Appendix A

SIP SIGNALING PROTOCOL

The signaling protocol (SIP), the reference architecture and protocol characteristic procedures will be analyzed and described in this chapter. In the second part, the evolution of SIP will be studied, in particular the extensions that support new services for third generation networks.

Protocol Introduction

SIP, Session Initiation Protocol, is an application-layer control protocol that can establish, modify and terminate multimedia sessions (conference, Internet video telephony, etc) o just phone calls.
The SIP protocol has the following characteristics:

- Client / Server
- Technologies tied to the Internet world, particularly to HTTP; it was not conceived with the goal of guaranteeing inter work with PSTN networks even if this is possible.
- Decentralized call control of intelligent central terminals.

The first SIP standard has been defined in RFC (Request for Comments) 2543, and it is under continuous revision; SIP is used together with other protocols, such as:

- RSVP (Resource Reservation Protocol), for keeping network resources;
- RPT (Real-time Transfer Protocol) and RTCP (Real-time Transfer Control Protocol), for transporting real-time data and for providing service quality feedback;
- RTSP (Real-time Streaming Protocol), for controlling delivery of streaming media;
- SAP (Session Announcement Protocol), for spreading multicast communications;
- SDP (Session Description Protocol), for describing multimedia sessions.

The exchanged messages contains most of the information content, thus reducing the protocol implementation complexity, and it guarantees significant network scalability. The Client/Server intelligent components of a SIP network remain free of particular tasks, and offer the following opportunities:

- Unicast and multicast support services;
- Information exchange about terminal capacity (through SDP);
- Redirect services;
- Call forwarding;

- Concerned terminals kind selection and dealing;
- Mobility support;
- Call transfer;
- Third party registration;
- Third party set up;
- Programmable services by the client.

The multimedia communication installation, as defined by the SIP protocol, requires 5 steps:

- *Client location*: which end-system has to participate in the communication;
- *Client capacity* parameters and means to adopt to communicate;
- *Client availability* called availability to communicate;
- *Call setup*: ringing, and parameters allocation from called and calling;
- *Session administration*: it includes call transfer and ending.

Sip permits also the introduction of extensions to signaling messages and changes to the protocol, acting as header, parameters and methods. It permits a continuous update, depending on the importance of supporting new functionalities.

Reference SIP Architecture

Two abstractions identify entities that characterize a session: User Agent Client (UAC) and User Agent Server (UAS).

The UAC is a client application, which generates signaling messages and sends requests; the UAS is an application server to which the requests arrive and generates or conveys responses to another server.

An applicative program can act as a client or as a server according to necessity; for example, an application that controls a multimedia conference can act as UAC to begin calls and to invite other users to participate, and as UAS to accept invitations.

The procedures defined by the SIP protocol to manage multimedia sessions can involve the following members:

- TERMINALS;
- LOCATION SERVER;
- REGISTRAR SERVER;
- PROXY SERVER;
- REDIRECT SERVER.

Terminals

They are intelligent endpoints, which support real-time and bi-directional communications between same level entities.

The terminals in SIP architecture can send, receive and elaborate information about the session in which they want to participate.

Location Server

This type of server supplies information to the Proxy and Redirect Server about the called user location.

Such information can be obtained through consultation of databases that contain previous registrations, or using mobility protocols.

Register Server

It accepts REGISTER requests, is located inside a Proxy or a Redirect Server and it offers location services.

Proxy Server

A Proxy Server is an intermediate element between the calling terminal and the called terminal acting at the same time as client and as server (UAC and UAS).

The incoming requests are worked out locally (sending a message to the receiver terminal) or to an other server able to do it; this happens through message translation and formatting .

There are two types of Proxy Server: stateful and stateless.

A stateful server has in memory the incoming and exiting requests to allow a session state control.

A stateless proxy cancels all information in and out when a request is generated.

Redirect Server

The proceedings used by a Redirect Server to guarantee the exchange of messages between two endpoints are different from the ones in a Proxy.

When it contacts a client, the server gathers a list of alternative locations from the Location Services and returns a final response of 3xx class.

Messages

The SIP protocol uses the same text messages both for signaling management and for negotiation between two endpoints, exchanging codified parameters as foreseen by SDP.

As Sip is similar to the HTTP protocol, it has the same semantics and syntax, and the possibility of transporting with such messages information that permit terminals to communicate to set up a session.

Sip gets from HTTP header fields, that is information about sender, addressee, type of code and capacity.

A SIP message-body has:

- Start line, which can be a request or a response line;
- One or more header fields;
- Empty line, end of field header;
- Message body.

The SIP messages can be requests (from client to server) and responses (from server to client).

Request-URI

The Request-URI is a SIP URL (Uniform Resource Locator) or a URI (Uniform Resource Identifier), it indicates the user or the service requested.

A SIP address can set up as user@host, where user can be the user name or a telephone number, and host is the domain or network address.

Examples;

- `sip:j.my@provider.com;`
- `sip:gustavo@100.7.5.3;`
- `sip:d.my:secrt@provider.com;transport:tcp;`
- `sip:+39-02-1234-5678:333@gateway.com:user=phone.`

Proxy and Redirect Server can use information kept in the field to adapt the request and eventually to write the Request-URI.

Requests

SIP requests are distinguished by having a Request-line that contains a METHOD name, a Request-URI, and a SIP protