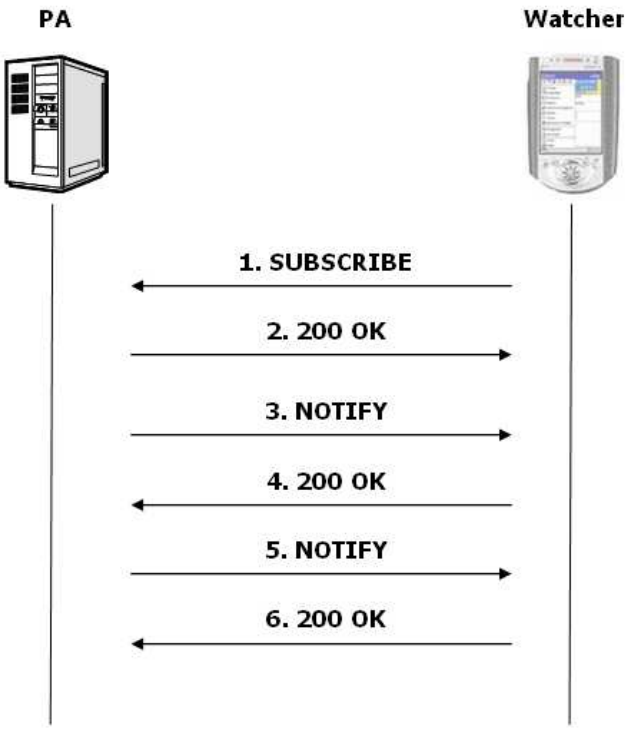

Figure 3.5: Subscription and Notification of Presence Information

Figure 3.5 shows a typical flow of messages for subscription and notification of presence information. A `SUBSCRIBE` request (1) is sent from the watcher

to the PA. The PA authenticates the watcher and checks whether it has the necessary authorization and responds with a `200 OK` (2), followed by a `NOTIFY` request (3) containing a PIDF document. The watcher replies with a `200 OK` response (4). If there is any change of the presentity's presence information, the PA sends to the watcher another `NOTIFY` request (5) with a new PIDF document. The watcher responds with a `200 OK` (6).

Tables 3.4 and 3.5 shows an example of a SIP `SUBSCRIBE` and `NOTIFY` request. We notice the `Event` header which indicates the event package used for subscription and notification. In our case, we use the `Presence` package.

```
SUBSCRIBE sip:pa@example.com SIP/2.0
Via:  SIP/2.0/UDP watcher.example.com
To:  <sip:pa@example.com>
From:  <sip:user@example.com>
Call-ID: 123456789@watcher.example.com
CSeq:  1 SUBSCRIBE
Event :  Presence
Accept :  application/pidf+xml
Expires:  600
Content-Length:  0
```

Table 3.4: Example of SIP `SUBSCRIBE` request

```
NOTIFY sip:user@example.com SIP/2.0
Via:  SIP/2.0/UDP watcher.example.com
To:  <sip:user@example.com>
From:  <sip:pa@example.com>
Call-ID: 123456789@watcher.example.com
CSeq:  1 NOTIFY
Event :  Presence
Subscription-state:  active; expires:  600
Content-Type :  application/pidf+xml
Content-Length:  584
```

Table 3.5: Example of SIP `NOTIFY` request

Chapter 4

Instant Messaging Service

Instant Messaging (IM) is the fastest growing business communication medium ever. Gartner Inc. predicts that by 2005, at least 60% of businesses will rely on IM to interact with consumers. Figure 4.1 shows the growth of IM users.

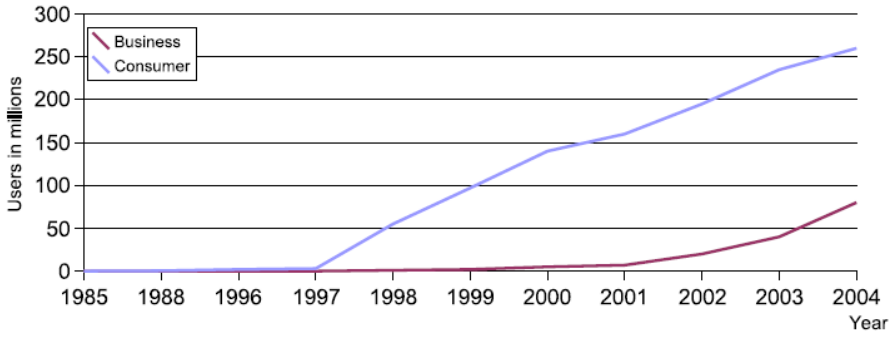

Figure 4.1: Growth of IM users. Source : IDC

Forecaster IDC estimates that about 200 million corporate users worldwide were hooked to IM by 2004, up from 5.5 million in 2000 [4]. A question we should ask is : why do people want instant messaging? People answer that IM is the immediacy of the telephone with the power of written words without the expense of long-distance phone calls. IM softwares such as AOL IM, MSN Messenger, ICQ, and Yahoo! Messenger offer online presence awareness as explained in Chapter 3 and allow users to talk in a private dialog or either conference with multi-users. We should also outline the

fabulous growth of the popular Short Message Service (SMS) on mobile phones which can be considered as a kind of IM [9]. As of today, many Internet messaging services allow messages to be delivered from computers to mobile phones and the inverse is also true. But like presence services, IM services suffer from lack of interoperability among them. Each IM service is based on its own protocol which involves that users have to agree with their contacts on which services to use or they have to subscribe to several services in order to keep in touch with all their contacts.

4.1 Proprietary Instant Messaging Systems

We will present in this section several IM systems based on proprietary protocols. The three biggest companies, America Online, Microsoft and Yahoo represent the largest part of the market. They offer almost the same functionalities and from the viewpoint of IM, they have no differences. Regarding open instant messaging systems, Jabber is the most popular and has been previously described in Section 3.1.3.

4.1.1 ICQ

ICQ (I Seek You) was founded in July 1996 by a young Israeli company named Mirabilis. At that time, the founders observed the mounting popularity of surfing and browsing with the exponential growth of users interacting with web servers. They realized that all these users were connected to the hugest world wide network, the Internet, but not interconnected. They solved the problem by proposing a new technology that would enable online users to contact each other through a peer-to-peer connection in a very straightforward and easy way. ICQ was born. ICQ was the first application to offer buddy list, presence awareness and IM functionalities. In 1998, Mirabilis's assets were acquired by AOL and about 12 million users are communicating with ICQ today [22].

4.1.2 AOL Instant Messenger

In May 1997, America Online (AOL) launched AOL Instant Messenger (AIM) freely distributed to anyone on Internet. AIM became very popular in the USA and started to struggle with its main competitor, ICQ. The competition between ICQ and AIM ended in 1998 when AOL bought Mirabilis, the company that designed ICQ. AIM offers classic services such as presence awareness, privacy controls, file transfer and chat but its particularity is that AIM is available on multiple platforms (PC, Mac, Linux, Java on Web). AIM can be accessed through PDAs and mobile phones and devices. Today, the AIM community is the world's largest instant messaging base with more than 100 million users whose 35 million are considered as active users [3].

4.1.3 Yahoo! Messenger

In June 1999, one of the world's most popular search engine, Yahoo, launched a new instant messenger called Yahoo! Messenger. Yahoo! Messenger is more than just an application to "page" each other, it also offers voice chat, voice conferencing, news, sports scores, and several types of alerts [23]. Nowadays, the number of Yahoo! Messenger's users reaches 36 million.

4.1.4 Microsoft MSN Messenger

In July 1999, Microsoft's answer to America Online's popular AIM and ICQ saw its first daylight under the name of MSN Messenger Service v1.0. MSN Messenger Service reaches over 700,000 people in its first six days of availability. At its start, MSN Messenger Service allowed access to AIM service but two days after its release, AOL modified their servers to block MSN Messenger's users. In order to build a large user base and offer a more complete range of services, MSN Messenger Service was coupled with Hotmail and Microsoft Windows. Today, we should distinguish two Microsoft instant messengers : MSN Messenger and Windows Messenger. Windows Messenger is more business focused and provides support for SIMPLE and the Exchange IM Server. However these two clients remain interoperable

because they keep using the same instant messaging network. In June 2004, MSN Messenger counted over 130 million users through the world whose 2,2 million in Belgium which is more than half of the Belgian Internet population [32].

4.2 Standards and Protocols

Instant messaging model has been developed simultaneously with the presence model by the same IETF working groups described in Section 3.1. As outlined in Section 3.1, the current presence and instant messaging systems are solutions based on proprietary protocols which were detailed in the previous section. The IETF works on standardized solutions to provide interoperability among instant messaging systems. XMPP and SIMPLE remain the two standard solutions proposed by the IETF.

4.3 Concepts and Model

We will introduce here the IMPP instant messaging model defined in RFC 2778 [28] referred to as *A Model for Presence and Instant Messaging*. As mentioned in Section 3.2, this model aims at developing an open standard, interoperable and protocol independent for presence and IM services.

According to [15], ***Instant messaging (IM) is the service that allows a user to send some content to another user in near-real time. Due to real time characteristics of instant messages the content is typically not stored in network nodes, like it often happens with other service like e-mail.***

Many other definitions include presence service in instant messaging system. In this thesis, we will maintain the previously mentioned definition. Thus, IM and presence service are two distinct services which can be perfectly combined together. For instance, users may be able to send instant messages to their contacts if they are aware of their availability and start some

interactive conversations. Instant messages will typically transport a text message but can also convey a HTML page, a picture, an audio/video file or any other files. In comparison with e-mail service, IM service is different in common usage in that the communication consists of numerous messages sent back and forth. These messages are grouped together into brief live conversation. Figure 4.2 shows an overview of the instant message service.

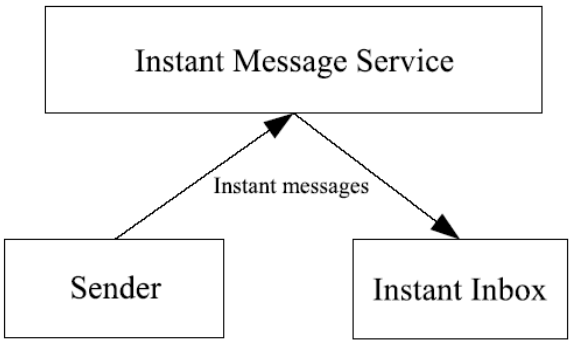

Figure 4.2: Overview of Instant Message Service

The instant message service is composed of two types of logical entities :

- **Sender** : logical entity that provides instant messages to the instant message service
- **Instant Inbox** : logical entity that receives the instant message from the instant service

Each instant message is addressed to a particular instant inbox address, and the instant message service attempt to deliver the message to a corresponding instant inbox. Typically, the instant inbox may reside at the client application although it is not recommended by the IMPP specification. The instant message service does not require a distinct server i.e the instant message service may be implemented as a direct communication between the sender and the instant inbox. The instant message service may require authentication of senders and/or instant inboxes while receiving/delivering instant messages [28].

4.4 Instant Messaging with SIP

We will develop and explain the instant messaging part of SIMPLE in this section. We may distinguish two types of model for instant message service, depending whether there is no explicit association between messages or whether there is an explicit conversation with a clear beginning and end.

We refer to a *pager-mode* model when each instant message stands alone having no relation with previous or future instant messages. This model is referred to as *pager-mode* because the way of sending instant messages uses a metaphor similar to that of a two-way pager or SMS (Short Message Service) in cellular networks.

We refer to a *session* model when each instant message is sent within a session, typically initiated by a SIP `INVITE` request and terminated by a SIP `BYE` request [15].

Most current IM clients offers both implementations. The pager model is used when the user needs to send a small number of instant messages to a single contact while the session model is used when the user joins chat groups or invite one or more contacts in a conversation.

Both models have their own implementation as their requirements and constraints are different. The implementation of the pager model is specified in RFC 3428 referred to as *Session Initiation Protocol Extension for Instant Messaging [5]* while the implementation of the session model uses a new protocol defined in the Internet-Draft *The Message Session Relay Protocol [6]*. The following sections describe the implementation of both models.

4.4.1 Pager-mode Instant Messaging

The IETF proposed a new method called `MESSAGE`, an extension to SIP that allows the transfer of instant messages between two user agents. The `MESSAGE` request is able to carry any type of MIME content as payload in

the body of the message. As each instant message stands alone in the pager model, `MESSAGE` requests do not initiate themselves a SIP call leg.

An example message flow is shown in Figure 4.3 between Alice and Bob, both are in the same domain `info.be`. Alice sends an initial `MESSAGE` request through the proxy server (1). The proxy forwards the `MESSAGE` request (2) like any other SIP request, even if the proxy does not support or understand the SIP `MESSAGE` method. Eventually, Bob will receive it (3) and answer by a `200 OK` response (4) that is forwarded to Alice through the proxy. Then Bob may send another `MESSAGE` request replying to Alice's initial message (5-6-7-8). Here, Alice and Bob play both the role of UAC and UAS. When Alice sends her first `MESSAGE` request, she is the UAC and Bob is the UAS. When Bob replies to Alice's message, he is the UAC and Alice is the UAS [15].

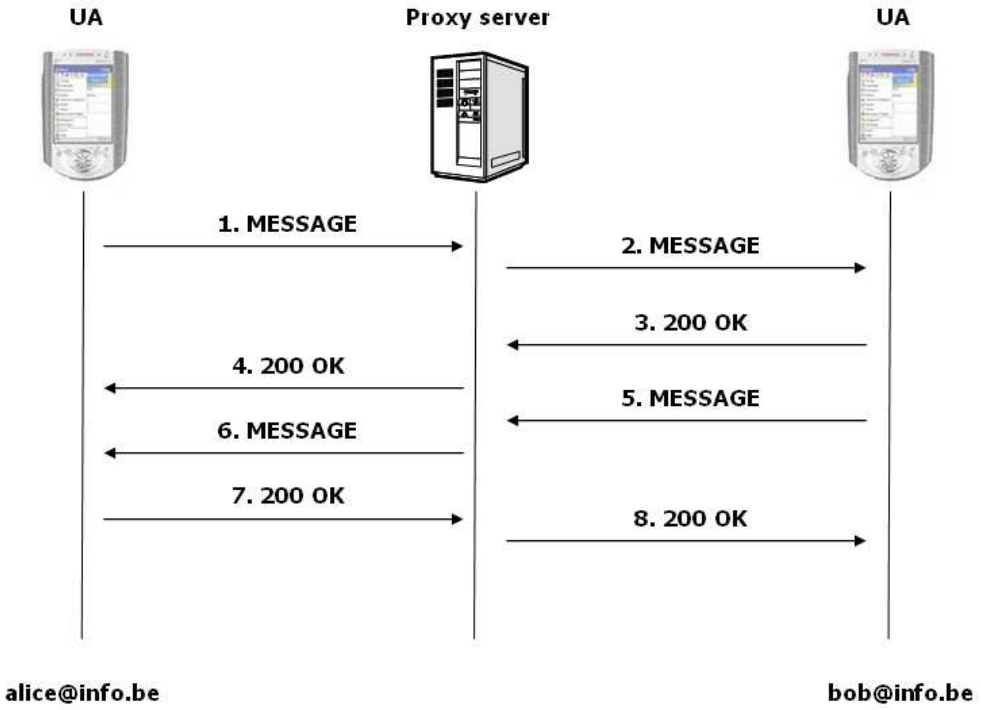

Figure 4.3: Pager-mode Instant Message Flow

When a UAC receives a `200 OK` response, it may assume that its `MESSAGE` request has been delivered to its final destination but it must not assume that the recipient has actually read or understood the instant message. If the UAC receives a `202 Accepted` response, it must not assume that its `MESSAGE` request has been delivered to its final destination but it may assume that its message has reached a gateway or an intermediary server that may eventually deliver the message [5]. Table 4.1 shows the detailed content of a SIP `MESSAGE` request.

```
MESSAGE sip:bob@info.be SIP/2.0
Via:  SIP/2.0/UDP 192.108.119.243:5060
To:  alice <sip:alice@info.be>
From:  bob <sip:bob@alice.be>
Call-ID: 123456789
CSeq:  1 MESSAGE
Content-Type:  text/plain;charset=UTF-8
Content-Length:  6

Hello!
```

Table 4.1: Example of SIP `MESSAGE` request

4.4.2 Session-based Instant Messaging with MSRP

The session model is based on the *Message Session Relay Protocol (MSRP)*. MSRP is a text-based protocol for exchanging MIME content, especially instant messages whose main feature is that it works only with protocols which offer end-to-end congestion control such as TCP and SCTP. Therefore MSRP does not run over UDP and does not place a limit on the size of instant messages. Session-based instant message mode uses the SIP `INVITE` request the same way an audio/video session is set up. For instance, Alice wishes to communicate with Bob but she does not know whether Bob has his phone or his IM client handy. Alice sends an `INVITE` request that contains a SDP body offering both voice and IM sessions with MPEG audio codec and MSRP respectively as media. As one of the SIP purposes is to separate the signaling part from the media part, MSRP will not go through

SIP proxies but should be handled from end-to-end or through MSRP relays. This is a non negligible advantage as SIP proxies does not deal with large instant messages [6] [15].

Figure 4.4 shows a successful establishment of an instant message session.

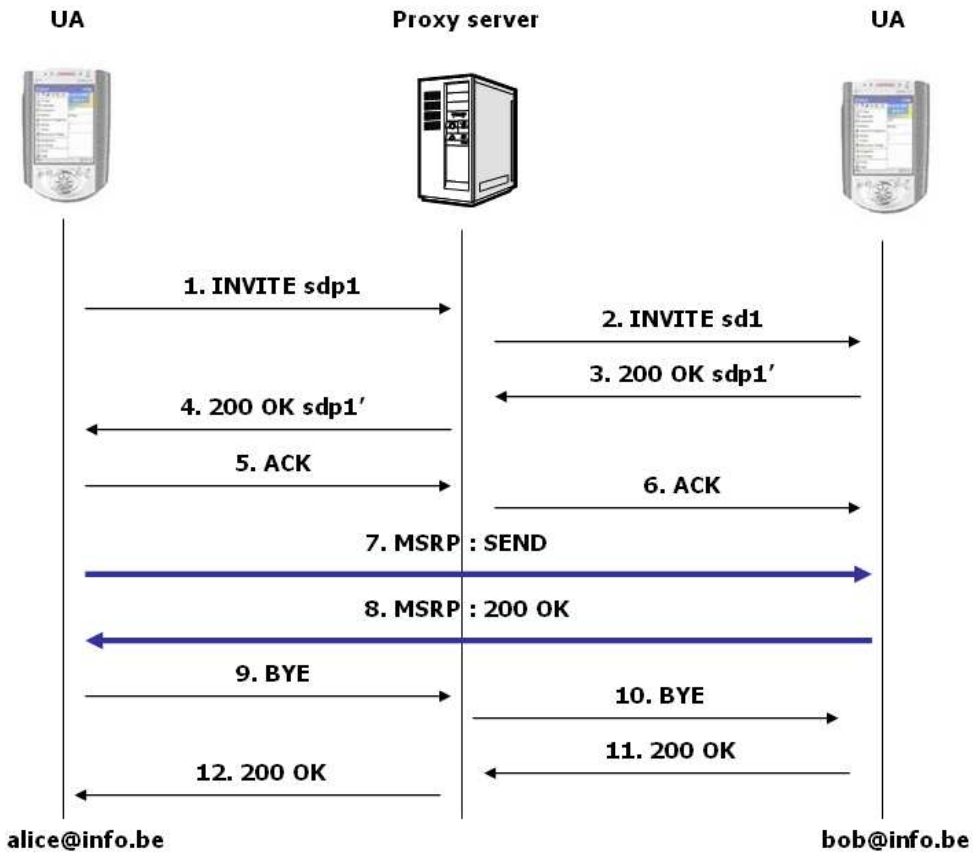

Figure 4.4: Successful Establishment of Instant Message Session

Alice, a SIP UA, sends to Bob, another SIP UA, a SIP invitation containing a SDP payload (1-2) which indicates MSRP as the media. Table 4.2 shows an example of SDP body with MSRP media. Refer to Table 2.9 (Section 2.6.1) for a line by line description. Bob accepts and replies with a `200 OK` response which includes his media choice in a SDP body (3-2). Alice acknowledges by sending an `ACK` (5-6). Alice opens a direct communication

between her and Bob by sending a MSRP `SEND` request with her initial instant message (7). MSRP `SEND` requests are used to deliver a complete message or a chunk. Table 4.3 shows an example of MSRP `SEND` request followed by a line by line description (Table 4.4). Bob answers with a MSRP `200 OK` (8). Alice may want to end the IM session by sending to Bob a SIP `BYE` request (9-10) and Bob answers with a SIP `200 OK` (11-12).

MSRP uses its own URLs to address MSRP resources : `msrp` and `msrps`. `msrps` is different from `msrp` in that it uses secure TLS connection over TCP. a MSRP URL has the following format where `bob.info.be` is the username, `8888` the host port and `9di4ea` the resource [15] :

`msrp://bob.info.be:8888/9di4ea`

```
v=0
o=alice 2890844557 2890844559 IN IP4 alicepc.info.be
s=
c=IN IP4 alicepc.info.be
t=0 0
m=message 7777 msrp/tcp *
a=accept-types:text/plain
```

Table 4.2: Example of SDP Content with MSRP media

```
MSRP d93kswow SEND
To-Path:msrp://bob.info.be:8888/9di4ea;tcp
From-Path:msrp://alicepc.info.be:7777/iau39;tcp
Message-ID: 12339sdqwer
Content-Type:  text/plain
```

Table 4.3: Example of MSRP `SEND` request

LINE	DESCRIPTION
`MSRP d93kswow SEND`	The request line starts with `MSRP` followed by a transaction identifier and the name of the method
`To-Path : msrp://bob.info.be:8888/ 9di4ea;tcp`	The `To-Path` header contains the path of URLs to the destination
`From-Path : msrp://alicepc.info.be:7777/ iau39;tcp`	The `From-Path` header contains the path of the URL's originator
`Message-ID : 12339sdqwer`	The `Message-ID` header contains a unique identifier used to correlate responses and status reports with the original message

Table 4.4: MSRP Content and Line By Line Description

Chapter 5

Mobility Support

As of today, we are living in a world where Internet is omnipresent and becomes more than essential in our every day life. The diversity and usage of Internet technologies going from the most simple ones like web browsing and e-mailing to real time multimedia applications are far from what we expected a few years ago. Due to this fast exponential growth, improvements such as introduction and development of QoS (Quality of Service) have been made. Furthermore, Internet users are demanding to be reachable anywhere at any time. The increasing mobile computer usage introduces high requirements from the future Internet network to provide mobility support.

But mobility does not only mean being available anywhere but also means that a user should be able to get access to the services he subscribed to, no matter where he is. Mobility also means that a user should be able to maintain his session while moving between networks without being interrupted.

Direct applications of mobility management in packet switched networks are used in standards such as General Packet Radio Service (GPRS) and more recently, Universal Mobile Telecommunication System (UMTS).

This chapter aims at explaining the different types of mobility and introducing two solutions. The first one is Mobile IP and the second one is based on SIP.

5.1 Types of Mobility

As stated in [18], there exist four modes of mobility :

- **Terminal mobility** : allows a device to move between IP subnets, while continuing to be reachable for incoming requests and maintaining sessions across subnet changes.
- **Session mobility** : allows a user to maintain a media session even while changing terminals.
- **Personal mobility** : allows to address a single user located at different terminals by the same logical address.
- **Service mobility** : allow a user to maintain access to his services even while moving or changing devices and network service providers.

5.2 Mobile IP

In order to solve mobility issues in IP-based networks, the IETF chartered a new working group called Mobile IP. Mobile IP handles the mobility at the network layer level and is considered as the *de facto* standard for terminal macro-mobility which refers to movement between networks while micro-mobility refers to the case where the user moves within the same administrative domain. Mobile IP makes mobility transparent to the higher layers and allows the maintenance of ongoing applications by enabling users to keep the same IP address while roaming between networks [34]. The following section will give an overview of Mobile IPv4 architecture.

5.2.1 Architecture

IP packets are routed from a source endpoint to a destination through routers which forward them from incoming network interfaces to outbound network interfaces. Routers rely on routing tables which contains the outbound interface for each destination IP address. Thus, the IP address of a packet specifies the IP node's point of attachment to the network. On the

one hand, transport layer connections should be maintained while node is moving. But as TCP connections are identified by IP addresses and port numbers of both source and sink, changing any of them would result in a disconnection. In the other hand, a packet can be delivered to the correct Mobile Node's point of attachment if the network number contained inside the mobile node's IP address is corresponding to the new point of attachment. To solve this problem, Mobile IP assigns two IP addresses to the Mobile Node : a fixed home address and a care-of address that changes at each new point of attachment [40].

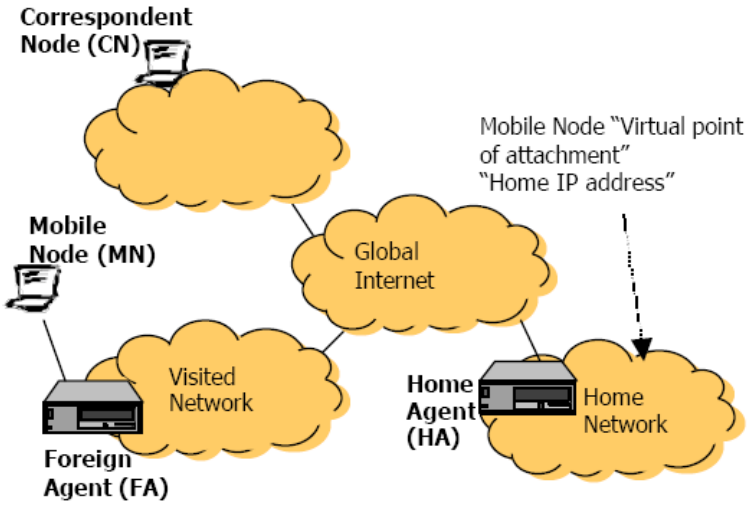

Figure 5.1: Mobile IP Architecture. Source : [24]

As shown in Figure 5.1, a Mobile IP enabled network is composed of four basic entities :

1. The Mobile Node (MN)
2. The Foreign Agent (FA)
3. The Home Agent (HA)
4. The Correspondent Node (CN)

Mobile IP introduces two new entities called the *Home Agent* and the *Foreign Agent*. If considering that the *Mobile Node* remains within the home network, the Home Agent just needs to route the packets to the Mobile Node's point of attachment. Whenever the Mobile Node leaves its point of attachment, it registers its new care-of address with its Home Agent. Then using the new care-of address, the Home Agent will route the traffic directed to the Mobile Node, which is now attached to a foreign network, to a Foreign Agent. Every packet header destined to the Mobile Node should be modified/extended by the Home Agent so that the destination header indicates the new care-of address. This modification of the packet header is sometimes called *redirection*. When the packet, forwarded by the Foreign Agent, arrives at the Mobile Node, the reverse transformation is performed so that the packet destination IP address contains the home address.

The redirection is achieved by encapsulating the original data packet into a new IP packet with the care-of address as destination IP address. This encapsulation is better known under the name of *tunneling* in the way that the original packet is hidden by the new headers while the encapsulated IP header is totally ignored during the redirection. Once the Foreign Agent receives the encapsulated packet, it takes out the original packet and routes it to the Mobile Node where it will be processed properly by TCP or others higher layer level protocols. Tunneling may be accomplished using IP-IP encapsulation or Generic Routing Encapsulation (GRE) [24].

Triangular Routing

Mobile IP is based on a *triangular routing* model as shown in Figure 5.2. Triangular routing means that packets take different paths depending on whether they are routed from or to the Mobile Node. When coming from the *Correspondent Node* and directed to the Mobile Node attached to a foreign network, packets are routed through the Home Agent. The Home Agent will then encapsulate the packets and sends them through the Mobile IP tunnel to the Foreign Agent. Once received by the Foreign Agent, the

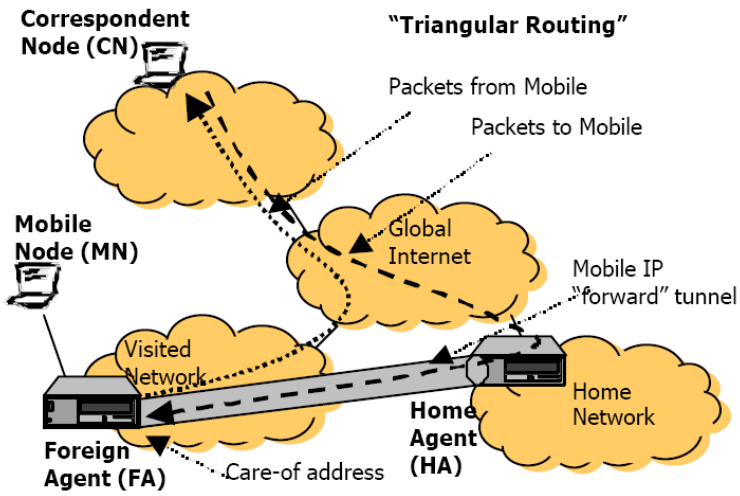

Figure 5.2: Triangular Routing. Source : [24]

original data packets are extracted and sent to the Mobile Node.

When packets are coming from the Mobile Node and directed to the Correspondent Node, there is no need to use tunneling. Since the Correspondent Node is supposed to have a public IP address, packets sent from the Mobile Node may be directly routed to the Correspondent Node, bypassing the Home Agent [24].

Agent Discovery

Home Agents and Foreign Agents regularly broadcast *agent advertisements*. These agent advertisements carry information about default routers, just as before, but also information about one or more care-of addresses. But an agent advertisement can also performed other functions as defined in [40] :

- allows the detection of Mobile Nodes
- informs the Mobile Node about special features provided by Foreign Agents, for instance, other encapsulation techniques
- lets Mobile Nodes determine whether they are in home network or foreign network

- lets Mobile Nodes determine the network number and status of their link to Internet

If a Mobile Node is visiting a foreign network and need to get a new care-of address without waiting for the periodic agent advertisements broadcast, it can broadcast itself a request that is answered by a Foreign Agent.

Registration

Once a Mobile Node receives a new care-of address from the Foreign Agent, it has to inform its Home Agent. The registration procedure starts when the Mobile Node sends to its Home Agent a registration request containing the care-of address. When the Home Agent receives and approves the request, it updates its routing tables and sends back a registration reply to the Mobile Node. The update process also called *binding update* consists in maintaining a mapping between the home address and the care-of address of the Mobile Node before the registration life-time expires.

5.2.2 Limitations

Due to its triangular routing model, Mobile IP adds traffic delay for packets directed to Mobile Node but not from the Mobile Node. This would considerably increase the latency which is not acceptable for delay sensitive multimedia applications but also place a heavy load on the Home Agent which has to forward all packets sent by the Correspondent Node to the Foreign Agent. Another issue is that Mobile IP encapsulation typically adds 20 bytes of overhead to each packet. Furthermore tunneling hides QoS information within the original packet and therefore decreases routing efficiency [18].

Some Internet routers and firewalls expect packets originating from a topologically correct subnet to use the care-of address. Thus a Foreign Agent is unable to send directly packets to a Correspondent Node as the source IP address contains the home address. This phenomena is known as *ingress filtering*. The *reverse tunneling* technique has been proposed to solve this

problem which means that the Foreign Agent sends back the packets to the Home Agent via the tunneling mechanism and keeps using the care-of address as source IP address [45]. However this also adds delay in traffic and the Home Agent and Foreign Agent are likely to become bottlenecks as reverse tunneling implies heavy load on these two agents.

5.2.3 Mobile IPv6

Mobile IPv6 has been designed to overcome Mobile IPv4 inefficiencies and take advantage of IPv6. Here is the list of some improvements made for Mobile IPv6 [11] :

- IPv6 structure allows a larger space of addresses.
- Mobile IPv6 does not implement Foreign Agents anymore as it should work in any location without support required from the local router.
- Route optimization is now fully integrated and fundamental part of Mobile IPv6 rather than an extension like in Mobile IPv4. Each Correspondent Node is equipped with a binding cache and every time a Mobile Node detects that a Correspondent Node is not aware of its location, it sends a binding update. When receipting the new binding update, the Correspondent Node updates its binding cache and the next time it has to send packets to the Mobile Node, it will use directly the care-of address instead of the home address avoiding the triangular routing.
- Encapsulation is not applied anymore in Mobile IPv6. Packets sent to a Mobile Node away of its home network use IPv6 routing header which contains, in addition of its home address, the Mobile Node's care-of address. This reduces the overhead due to IP encapsulation and thereby no tunneling is needed anymore.
- Mobile IPv6 provides support for Mobile Nodes to coexist with ingress filtering firewalls.
- Mobile IPv6 provide security mechanism using IPSec.

5.3 Mobility Support Using SIP

We already know that SIP supports personal mobility as it allows users with multiple devices to be identified by a unique SIP URI. A user may have a PDA, laptop or a wireless device that he can use at the same time or in alternation. Thanks to forking proxies, the user may be reachable independently of the device he is currently using. His choice remains transparent to the calling parties.

We present here a SIP based solution proposed in [18] and [17] which handles the terminal mobility at the application layer level. The advantages is that application layer mobility does not imply changes neither at the operating system nor at the lower layers for any of the users or entities and therefore can be widely deployed in an easier way than Mobile IP.

5.3.1 Mobile Node Registration

We assume that the Mobile Host[5] belongs to a home network where a SIP server, for instance, a registrar co-located with a redirect server, receives all registration requests each time the Mobile Host acquires a new IP address (Figure 5.3). Thus, the Mobile Host sends a SIP re-`REGISTER` request. This is similar to the Home Agent registration procedure in Mobile IP.

[5]Node is synonymous with Host

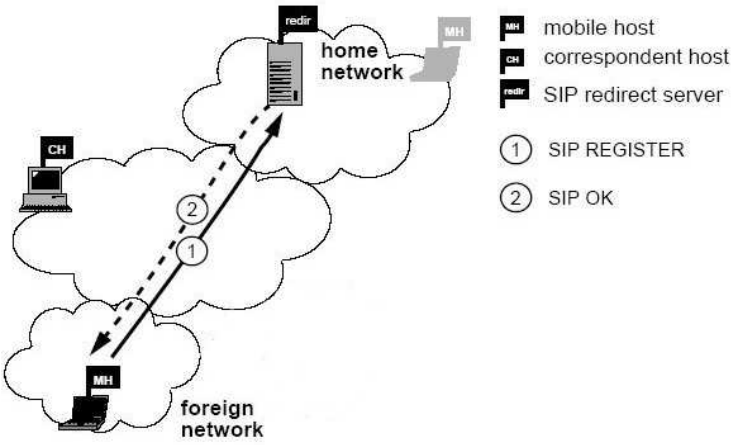

Figure 5.3: Mobile Host Registration. Source : [17]

5.3.2 Pre-Call Mobilty

Once the Mobile Host updates its registration at the registrar, every SIP `INVITE` request (1) sent from the Correspondent Host to the Mobile Host is received by the redirect server which knows the Mobile Host's current location. The redirect server sends back to the Correspondent Host a SIP `302 MOVED TEMPORARILY` response (2) with the Mobile Host's current IP address in the `Contact` header. The Correspondent Host will then redirect its SIP `INVITE` request (3) using the Mobile Host's current IP address. The Mobile Host answers with a SIP `200 OK` response then data may eventually be transferred (5). Figure 5.4 illustrates the pre-call mobility with SIP.

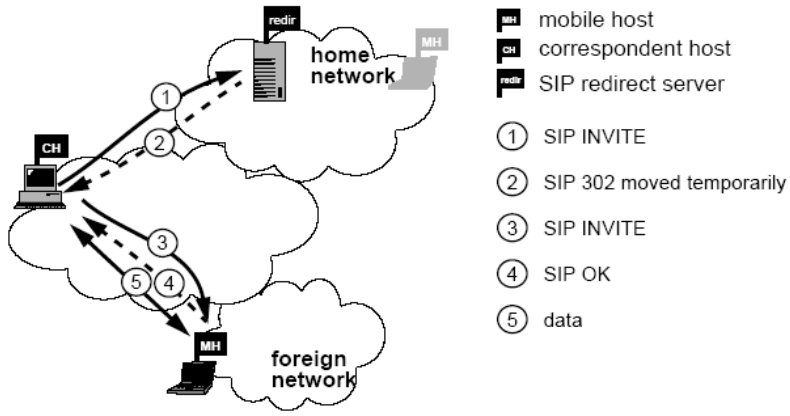

Figure 5.4: Pre-Call Mobility with SIP. Source : [17]

5.3.3 Mid-Call Mobilty

The most difficult part of SIP mobility is the mid-call mobility where the Mobile Host changes location during an active session. Figure 5.5 illustrates the mid-call mobility with SIP. When the Mobile Host moves during a session, it must send a SIP `INVITE` request (1) to the Correspondent Host with a new session description and it must also specify in the `Contact` header its new IP address. Depending on the capacity of the link, the Mobile Host may preferred to use, for instance, a more appropriate codec that it indicates in the SDP body. Once the Correspondent Host receives the SIP `INVITE` request and has updated its routing tables with the Mobile Host's new IP address, it replies with a SIP `200 OK` (2). Then a session may be re-established between the Mobile Host and the Correspondent Host (3).

In the case we have a wideband access, the hand-off delay is equal to the propagation delay plus a few milliseconds. But in the case of a narrowband access, the hand-off delay may be of several tens of milliseconds. A RTP translator can be used during the hand-off in order to avoid disruption in the media session. To perform this, the Mobile Host indicates the RTP translator's IP address as destination in the SDP content. Hence the Cor-

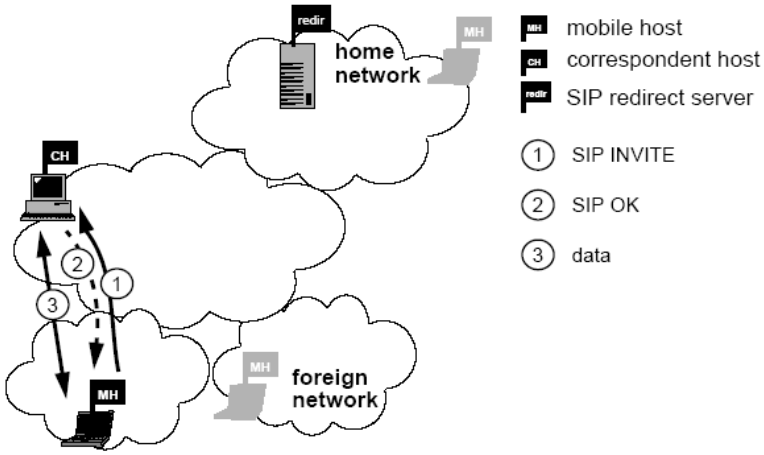

Figure 5.5: Mid-Call Mobility with SIP. Source : [17]

respondent Host will send packets through the RTP translator which will redirect them to the Mobile Host's current location. Any duplicate packets are handled by RTP. The RTP translator can also buffer media packets and transmit them to the Mobile Host's new location once the hand-off is over.

5.3.4 SIP Mobility Support with Mobile IP

As TCP uses IP addresses and ports from both source and sink to maintain connections between two endpoints, SIP mobility is not suitable for TCP connections. The best solution to offer maximal mobility support is to combine both SIP and Mobile IP [17]. SIP provides mobility support for real-time communication over UDP while Mobile IP would be used for long-lived TCP connections such as telnet or FTP (File Transfer Protocol). The Mobile Host will be given the choice to use its home address or care-of address. When establishing UDP connections such as RTP multimedia streams, the Mobile Host will use its care-of address and when setting up long-lived TCP connections like telnet, ftp, irc, ..., it will use its home address and the traffic should be directed to its Home Agent. Table 5.1

illustrates an example of mobile routing table using SIP mobility support with Mobile IP. In this table,

dest. addr.	netmask	port	mobile IP	Comment
0.0.0.0	0.0.0.0	23	yes	all telnet traffic should use mobile IP
0.0.0.0	0.0.0.0	21	yes	all ftp traffic should use mobile IP
0.0.0.0	0.0.0.0	0	no	all other traffic does not use mobile IP

Table 5.1: Example of mobile routing table. Source : [17]

Chapter 6

Analysis of Application Requirements

This chapter aims at providing an application analysis of the SIP client we want to implement. We also define the environment in which the application should work and the tools needed to develop it. This work was carried out within the framework of the project SIPMOB during a four-month internship at *Ecole Nationale Supérieure des Télécommunications de Bretagne* under the supervision of Professor J.M. Bonnin.

6.1 Requirements

The SIP client application should offer these functionalities :

- Audio/Video Sessions
- Edge Presence Server
- Buddy List
- Instant Messaging (IM) System
- Mobility Support
- Operating System (OS) Independence

6.1.1 Audio/Video Sessions

The application should allow the user to establish an audio and/or a video session with another user using a SIP client. A microphone and a webcam are required for that functionality.

6.1.2 Edge Presence Server

The application should allow the user to display its status and be notified of his buddies presence information. The presence service should be compliant with the CPP specifications as described in Section 3.2 and should used PIDF for presence information in order to ensure interoperability between different SIP presence clients.

As the presence agent module of our SIP server was still under development and as the SIP `PUBLISH` method was only released as a RFC in October 2004, we assume that the proxy server is not presence enabled and therefore presence information should be processed at the client side only.

6.1.3 Buddy List

The application GUI should include a buddy list. For each buddy of the list, presence status should be displayed. The GUI should allow the user to add or remove a buddy from the list.

6.1.4 Instant Messaging System

The application should allow the user to send an instant message to one of his buddies. This functionality will be based on the pager-mode model and should be compliant with the CPIM specifications as described in Section 4.3.

6.1.5 Mobility Support

The application should allow the user to change the codec used for an audio/video session while changing network. Some codecs require fewer band-

width availability and thus are more appropriate in some networks. The application should also provide pre-call and mid-call mobility as described in Section 5.3.

6.1.6 Operating System Independence

The application should at least run on Windows and Linux/Unix platform.

6.2 Choice of a SIP client

As the primary purpose of this work was to study the SIP signaling mechanism within SIMPLE and mobility support, we decided not to start from scratch but we preferred to extend an existing application which covers the requirements the best. For that purpose, we analyzed three SIP clients : Windows Messenger 5.0 , KPhone 3 and Sip Communicator.

6.2.1 Windows Messenger 5.0

Windows Messenger is the Windows XP's instant messaging service. It allows users to establish an instant messaging, audio and video session. Windows Messenger also offers a file transferring service. The latest versions of Windows Messenger, 4.7 and 5.0, provide SIP support for audio, video and instant messaging sessions.

6.2.2 KPhone 3

KPhone is a SIP user agent for Linux which can initiate VoIP connections over the Internet. It supports presence and instant messaging, and to some extent also video calls between two hosts.

6.2.3 Sip Communicator

Sip Communicator is a pure Java SIP user agent built on top of the JAIN SIP 1.1 stack (See Appendix A) and Java Media Framework (JMF) which makes it 100% portable (tested on Windows XP and Linux, and should work under Solaris). Currently Sip Communicator supports audio/video sessions

Figure 6.1: Windows Messenger

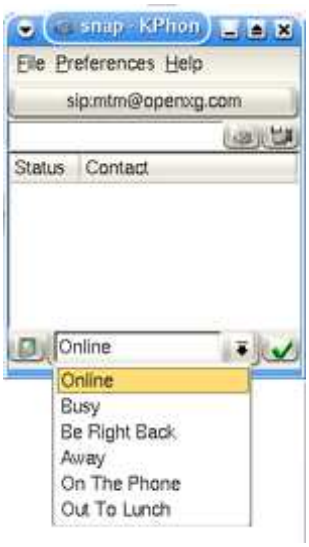

Figure 6.2: KPhone

over IPv4 and IPv6. Sip communicator is released under Apache Software License and is available at the following URL : https://sip-communicator.dev.java.net/. Sip Communicator project is lead by Emil Ivov from the Network Research Team, Louis Pasteur University, Strasbourg, France.

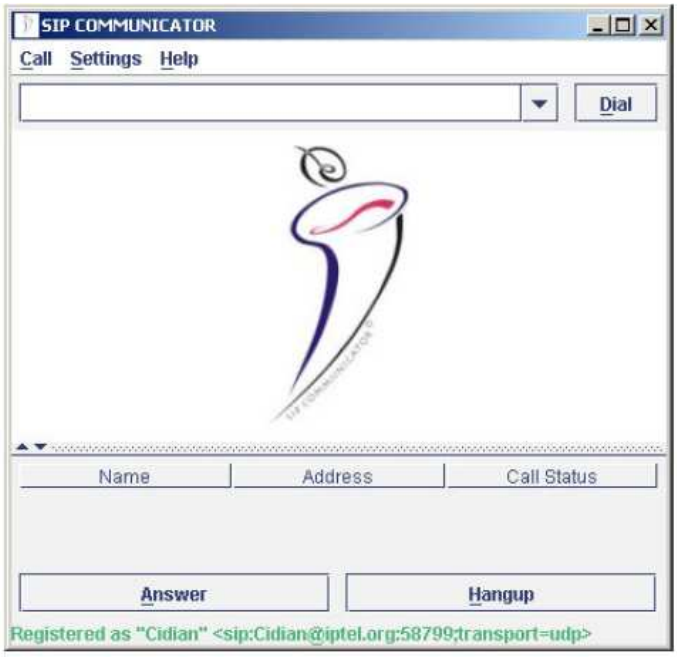

Figure 6.3: Sip Communicator

6.2.4 Comparison

Table 6.1 shows the features offered by each of the three SIP clients.

Clients	Windows Messenger	KPhone	Sip Communicator
Audio Session	Yes	Yes	Yes
Video Session	Yes	Yes	Yes
Presence	Yes	Yes	No
Instant Messaging	Yes	Yes	No
Mobility Support	No	No	No
License	Proprietary	GPL	Apache
OS	Windows	Linux	All
Programming Language	C++, .net	C++	Java
Available Support	No	Yes	Yes

Table 6.1: Comparison of Features

6.2.5 Conclusion

Windows Messenger would certainly not be the SIP client we would extend as this software is not Open Source nor OS independent. Regarding KPhone, it offers presence and instant messaging services and was released under the GPL license which are great advantages. However KPhone is especially designed for Linux systems and would not be able to run on any other OS. The third client, Sip Communicator, seems to be the best one although it does not offer presence and instant messaging services. Sip Communicator has been developed in Java and released under Apache License. We tested it under Linux and Windows and it works perfectly well. The Sip Communicator project seems to be popular and active within the Java development community as we noticed it on the mailing list which would provide us with a great support in case of troubles. We decided thus to choose Sip Communicator and add presence, instant messaging and mobility support. Windows Messenger and KPhone would be used for testing interoperability between them.

6.3 The Environment

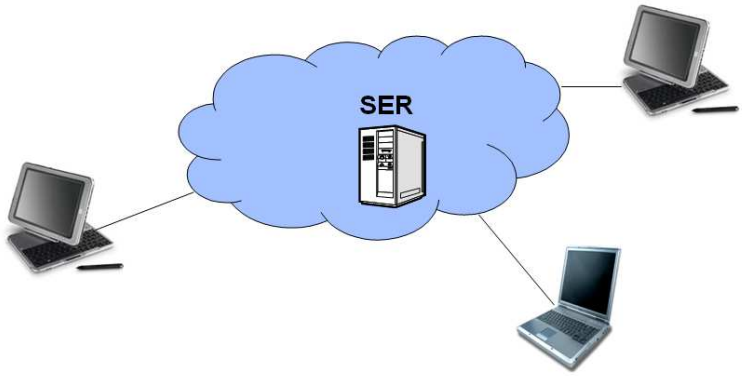

Figure 6.4: SIP Testbed

Figure 6.4 shows the environment provided by ENST Bretagne to implement and test the application. It was composed of :

1. **SIP Express Router (SER)** : Free SIP server installed under Linux 2.6 and configured to act as proxy and registrar.
2. **Pentium processor laptop under Windows XP** : Used to implement and test the application. We chose IntelliJ IDE 4.5 as Java environment for development with Java J2SE 1.4.6 and Apache Ant 1.6.2.
3. **HP Compaq Tablet PC tc1100 under Windows XP** : Runs Microsoft Windows Messenger 5.0 able to act as a SIP user agent supporting audio/video sessions, presence and instant messaging services.
4. **HP Compaq Tablet PC tc1100 under Linux 2.6** : Runs KPhone v3, a Linux SIP user agent supporting audio/video sessions, presence and instant messaging services.

The platform works over a IPv4 based network as Microsoft Windows Messenger 5.0 and KPhone v3 are not able to communicate over IPv6.

6.4 Utility Tools

6.4.1 Java 2 Platform, Standard Edition 1.4.2 (J2SE)

Java 2 Platform, Standard Edition (J2SE) provides a complete environment for applications development on desktops and servers and for deployment in embedded environments. Java is a simple and easy to use oriented-programming language when compared to the popular programming language, C++. The reason is that Java uses automatic memory allocation and garbage collection where C++ requires the programmer to allocate memory and to collect garbage. Another great advantage of Java is its portability which is required for our application. Indeed, the Java programs are compiled into Java Virtual Machine code called bytecode. The bytecode is machine independent and is able to run on any machine that has a Java interpreter especially designed for this type of platform. Java is dynamic and is able to adapt to an evolving environment. New methods and properties can be added freely in a class without affecting other classes. This will particularly concern our application as we did not start from scratch but had chosen to extend Sip Communicator which has been developed in Java. Java also offers a large number of libraries such as Swing for graphical interfaces and Java Media Framework for processing multimedia content.

6.4.2 JAIN SIP stack 1.1

SIP is an IETF (Internet Engineering Task Force) standard specification adopted by the communication industry. As a developer we are free to implement the protocol in any programming language and define our own interface for accessing the defined behavior of the protocol as outlined by the IETF standard. This standard ensures the interoperability between SIP stacks but does not guarantee the interoperability between applications based on different stacks. JAIN (Java API for Integrated Networks) SIP has been designed to satisfy this need using the Java programming language in the way, it ensures interoperability between stacks but also the interoperability of applications across stacks, referred to as application portability.

Sip Communicator has been built on the top of the JAIN SIP stack and we will keep using it to implement presence, instant messaging service as well as mobility support. A fully detailed description of the stack may be found in Appendix A.

6.4.3 IntelliJ IDEA 4.5

IntelliJ IDEA is an intelligent Java IDE intensely focused on developer productivity. It supplies an intelligent Java editor, coding assistance, advanced code automation tools and project management which enables Java programmers to boost their productivity while reducing routine time consuming tasks.

6.4.4 Apache Ant 1.6.2

Apache Ant is a Java-based build tool. It is analogous to the *Make* from the C language. Instead of writing shell commands like with *Make*, the configuration files are XML-based, calling out a target tree where various tasks get executed. Each task is run by an object that implements a particular task interface. In Sip Communicator, Apache Ant allows the user to compile source codes, generate and launch different binary packages which are ready to be executed on Windows, Linux or Solaris. It also allows to generate a Sip Communicator applet which might be used directly from a web page.

6.4.5 SIP Express Router (SER)

SIP Express Router (SER) is a high-performance, configurable, free SIP server. It can act as SIP registrar, proxy or redirect server. SER features an application-server interface, presence support, SMS gateway, SIMPLE2Jabber gateway, server status monitoring, etc. SER is written in C and ported to Linux, BSD and Solaris. It also provides support for both IPv4 and IPv6.

Chapter 7

Implementation

This chapter presents the implementation part of the work. It includes the architecture and design of the newly implemented functionalities. These functionalities should respect the requirements defined in the application analysis.

7.1 Architecture

Figure 7.1 shows Sip Communicator components and their dependencies. Components in blue are part of the original Sip Communicator while those in red have been added to implement the new functionalities.

Packages :

- `net.java.sip.communicator` :
  - `SipCommunicator` : It is the Sip Communicator's main component. `SipCommunicator` deals with `GuiManager`, `SipManager` and `MediaManager`, initializes them and manages interactions between them.

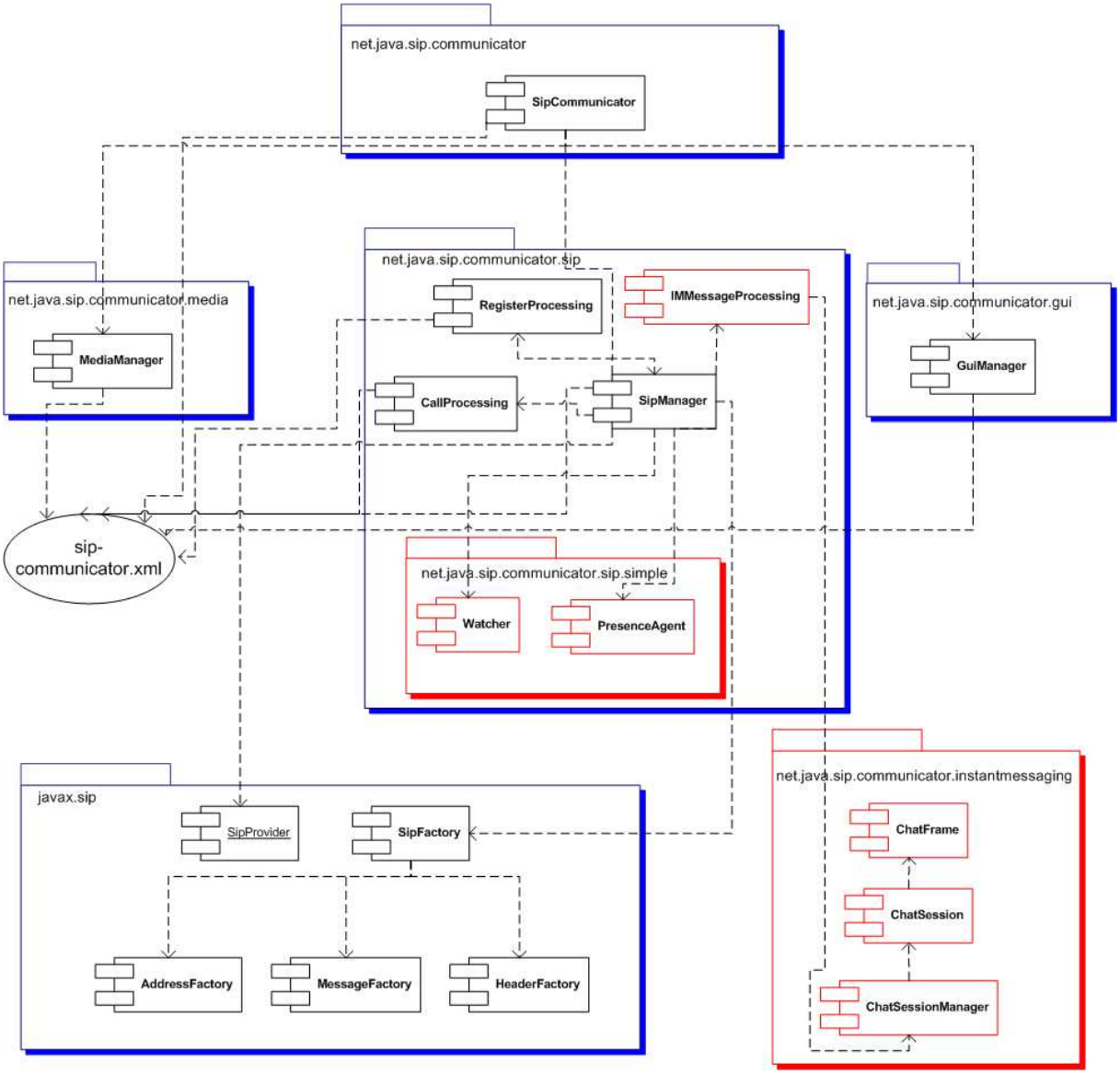

Figure 7.1: Sip Communicator Architecture

- `net.java.sip.communicator.sip` :
  - `SipManager` : This component implements `SipListener` in that, it defines the methods required to receive and process events that are emitted by the `SipProvider`.
  - `RegisterProcessing` : This component processes `REGISTER` re-

quests and responses for registration and un-registration. It ensures retransmission when `REGISTER` request time out expires.

- `CallProcessing` : This component processes call request i.e `INVITE` and `BYE`, and response i.e `180 RINGING`. It ensures users authentication before processing `INVITE` request.
- `IMMessageProcessing` : This component processes `MESSAGE` request and response which is then transmitted to the `ChatSessionManager`.

- `net.java.sip.communicator.media` :
  - `MediaManager` : This component relies on the Java Media Framework (JMF) which enables audio, video and other time-based media to be added to applications built on Java technology. `MediaManager` initializes and configures JMF processors. It also detects and initializes capture devices i.e webcams, processes audio/video flows for transmission/reception

- `net.java.sip.communicator.gui` :
  - `GuiManager` : This component relies on the Java Swing graphical package. It manages all graphical user interfaces and interactions with users. These interfaces allow users to configure the application, enter a user's URI to make calls and display video sessions.

- `net.java.sip.communicator.sip.simple` :
  - `Watcher` : This component is in charge of sending `SUBSCRIBE` request to every buddy on the list at the start of the application or when a new buddy is added. The `Watcher` component also processes `NOTIFY` requests and responses when notification is sent from SIP clients.
  - `PresenceAgent` : The `PresenceAgent` processes `SUBSCRIBE` requests coming from SIP clients and sends back `NOTIFY` request

with the current presence status. The `PresenceAgent` also sends `NOTIFY` request every time presence status is updated.

- `net.java.sip.instantmessaging` :
  - `ChatSessionManager` : This component manages all `ChatSessions`. The `ChatSessionManager` is created when launching Sip Communicator. When receiving a `MESSAGE` request, the `ChatSessionManager` forwards it to the right `ChatSession`.
  - `ChatSession` : A `ChatSession` represents a chat dialog between a user and one of its buddies. A `ChatSession` is created for every chat conversation initiated by the user himself or by one of its buddies.
  - `ChatFrame` : A `ChatFrame` is a frame where sent and received texts are displayed. The `ChatFrame` also allows the user to type his message and send it. Each `ChatFrame` is associated with one `ChatSession`.
- `javax.sip` :
  - `SipFactory` : This component contains the main interfaces that model the JAIN SIP architecture : `SipStack`, `ClientTransaction`, `ServerTransaction`, `SipProvider` and `SipListener`.
  - `MessageFactory` : This component provides factory methods that allow an application to create request and response messages.
  - `HeaderFactory` : This component provides factory methods that allow an application to create Header object.
  - `AddressFactory` : This component provides factory methods that allow an application to create Address objects and SIP URIs.
- `sip-communicator.xml` : XML file that contains Sip Communicator settings such as registrar port and IP address, preferred audio and video encoding, ports used for audio and video transmission, etc.

7.2 Application flow

Figure 7.2 displays the Sip Communicator application flow.

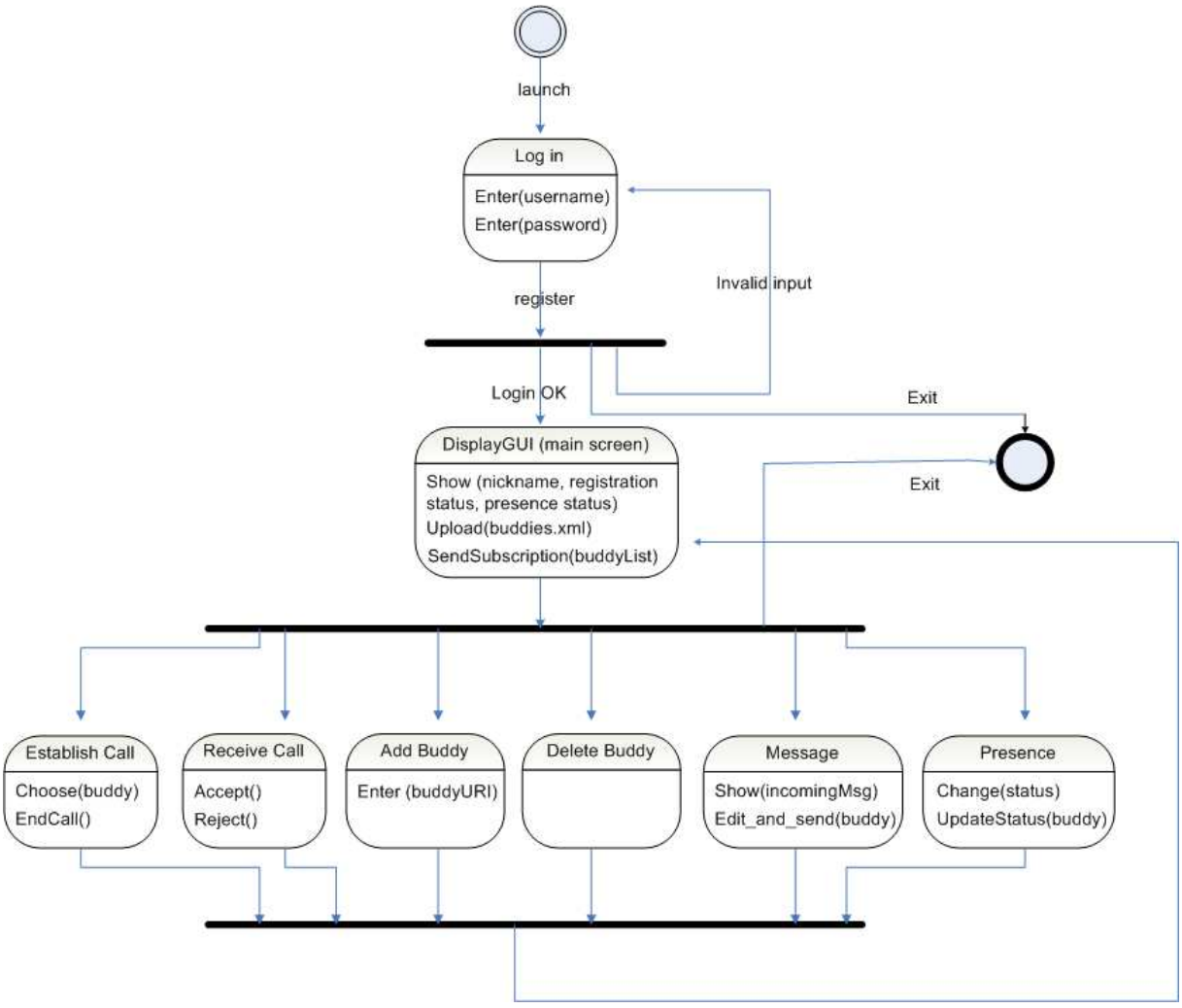

Figure 7.2: Sip Communicator Application Flow

After a successful log in, the main Sip Communicator frame will be shown to the user. The interface contains the user's buddy list which has been uploaded from a XML file. A subscription has been sent for each buddy of the list after registration to the registrar. From that interface, the user will be able to add or remove a buddy, make a call, accept or reject incoming calls

and send instant messages to his buddies. As soon as he receives an instant message, a frame is displayed with the corresponding text. The user also has the possibility to change its presence status or to exit the application at anytime. The buddy list will be updated with presence status every time a notification is sent from any buddy present on his list.

7.3 Presence

As we assume our proxy server was not presence enabled, we implemented the presence service only at the client side. We assume that each presentity has only one device and enough bandwidth. Therefore the PA and PUA are co-located in Sip Communicator which means that subscriptions and notifications are sent to and from this client directly. Such architecture is called *edge presence server*. In this model (Figure 7.3), a presentity must send subscription/notification request to every presentity/watcher.

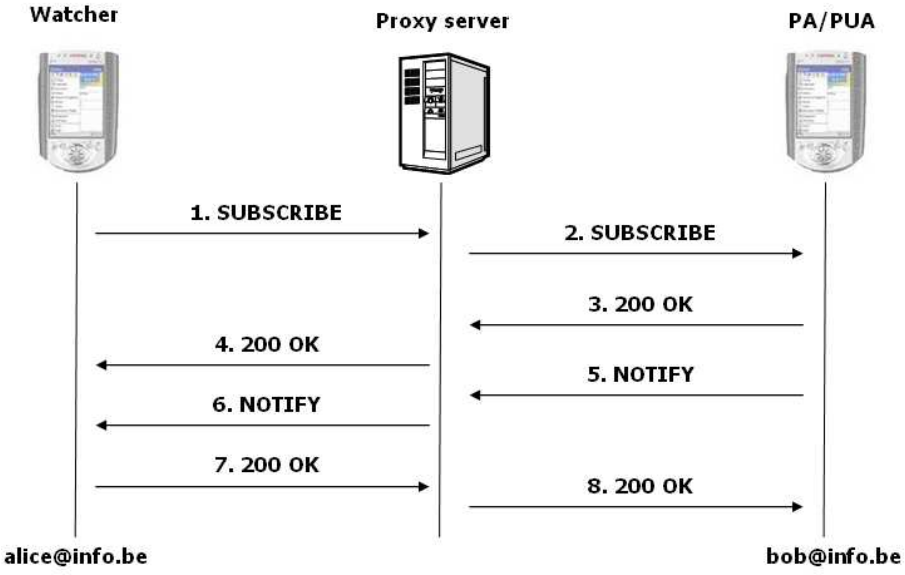

Figure 7.3: Presence End-to-End Model

7.3.1 Buddy List

A buddy list contains all the presentities that a user wants to subscribe to. The buddy list we implemented enables a user to :

- Add a presentity to the list
- Remove a presentity from the list
- Know presentities status (online, offline, away, busy, on the phone, gone to lunch)
- Initiate a call or send an instant message to a presentity by a simple click on its username on the list.

All the presentities from the buddy list are saved within `buddies.xml` when closing the application. When launching Sip Communicator, it parses this file and uploads all the presentities in the buddy list. From the mobility viewpoint, this file may be uploaded to a server when closing the application and downloaded from the server when starting it.

7.3.2 Watcher

Subscription Sending

As we are based on the `SUBSCRIBE/NOTIFY` model, our watcher is a subscriber. `SUBSCRIBE` requests are sent in two cases. In the first case, immediately after the registration process has been achieved and `buddies.xml` has been uploaded, the `Watcher` sends a `SUBSCRIBE` to every presentity present in the buddy list. In the second case, a `SUBSCRIBE` request is sent to a new presentity just after being added in the buddy list. Before adding a new contact, we check whether this presentity is present on the list.

Notification Processing

When a watcher sends a `SUBSCRIBE` request, it should expect presence information notification from the PA. Then `NOTIFY` requests are sent from the PA every time the presentity changes its presence information. The first thing a

watcher does when receiving a NOTIFY is to answer to the subscriber with a 200 OK response. Afterward, the watcher extracts the presence information contained in a PIDF document within the NOTIFY body. It parses it and takes out the relevant information in order to update presence status on the buddy list.

7.3.3 Presence Agent

Subscription Processing

Once a PA receives a SUBSCRIBE request, it is first transmitted to the SipManager. The SipManager calls the PresenceAgent to process the new request by replying with a 200 OK then stores it and sends a NOTIFY request with the presence information the watcher queried.

Notification Sending

A combo box, containing the statuses a user may choose, is displayed on the main interface. Every time the presentity changes its status, a NOTIFY request containing a PIDF body with the presence information is sent to watchers that have subscribed to. All NOTIFY requests sent to a same presentity must have the same CallID. Hence we have to store the CallID of the first NOTIFY request and use it back when future notifications are needed.

Mutual Subscription

As we assume that the server cannot support advanced features like presence, our application has to buffer subscription requests while the presentities are offline. When a Watcher sends out a SUBSCRIBE request and receives a 404 Not Found response, it temporarily stores the SUBSCRIBE request. Then, when receiving a SUBSCRIBE request, the Watcher first checks whether it is subscribed to the new subscriber. If not, it will send back the temporarily stored SUBSCRIBE request. In that way, both clients are subscribed to each other and can have access to presence information of each other. Figure 7.4 illustrates the mutual subscription.

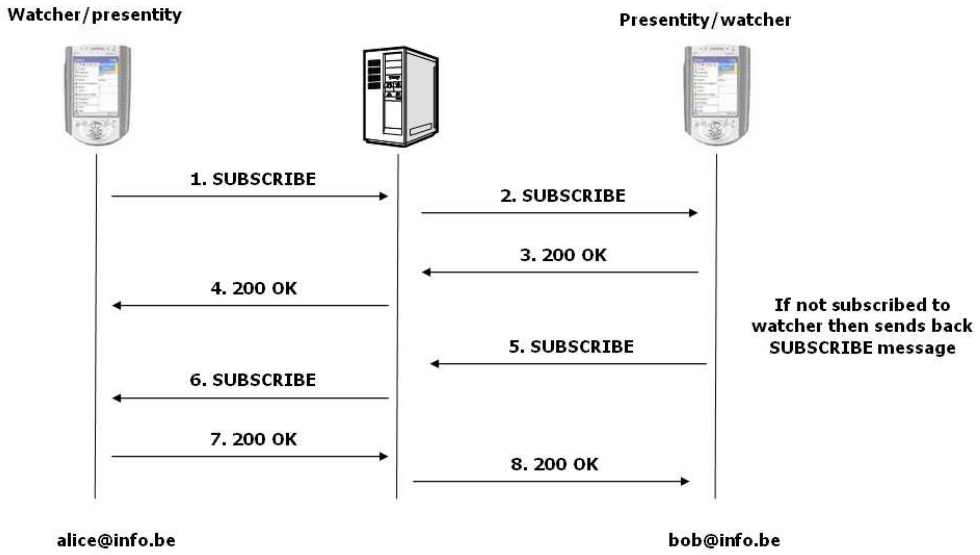

Figure 7.4: Mutual Subscription

7.4 Instant Messaging

As the Message Session Relay Protocol (MSRP) is still under development at IETF, none of the current SIP clients are multi-users chat enabled. For that reason, our instant messaging implementation is based on the pager-mode model as described in Section 4.4.1. Although `MESSAGE` requests may carry any type of MIME content, we will restrict it to text content.

7.4.1 `MESSAGE` Request Processing

After going through the JAIN SIP stack, the `MESSAGE` request will be transmitted to the `SipManager` which will invoke the `IMMessageProcessing` to process it. The `MESSAGE` request processing is composed of two parts. The first part consists in replying to the originator of the request with a `200 OK`. The second part consists in displaying the text in a `ChatFrame`. Figure 7.5 shows the activity diagram when the `ChatSessionManager` is called by the `IMMessageProcessing`. The `ChatSessionManager` checks whether there is a `ChatSession` on the run between the user and the message initiator. If it is the case, it then checks if a `ChatFrame` exists. If so, it will directly display

the message in the `ChatFrame`. If not, it will create a new `ChatFrame` that will display the new message. If a `ChatSession` does not exist, the `ChatSessionManager` will create one with a new `ChatFrame` and the message will appear inside of it. A `ChatSession` remains active even if a user closes the `ChatFrame`.

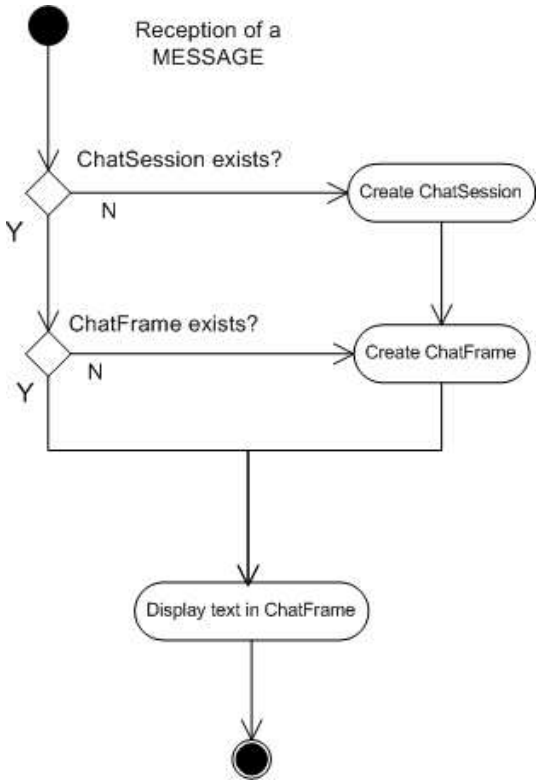

Figure 7.5: Activity Diagram : Reception of a `MESSAGE` request

7.4.2 `MESSAGE` Request Sending

The user types his message in a text area within the `ChatFrame` then clicks on the `Send` button. The message is transmitted to the `SipManager` which queries the `IMMessageProcessing`. Afterward, the `IMMessageProcessing` processes the `MESSAGE` request sending and encapsulates the text-encoded message in a SIP request by invoking JAIN SIP factories. Eventually the

MESSAGE request is sent to its destination. Once the user receives the 200 OK response from the recipient, its own message is displayed in the ChatFrame.

7.5 Mobility

7.5.1 Session Modification

While moving between subnets, change in bandwidth availability may appear. Therefore, multimedia sessions should use more appropriate audio and video encoding techniques depending on bandwidth availability. We have implemented session modification which enables a user to change audio and video encoding techniques during an ongoing session. Our implementation is based on the re-INVITE scheme as shown on Figure 7.6.

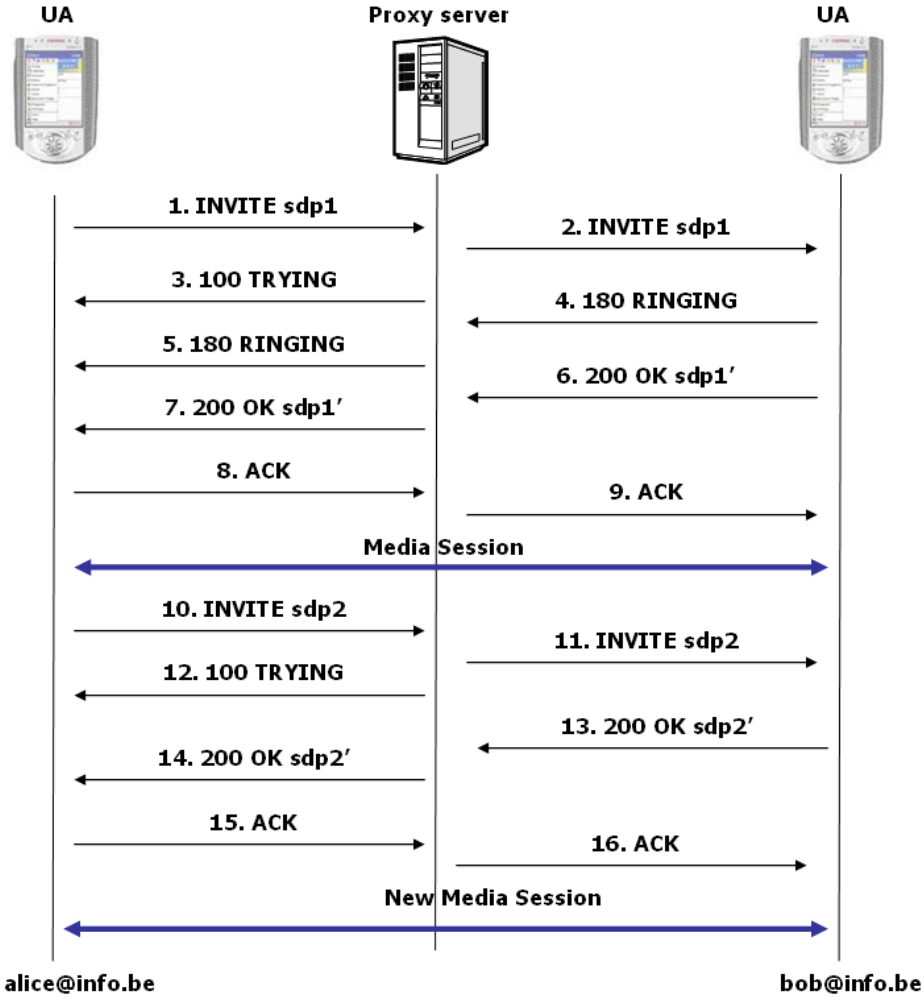

Figure 7.6: Session Modification

The first part, until the media session establishment, has been explained in Section 2.6.1. Once the media session has been set up between Bob and Alice, it can be modified with a second `INVITE` request called re-`INVITE` (10-11) with a new SDP content. The re-`INVITE` may be sent by either Bob or Alice. In our case, Alice wants to modify the media session. Bob answers with a `200 OK` response (13-14) containing its own SDP content. Alice replies with an `ACK` (15-16) and a new media session is set up based on the second `INVITE` request. We implemented a GUI frame where the user is allowed to choose among several audio and video codecs. Once the user

has confirmed its choice, a new SDP content is generated then encapsulated within an `INVITE` request and sent to the recipient. At this time, the re-`INVITE` request initiator closes the media streams, so does the recipient at the reception of the re-`INVITE` request. The JMF processors at both sides need to be initialized with the new codecs which implies the closure of both media streams. When the re-`INVITE` request initiator has sent its `ACK`, the session is re-established using the new codecs. We invoke the same methods as for a first `INVITE` request. The re-`INVITE` request differs from an `INVITE` request in that it contains a new SDP content.

7.5.2 Mobile Host Registration

When a MH leaves its home network and acquires a new IP address in a foreign network, it triggers a fresh `REGISTER` request to its home registrar. The detection of the new IP address is achieved through a thread whose job consists in comparing the initial IP address with a possible newly detected IP address every 10 ms. As soon as the thread finds out a new IP address, a new `REGISTER` request containing this address in the `Contact` field is created through the JAIN SIP stack and then sent to the home registrar. As the JAIN SIP stack is created on the basis of the IP address, the stack should be destroyed and re-built up with the new IP address. This operation is quickly achieved and takes a few ms.

7.5.3 Pre-Call Mobility

Pre-call mobility has been implemented by processing the `302 MOVED TEMPORARILY` response. When receiving a `302 MOVED TEMPORARILY` response, the Mobile Host's new IP address is extracted from the `Contact` header which is then used as `To` header in the `INVITE` request. The `INVITE` request can now be redirected to the Mobile Host.

7.5.4 Mid-Call Mobility

We implement the mid-call mobility without keeping the session ongoing and by restricting it to VoIP. Once the Mobile Host arrives in a foreign

network, the media session is disrupted as media packets sent from the Correspondent Host do not find the recipient IP address. As soon as the Mobile Host acquires a new IP address and has registered to its home registrar, it sends back a new `INVITE` request to the Correspondent Host in order to re-establish the session. The Mobile Host puts the new IP address in the `Contact` field which indicates the Correspondent Host where it wants to receive future SIP messages. The Mobile Host also indicates its new IP address in the `c=` (connection) field of the SDP content in order to redirect the media flow to its new location.

Chapter 8

Results and Discussion

In this chapter, we presents the results of our work. We will also present a critical review of the application. The last part of this chapter will take into account the future work that could be achieved to improve the application.

8.1 Results

We will present here experiments we have performed in order to test whether our newly implemented application fulfilled the requirements.

8.1.1 Presence

Scenario A.1 : Mutual subscription between Sip Communicator and Windows Messenger/KPhone

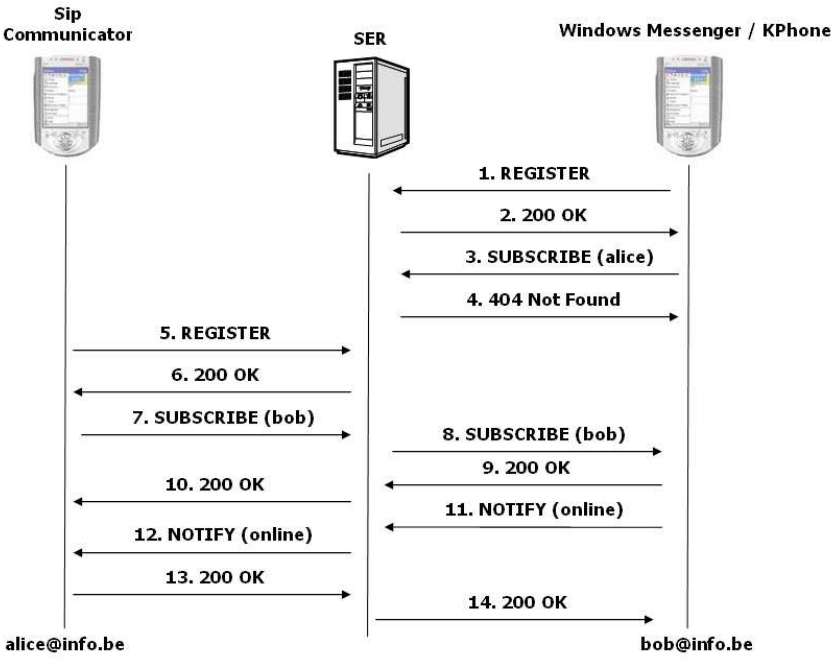

Figure 8.1: Scenario A.1

We tested here if there was a mutual subscription between our application and Windows Messenger or KPhone. As the result is the same for Windows Messenger as for KPhone, we will only mention Windows Messenger in our example. We assume here that Alice and Bob are within the same domain and that Alice is present on Bob's buddy list and the opposite is also true. As shown on Figure 8.1, we first started Windows Messenger with Bob's SIP URI (`sip:bob@info.be`). It sends a subscription request to Alice and

received a `404 Not Found` response as Alice is not registered yet. Then, Alice started Sip Communicator using her SIP URI (`sip:alice@info.be`), she sends a subscription request to Bob and immediately receives presence notification from Bob. As Bob's user agent is not taking in charge mutual subscription, Bob will never be notified of Alice's change of status as his earlier subscription has been rejected. This problem has been noticed on many SIP clients and may be solved by enabling presence feature on SIP server.

Scenario A.2 : Mutual subscription between Sip Communicator and Windows Messenger/KPhone

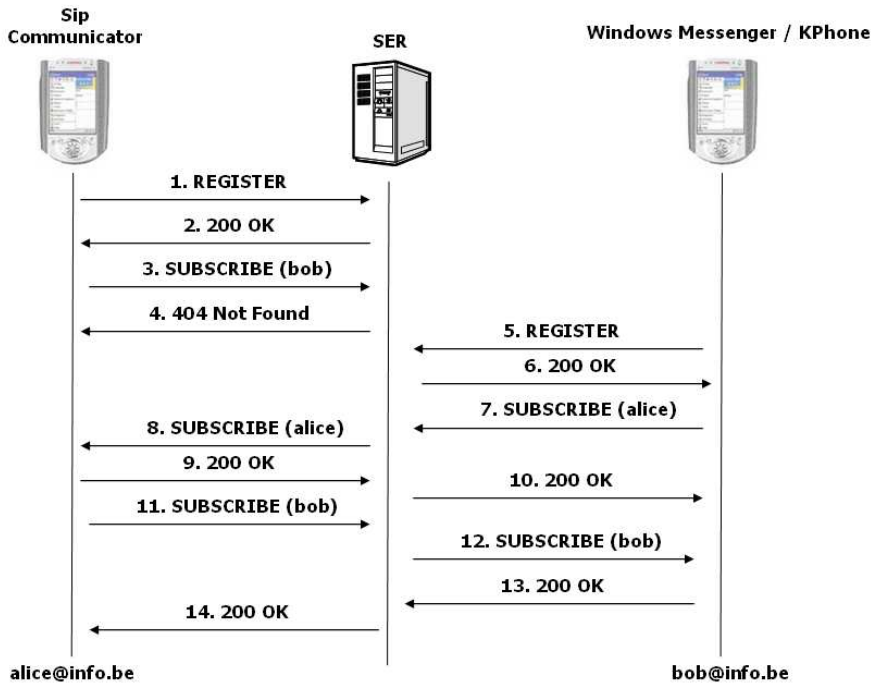

Figure 8.2: Scenario A.2

We have made the same assumptions as in Scenario A.1. The difference here is that we first launched Sip Communicator with Alice's username. On Figure 8.2, Alice sends a subscription to Bob and receives a `404 Not Found` response as Bob is `offline`. Then, Bob appears `online` and sends

a subscription request to Alice. From her side, Alice also sends back a subscription request to Bob. From that moment, both will be notified of status change of each other. We conclude that our application may process mutual subscription.

Scenario A.3 : Mutual subscription between two Sip Communicator's

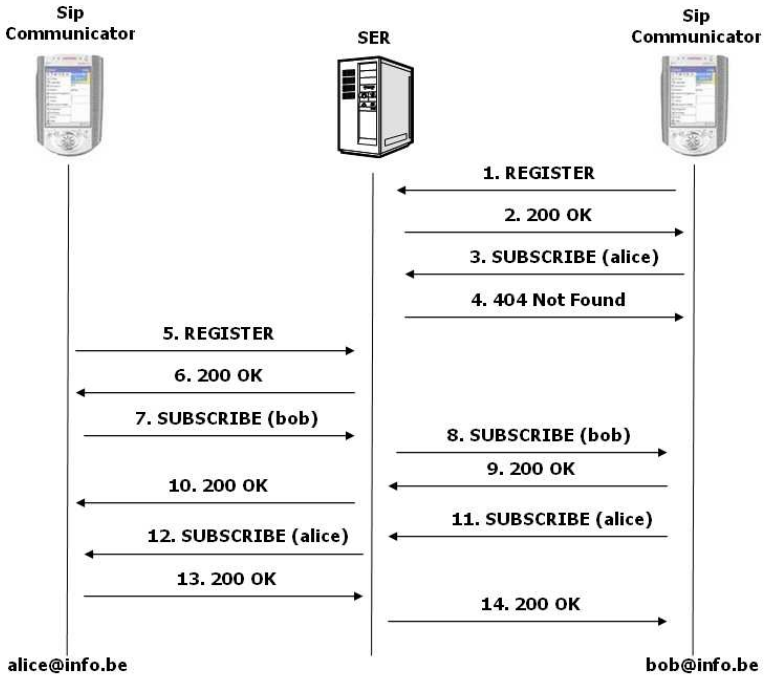

Figure 8.3: Scenario A.3

We also tested it between two Sip Communicator's and both user agents were subscribed to each other (Figure 8.3).

Scenario A.4 : Presence notification between two domains

We assume now that Alice and Bob belong respectively to domain `info.be` and `enst.fr`. They are both using Sip Communicator. In Figure 8.4, a proxy server is assigned to each domain and plays the role of outbound proxy for requests destined outside the domain. Bob notifies Alice that he

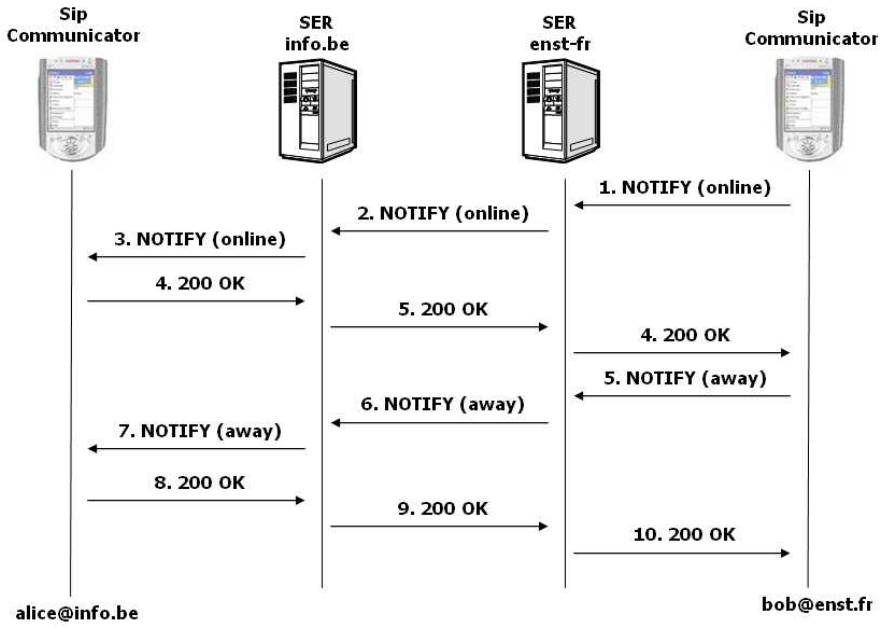

Figure 8.4: Scenario A.4

is `online`. Once his request reaches Bob's proxy, it is routed through Alice's proxy which forwards it to her. The same pattern occurs when Bob change his status to `away`.

8.1.2 Instant Messaging

Scenario B.1 : Instant Messaging between Sip Communicator and Windows Messenger/KPhone within one domain

We assume that Alice and Bob are within the same domain and are already subscribed to each other. Alice sends a `Hello` message to Bob and Bob answers `How are you?`. This is illustrated on Figure 8.5. We also tested this scenario between two Sip Communicator's and it works perfectly well.

Scenario B.2 : Instant Messaging between two domains

As in Scenario A.4, Alice and Bob belong respectively to domain `info.be` and `enst.fr` and they are both using Sip Communicator. Alice sends a `Hello` message which is first routed to her proxy. The message request is

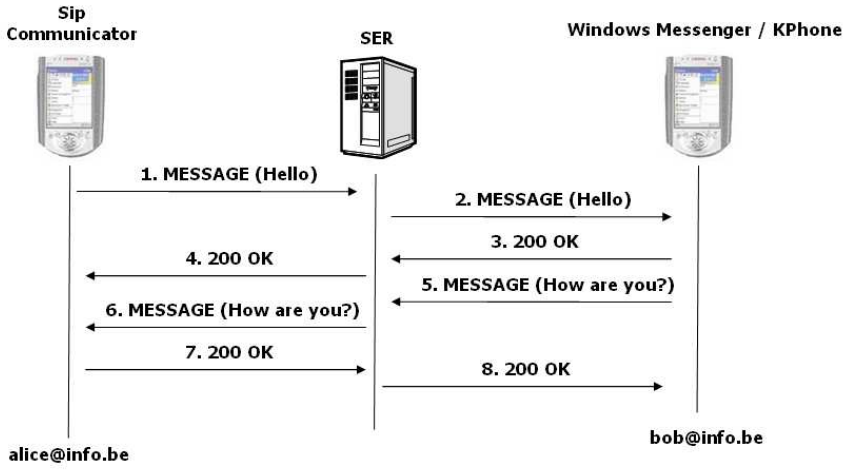

Figure 8.5: Scenario B.1

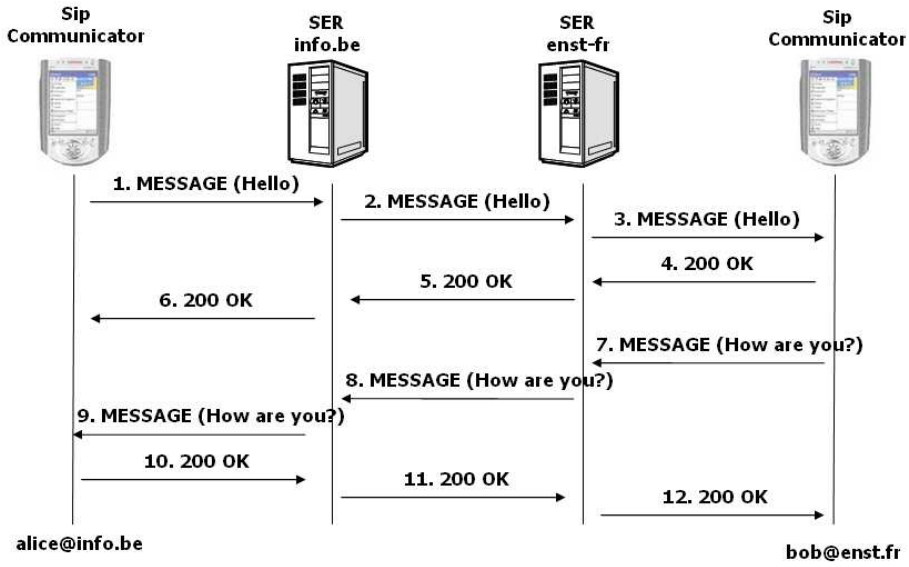

Figure 8.6: Scenario B.2

then forwarded to the Bob's proxy and is eventually directed to Bob. Bob answers `How are you` which is routed through Alice's message reverse path.

8.1.3 Mobility

Scenario C.1 : Pre-call mobility - Mobile Host moves in the same subnet

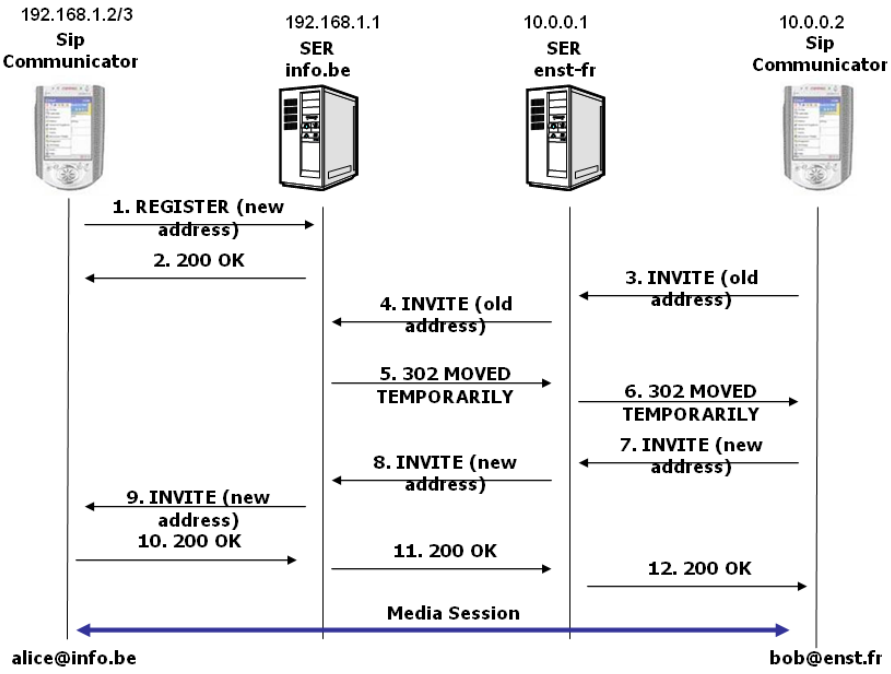

Figure 8.7: Scenario C.1

We tested here our application through a 802.11g Wi-Fi connection. We still assume that Alice and Bob belong respectively to domain `info.be` and `enst.fr` and they are both using Sip Communicator. When moving within a same subnet, a Mobile Host normally keeps its IP address (192.168.1.2). For this experiment, we have manually changed the Mobile Host's IP address to 192.168.1.3. So as shown on Figure 8.7 the Mobile Host sends a re-`REGISTER` request (1) to update the server `info.be` location database. The Correspondent Host wants to establish a session but has not been informed about the Mobile Host's change of IP address. The Correspondent Host therefore sends its `INVITE` request (3-4) to the old Mobile Host's IP address. The Mobile Host's server sends back a `302 MOVED TEMPORARILY` response (5-6) indicating the Mobile Host's new IP address. The Correspondent

Host will then redirect its `INVITE` request using the Mobile Host's current IP address (7-8-9) and then the session could be established.

Scenario C.2 : Mid-call mobility - Mobile Host moves to different subnet

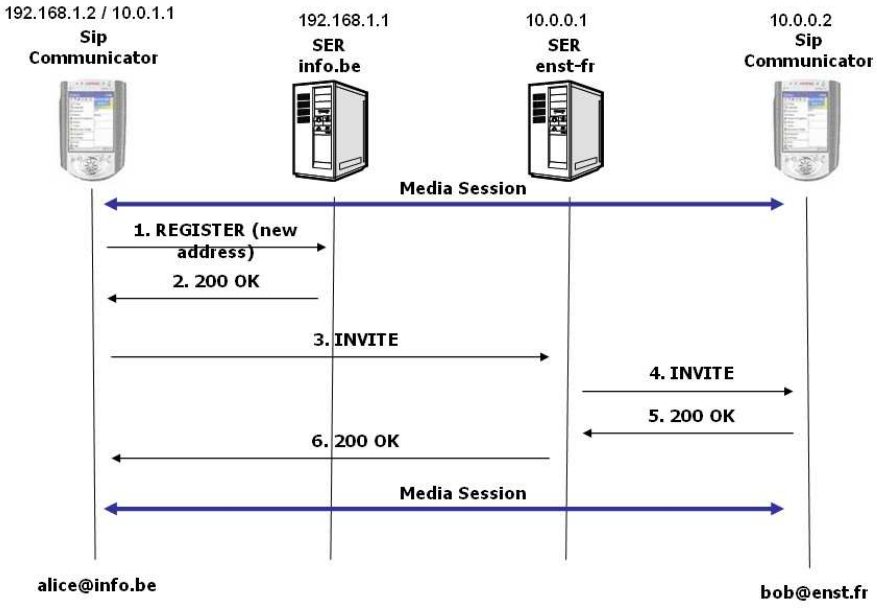

Figure 8.8: Scenario C.2

In this scenario (Figure 8.8), we assume that a media session is ongoing between the Mobile Host and the Correspondent Host. After moving to a foreign network, the Mobile Host does not receive any packets from the Correspondent Host. Therefore, the connection is disrupted. Once the Mobile Host acquires a new IP address (10.0.1.1), it sends a re-`REGISTER` request (1) to the server. It then wants to re-establish the session with the Correspondent Host. For that purpose, the Mobile Host sends an `INVITE` (3-4) request which is routed through the Correspondent Host's server then forwarded to the Correspondent Host itself. A new session is set up between the Mobile Host and the Correspondent Host.

8.2 Fulfillment of the Requirements

We will check here whether our application fulfill the requirements stated in Section 6.1.

8.2.1 Audio/Video Sessions

Sip Communicator already offers users the possibility to establish audio and video sessions with other SIP clients.

8.2.2 Edge Presence Server

Sip Communicator has been augmented with the presence function. It allows users to be notified of their buddies presence information and notify their own status. The presence function is completely CPP-compliant and fulfills SIMPLE specifications. SIP presence messages carry PIDF content to ensure interoperability. Many tests have been performed to test interoperability among several SIP clients and have been successfully passed by our application. Presence service has been completely implemented at the client side and therefore the application works without presence enabled server.

8.2.3 Buddy List

A buddy list has been added to the Sip Communicator GUI (Figure 8.9). From that buddy list, a user is allowed to see his buddies status, add or remove a buddy and initiate an audio/video session with one of his buddies.

Figure 8.9: Buddy List

8.2.4 Instant Messaging System

Sip Communicator has been enhanced with the instant messaging function (Figure 8.10). It allows users to send real-time text-based message to their buddies. The instant messaging function has been implemented following the pager-mode model and is fully compliant with CPIM and SIMPLE specifications. Our application has successfully passed interoperability tests with other SIP clients.

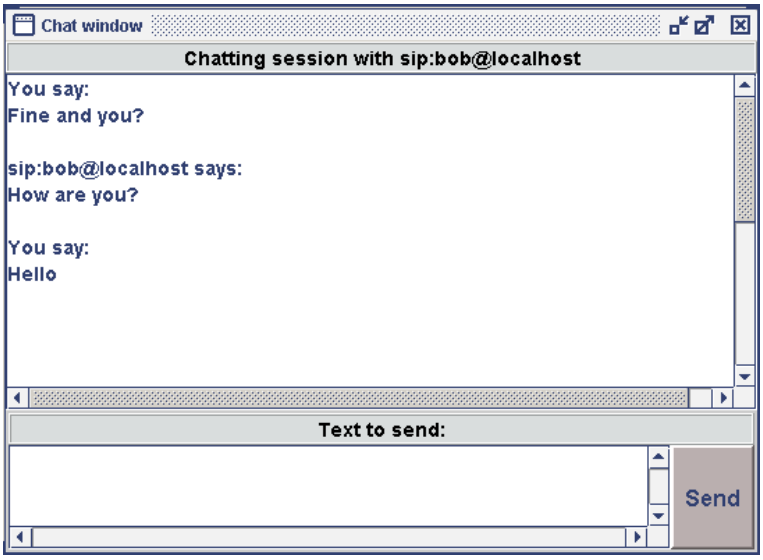

Figure 8.10: Chat Frame

8.2.5 Mobility Support

Our application allows users to change codecs during an ongoing session. This might be very useful in some networks where bandwidth is narrow. Automatic re-registration has been implemented when Sip Communicator is moving to a foreign network. Pre-call and mid-call mobility without continuous connectivity have been implemented following the solution proposed in Section 5.3. Mobility with continuous connectivity may be implemented at the server side. A solution is suggested at the end of this chapter.

8.2.6 Operating System Independence

Our application is written in Java language and therefore should run on any platforms. We tested it on Linux and Windows XP and concluded that this requirement is fulfilled.

8.3 Critical Review

8.3.1 Pushing Presence Service to the Edges

If we do not consider the fact that our SIP server is not presence enabled, what would be the advantages and drawbacks of implementing presence function at the edge? Firstly, this model would better match SIP's conception of networks as we introduced it in Chapter 2 : *"smart endpoints in dumb networks"*. Contrary to traditional telephone network, SIP offers a wider range of services than just making calls between two endpoints and that is the reason why SIP features should be implemented at the edge. Secondly, pushing presence service to the edges increases the robustness in a peer-to-peer SIP network. Indeed, a centralized SIP presence server would constitute a single point of failure and in case of crash, there would be disastrous consequences i.e. loss of all subscription requests. In case of failure of our application, this would only affect subscriptions addressed to that client and would not have any incidence on the whole network. Thirdly, due to their success, presence and instant messaging security threats are still increasing. Bypassing presence servers would allow end-to-end encryption which is the best way to preserve message confidentiality and integrity.

But pushing presence service to the edges results in serious increase in network traffic. This increase may be explained by two main reasons. The first reason is the fact that the watcher needs to generate subscription request for each buddy of his buddy list. And the second reason is that each change of presence status implies notification to each watcher. The use of a presence server (Figure 8.11) would considerably reduce the network traffic. Indeed, presence server enables a watcher to subscribe to his whole list with only

one SUBSCRIBE request and instead of sending notifications to each buddy, a presentity may just send a simple PUBLISH request.

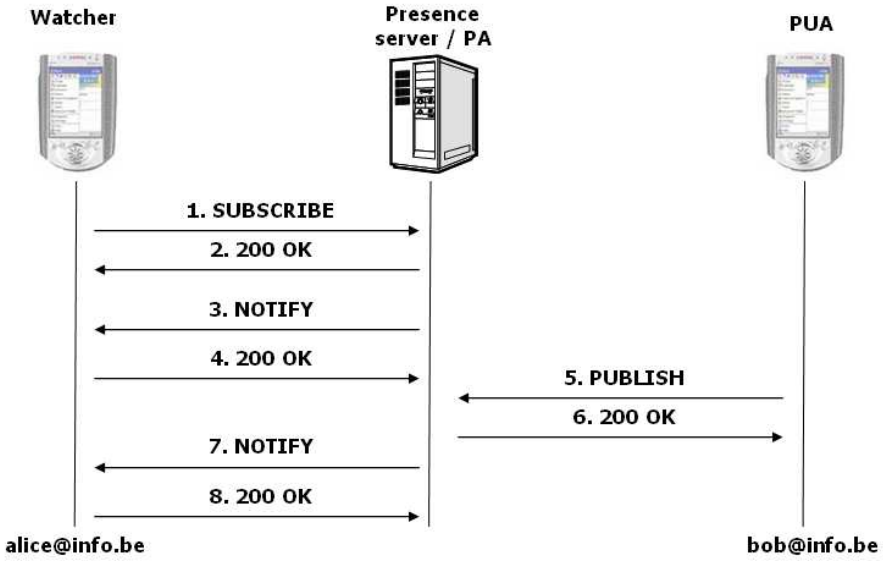

Figure 8.11: Presence Server Model

We measured the traffic load related to presence service in one hand with MSN Messenger which uses a presence server and in the other hand with our client without using presence server. The measurement has been carried out during one hour with ten buddies. The results are shown in Table 8.1. We may observe that the total traffic is about 20 times greater with our client than with MSN Messenger. This may be explained by the use of MSN presence servers and the small packet size of the MSNMS protocol.

	Protocol	Packets	Avg. Packet Size	Total Traffic
MSN Messenger	MSNMS	212	146 bytes	30950 bytes
Our SIP Client	SIP	1160	643 bytes	745770 bytes

Table 8.1: Traffic Load Comparison

A solution to this issue would be the combination of presence function at the edge with SIP signaling compression mechanism called SigComp. Following [29], if using good compressing algorithms, this could result in reduction by half of the traffic volume but would require more processing to compress and decompress at the endpoints. Another solution would be the use of a throttle mechanism for limiting the rate of notifications as specified in the Internet-Draft *SIP Event Notification Extension for Notification Throttling* [35]. This mechanism enables a subscriber to set a minimum time period between the generation of two notifications coming from a single watcher. As a result, this would imply a decrease in the number of notifications but also in the accuracy of the presence information depending on the throttling value.

Another drawback is the loss of presence and instant messaging messages while the Mobile Host is moving to a foreign network. Indeed, the Correspondent Host keep sending packets to the old location and receives a `404 NOT FOUND` response from the proxy. A solution would be mid-call mobility with continuous connectivity which is detailed in Section 8.4.2.

8.3.2 JAIN SIP Stack Destruction

In Section 7.5.2, once the Mobile Host moves to a new location, the JAIN SIP stack is destroyed then re-initialized with the new IP address. During this short period, all incoming SIP messages would be temporarily lost as all port listeners have been deleted. The Correspondent Host will keep re-sending the message until it receives a `408 REQUEST TIMEOUT` response after 30 seconds which is enough to the stack to be re-initialized and therefore the reception of the message by the Mobile Host at its new location would be possible.

8.4 Future work

8.4.1 Automatic Session Modification

Due to network congestion, periods of low bandwidth availability may appear. For instance, Alice has established a media session with Bob using PCM μ-law codec. The available bandwidth suddenly drops resulting in a poor quality call. In order to increase call quality, the session may automatically adjust to a higher compression codec, i.e GSM, without users intervention. As mentioned in Section 2.4.1, RTCP packets provide feedbacks on call quality such as packet loss and jitter. Based on these reports, our client may detect network congestion and automatically trigger re-`INVITE` request using a more suitable codec to the bandwidth availability.

8.4.2 Enhanced Mid-Call Mobility

In order to provide mid-call mobility with continuous connectivity, mobility should be handled at the server side as suggested in [16]. After moving, the Mobile Host informs the SIP registrar about its move. The Correspondent Host will still keep sending packets to the Mobile Host's old IP address which is not valid anymore. Therefore, the server must create a virtual interface with the same IP address than the Mobile Host's old one as soon as it receives a new registration from the Mobile Host. From that time, the proxy will receive all packets destined to the Mobile Host's old IP address and will forward them to the Mobile Host's new IP address. To create that interface and forward packets, a Linux tool called ipchains may be used. The SIP registrar will implement the ipchains tool through a SIP-CGI script. Once the Correspondent Host receives the `INVITE` request from the Mobile Host's new location, it will directly send media packets to the Mobile Host's new IP address. Figure 8.12 illustrates the proposed solution.

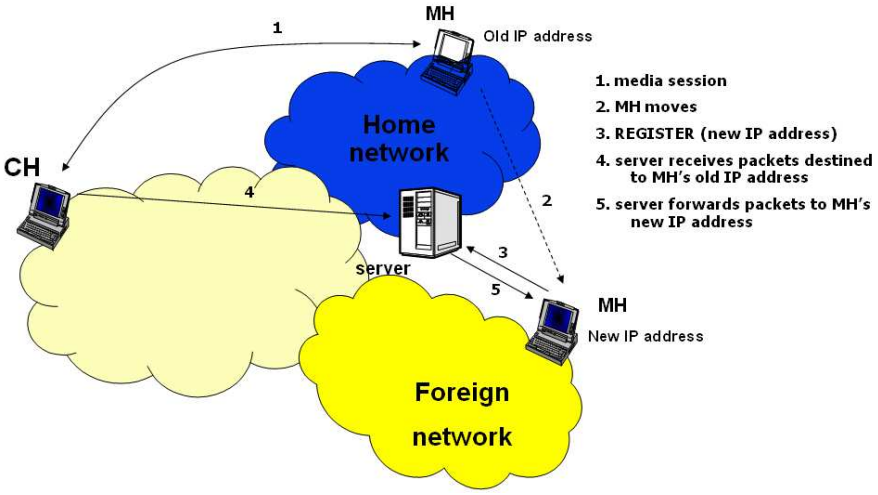

Figure 8.12: Mid-Call Mobility with Continuous Connectivity

Chapter 9

Conclusions

As of today, presence and instant messaging systems are becoming part of our daily life at home as well as at work. Each of these systems relies on its own protocol which implies interoperability issues. Several IETF working groups have been created whose goals are to design an open, interoperable and standardized protocol. We focused on one of those solutions, SIMPLE, which has been developed to extend SIP features.

Throughout this master thesis, we attempted to demonstrate the versatility and power of SIP. SIP seems to be the key to IP-based services convergence. SIP not only manages audio and video sessions but also offers presence and instant messaging function through its extension, SIMPLE. Our client has been enhanced with pre-call and mid-call mobility support without continuous connectivity. In order to complete the application, full mobility support might be implemented at the server side as described in Section 8.4.2. The application resulting from this work allowed us to better understand SIP signaling mechanism and the role it could play in the convergence issue.

Nowadays SIP has been widely spread through the world of enterprises and has been adopted by 3GPP to fulfill call control and signaling function. Many IETF working groups are still developing new functionalities to make SIP one of the most revolutionary protocols of the Internet.

Appendix A

JAIN SIP stack

In this section, we present a general overview of the JAIN SIP specifications by describing the architecture and functionalities supported and we analyze the different interactions between JAIN SIP interface, Java interfaces and SIP. These specifications are based on [37] and [38].

A.1 Introduction

SIP is an IETF (Internet Engineering Task Force) standard specification adopted by the communication industry. As a developer we are free to implement the protocol in any programming language and define our own interface for accessing the defined behavior of the protocol as outlined by the IETF standard. This standard ensures the interoperability between SIP stacks but does not guarantee the interoperability between applications based on different stacks. JAIN (Java API for Integrated Networks) SIP has been designed to satisfy this need using the Java programming language in the way, it ensures interoperability between stacks but also the interoperability of applications across stacks, referred to as application portability.

JSR 32 [37] (Java Specification Request) defines the JAIN SIP API specification. It is rich semantically and on definition to the SIP protocol. JAIN SIP offers to developers a Java-standard interface for SIP services and support for the RFC 3261 functionality and the following extensions; the INFO method (RFC 2976), Reliability of provisional responses (RFC 3262), Event Notification Framework (RFC 3265), the UPDATE method (RFC 3311), the Reason Header (RFC 3326), the Message method (RFC 3428) defined for instant messaging and the REFER method (RFC 3515).

A.2 Responsibilities of JAIN SIP

JAIN SIP aims at :

- providing methods to format SIP messages
- ensuring the ability for an application to send and receive SIP messages
- parsing incoming messages and enable application access to fields via a standardized Java interface
- invoking appropriate application handlers
- providing transaction support
- providing dialog support

A.3 JAIN SIP Object Architecture

A.3.1 SipStack Interface

This interface represents the management interface of a SIP stack implementing the JAIN SIP specification and it specifies the methods required

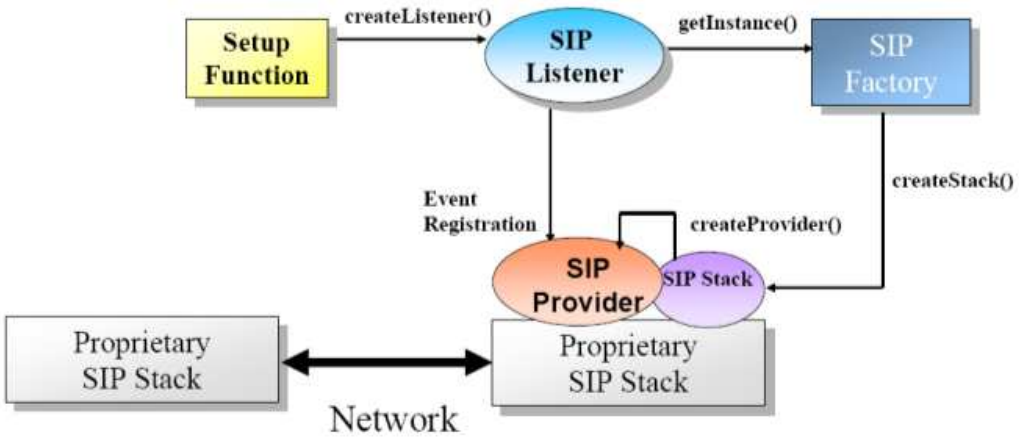

Figure A.1: JAIN SIP Architecture

to interact with a proprietary SIP protocol stack.

This `SipStack` interface defines the methods that are used by an application implementing the `SipListener` interface to control the architecture and setup of the SIP stack. These methods include :

- Creation/deletion of `SipProvider's` that represent messaging objects that can be used by an application to send request and response messages statelessly or statefully via Client and Server transactions.
- Creation/deletion of `ListeningPoint's` that represent different ports that a `SipProvider` can use to send and receive messages.

A.3.2 Architecture

A `SipStack` object is associated with a single IP address but there is a 1-N relationship between a `SipStack` and a `SipProvider`. There is a 1-N relationship between a `SipStack` and a `ListeningPoint`.

A.3.3 SipStack Creation

A `SipStack` object is instantiated by the `SipFactory` and initialized with a property set. Following the naming convention defined in `SipFactory`, the methods of the `SipStack` interface are implemented in SipStackImpl. `javax.sip.*` properties are reserved and names are defined for stack configuration properties.

A.3.4 Retransmissions

JAIN SIP provides a convenience function that ensures all retransmissions are handled by the JAIN SIP implementation which reduces the complexity for applications acting as user agents. These retransmissions can be configured via Java properties on the `SipStack` interface.

A.3.5 SipProvider Interface

This interface represents the messaging entity of a SIP stack and defines the methods that enable any application implementing the `SipListener` interface to :

- Register a `SipListener` to the `SipProvider`. Once the `SipListener` is registered with the `SipProvider` it will get notified of events representing either request, response or time out messages.
- Unregister a `SipListener` from the `SipProvider`. Once a `SipListener` is un-registered, it will no longer receive any events from that `SipProvider`.
- Provide client and server transaction creation methods.

A.3.6 Architecture

There is a N-1 relationship between a `SipProvider` and a `SipStack`, a 1-1 relationship between a `SipProvider` and a `ListeningPoint` and a N-1 relationship between a `SipProvider` and a `SipListener`.

A.3.7 SipListener Interface

This interface defines the methods required by an application to receive and process events that are emitted by an object implementing the `SipProvider` interface.

The Events accepted by a `SipListener` may be one of these three types :

- `RequestEvent` : these are request messages emitted as events by the `SipProvider`. Request events encapsulate request messages i.e. `INVITE`, that are received from the network and transmitted to the application via the underlying stack implementation.
- `ResponseEvent` : these are response messages emitted as events by the `SipProvider`. Response events encapsulate response messages i.e. 2XX's, that are received from the network to the application via the underlying stack implementation.
- `TimeoutEvent` : these are time out notifications emitted as events by the `SipProvider`. These time out events notify the application that a retransmission is required or a transaction has timed out.

A.4 JAIN SIP Messaging Architecture

This architecture follows the Listener/Provider event model, which is suitable for applications that are unsure when the next event will arrive i.e. the applications listen to it.

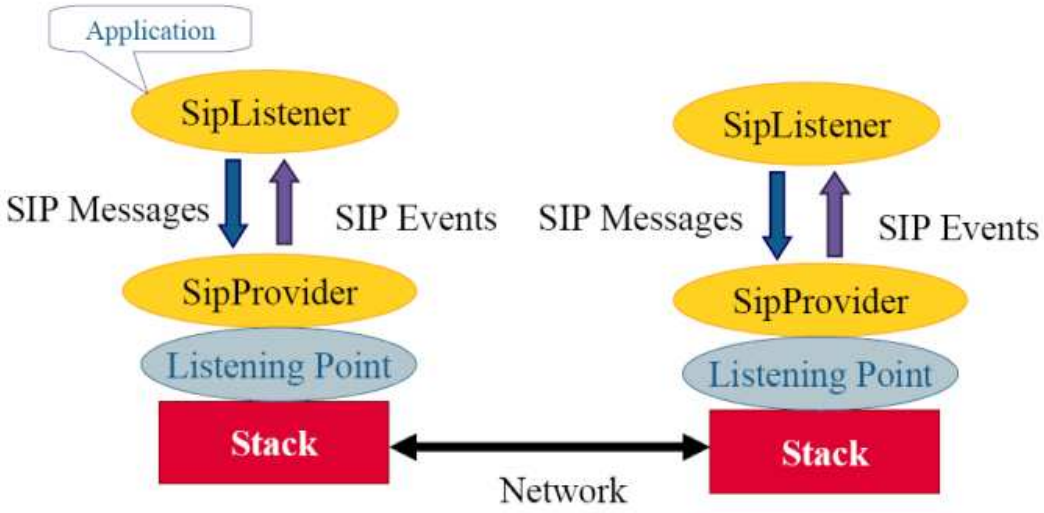

Figure A.2: JAIN SIP Messaging Architecture

A.4.1 Responsibilities of the Application

An application must register an instantiated `SipListener` object to interact with the SIP stack, it also must register with the `SipProvider` for all messaging capabilities with the stack. An application only sends out SIP messages not events. The `SipListener` listens to events containing incoming requests and responses to initiate dialogs or new incoming dialogs. The `SipProvider` receives messages from the network and transfers them to the application as events.

A.5 Packages

- `javax.sip` : Defines the main interfaces, the client and server transaction and dialog interfaces.
- `javax.sip.address` : Contains a generic URI wrapper and format SIP URIs.

- `javax.sip.header` : Contains all the supported headers and extension headers interfaces.
- `javax.sip.message` : Contains the interfaces representing SIP messages i.e. request/response messages.

A.6 Factories

JAIN SIP defines four different factories, each with respective responsibilities :

- `javax.sip.SipFactory` : This class defines methods to create new `Stack` objects and other factory objects.
- `javax.sip.address.AddressFactory` : This interface provides factory methods that allow an application to create `Address` objects and SIP URIs.
- `javax.sip.header.HeaderFactory` : This interface provides factory methods that allow an application to create Header object.
- `javax.sip.message.MessageFactory` : This interface provides factory methods that allow an application to create `Request` and `Response` messages.

A.7 Headers

`javax.sip.header` provides specific interfaces for each SIP header such as `fromHeader`, `CallIdHeader` or `toheader` as opposed to have a generic interface to handle all header information. This allows for each interface to specify the headers acceptable parameters and have more protocol support such as parsing support for each header.

A.8 Messages

JAIN SIP defines two types of messages as interfaces :

- **Request messages** which are sent from the client to the server : ACK, BYE, CANCEL, INVITE, OPTIONS, REGISTER
- **Response messages** which are sent from the server to the client in response to a request. Responses contain a status-code and a reason-phrase, as well as headers and a possible message body.

Both messages use the basic format specified in RFC 2822. The message-body may contain a Session Description Protocol (SDP) which is handled by JAIN SIP as an object. This allows the body to be a string or an object type.

A.9 Generic SIP Application Structure

A.9.1 Transaction Support

Two kinds of transactions exist within the `SipStack` implementation :

- `ClientTransaction` : A client transaction is used by a User Agent Client application to send request messages to a User Agent Server application. The client transaction is also used to match responses with previously sent requests from the User Agent Server. It also fires response events to the `SipListener` for a specific client transaction. This interfaces enables an application to send a request statefully.
- `ServerTransaction` : A server transaction is used by a User Agent Server application to send response messages to a User Agent Client application. When a request arrives, the `SipProvider` determine whether it is associated with a `ServerTransaction`. If not, the `SipProvider` creates a new `ServerTransaction`. A server transaction

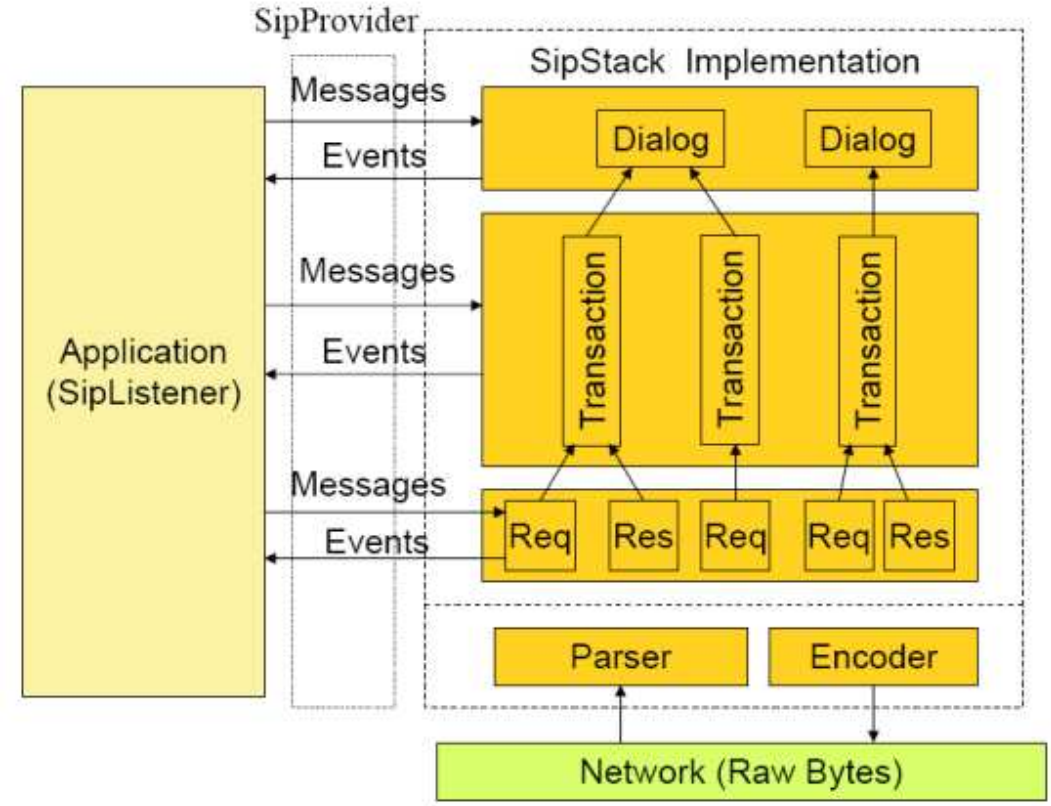

Figure A.3: Generic SIP Application Structure

also fires request events to the `SipListener` for a specific server transaction. This interfaces enables an application to send a response statefully.

A JAIN SIP transaction is a request sent by a client transaction to a server transaction, along with all responses to that request sent from the server transaction back to the client transaction.

A.9.2 Dialog Support

A dialog represents a peer-to-peer SIP relationship between two communicating SIP endpoints that persists for some time. The dialog represents a

context in which to interpret SIP messages. A dialog is used to maintain data needed for further message transmissions within the dialog such as sequence number, URIs, route sets, etc. The dialog facilitates sequencing of messages between the user agents and proper routing of requests.

Bibliography

[1] Sip vs. h.323, a business analysis. Wind River, 2001. 2.8

[2] Ed. A. Niemi. Session initiation protocol (sip) extension for event state publication (rfc 3903). The Internet Society, 2004. 3.3.3

[3] America Online (AOL). Aol instant messenger (aim) service : The world's leading instant messaging community. Available at : http://corp.aol.com/products/brands_aim.shtml, consulted in April 2005. 4.1.2

[4] Appliansys. Why instant messaging? Appliansys, 2004. 4

[5] J. Rosenberg H. Schulzrinne C. Huitema D. Gurle B. Campbell, Ed. Session initiation protocol (sip) extension for instant messaging (rfc 3428). The Internet Society, 2002. 2.5.1, 4.4, 4.4.1

[6] R. Mahy Ed. C. Jennings Ed. B. Campbell, Ed. draft-ietf-simple-message-sessions-10.txt - the message session relay protocol. The Internet Society, February 2005. 4.4, 4.4.2

[7] Editor B. Ramsdell. S/mime version 3 message specification. The Internet Society, 1999. 2.7.3

[8] Javvin Company. Sap (v1 & v2): Session announcement protocol. Available at : http://www.javvin.com/protocolSAP.html, consulted in March 2005. 2.4.4

[9] The International Engineering Consortium. Instant messaging. IEC. 4

[10] Jonathan Cumming. Sip market overview : An analysis of sip technology and the state of sip market. Data Connection, 2003. 2.2

[11] J. Arkko D. Johnson, C. Perkins. Mobility support in ipv6 (rfc 3775). The Internet Society, 2004. 5.2.3

[12] S. Donovan. The sip info method (rfc 2976). The Internet Society, 2000. 2.5.1

[13] DynamicSoft. Sip for presence. Available at : http://www.dynamicsoft.com/innovation/sip4presence.php, consulted in March 2005. 3

[14] Jabber Software Foundation. What is jabber? Available at : http://www.jabber.org/about/overview.shtml, consulted in April 2005. 3.1.3

[15] M.A. Garcia-Matin G. Camarillo. The 3g ip multimedia subsystem. 2004. (document), 3, 3.2, 3.3, 3.3.2, 4.3, 4.4, 4.4.1, 4.4.2, 4.4.2

[16] A. Dutta H. Schulzrinne, P-Y. Hsieh. Application layer mobility proxy for real-time communication. Department of Computer Science, Columbia University, 2003. 8.4.2

[17] E. Wedlund H. Schulzrinne. Mobility support using sip. Department of Computer Science, Columbia University, 1999. (document), 5.3, 5.3, 5.4, 5.5, 5.3.4, 5.1

[18] E. Wedlund H. Schulzrinne. Application-layer mobility using sip. Department of Computer Science, Columbia University, 2000. 5.1, 5.2.2, 5.3

[19] P. Kyzivat J. Rosenberg H. Schulzrinne, V. Gurbani. Work in progress : draft-ietf-simple-rpid-05 - rpid: Rich presence extensions to the presence information data format (pidf). The Internet Society, 2004. 3.3.2

[20] A.B. Johnston H. Sinnereich. *Internet Communications Using SIP*. Wiley, 2001. 3.2

[21] G. Klyne A. Bateman W. Carr J. Peterson H. Sugano, S. Fujimoto. Presence information data format (pidf) (rfc 3863). The Internet Society, 2004. 3.3.2

[22] ICQ Inc. The icq story. Available at : http://www.icq.com/info/icqstory.html, consulted in April 2005. 4.1.1

[23] Yahoo Inc. Yahoo! messenger makes the world a little smaller, more informed. Available at : http://docs.yahoo.com/docs/pr/release331.html, consulted in April 2005. 4.1.3

[24] ipUnplugged. White paper : Mobility and mobile ip, introduction. ipUnplugged, 2003. (document), 5.1, 5.2.1, 5.2, 5.2.1

[25] G. Camarillo A. Johnston J. Peterson R. Sparks M. Handley E. Schooler J. Rosenberg, H. Schulzrinne. Sip: Session initiation protocol (rfc 3261). The Internet Society, 2002. 2, 2.2, 2.3.1, 2.5.1, 2.7.2, 2.7.3, 2.7.4

[26] K.W. Ross J.F. Kurose. *Computer Networking, A Top-Down Approach Featuring The Internet*. Addison-Wesley, 2003. 2.4, 2.4.1, 2.4.1, 2.4.2, 2.4.3

[27] G. Mohr J. Vincent M. Day, S. Aggarwal. Instant messaging / presence protocol requirements (rfc 2779). The Internet Society, 2004. 3.1.1, 3.1.3

[28] H. Sugano M. Day, J. Rosenberg. A model for presence and instant messaging (rfc 2778). The Internet Society, 2000. 3.2, 4.3, 4.3

[29] Z. Sogor G. Sey M. Fidrich, V. Bilicki. Sip compression. 2003. 8.3.1

[30] V. Jacobson M. Handley. Sdp : Session description protocol (rfc 2327). The Internet Society, 1998. 2.6.1

[31] K. Kiss M. Lonnfors. Work in progress : draft-ietf-simple-prescaps-ext-00 - user agent capability presence status extension. The Internet Society, 2004. 3.3.2

[32] mess.be. Happy birthday msn messenger!!! (archive of the week of august 01 2004). Available at : http://www.mess.be, consulted in April 2005. 4.1.4

[33] Cathleen Moore. Xmpp vs simple: The race for messaging standards. Available at : http://www.computerworld.com.au/index.php?id=940058663&fp=16&fpid=0, May 2003, consulted in April 2005. 3.1.3

[34] C. Politis R. Tafazolli N. Akhtar, M. Georgiades. Sip-based end system mobility solution for all-ip infrastructures. Centre for Communication Systems Research (CCSR), University of Surrey, UK, 2003. 5.2

[35] A. Niemi. Work in progress : draft-niemi-sipping-event-throttle-03 - sip event notification extension for notification throttling. The Internet Society, 2005. 8.3.1

[36] M. Sjöstedt O. Bergquist. Ip telephony, a swedish perspective. Master's thesis, Kungl Tekniska Högskolan, Vetenskap Och Konst, 2003. 2.4

[37] M. Ranganathan P. O'doherty. Jain sip api specification - jsr 32. Available at : Seehttp://jcp.org/en/jsr/detail?id=32, 2002. A, A.1

[38] M. Ranganathan P. O'doherty. Slides : Jain sip tutorial, serving the developer community. Sun Microsystems, NIST, 2002. A

[39] Ed. P. Saint-Andre. Extensible messaging and presence protocol (xmpp) : Core (rfc 3920). The Internet Society, 2000. 3.1.3

[40] C. E. Perkins. Mobile networking through mobile ip. IEEE Internet Computing, 1998. 5.2.1, 5.2.1

[41] A. B. Roach. Session initiation protocol (sip)-specific event notification (rfc 3265). The Internet Society, 2002. 2.5.1, 3.1.2, 3.3.3

[42] J. Rosenberg. The session initiation protocol (sip) update method (rfc 3311). The Internet Society, 2002. 2.5.1

[43] J. Rosenberg. A presence event package for the session initiation protocol (sip) (rfc 3856). The Internet Society, 2004. 3, 3.3

[44] P. Salin. Mobile instant messaging systems - a comparative study and implementation. Master's thesis, Helsinki University of Technology, 2004. 3.1

[45] L. Schumacher. Slides from the course of "reseaux matières approfondies" : Chapter08, mobility in all-ip networks. Facultés Universitaires Notre Dame de la Paix (FUNDP), Namur, Belgium, 2004. 5.2.2

[46] R. Sparks. The session initiation protocol (sip) refer method (rfc 3515). The Internet Society, 2003. 2.5.1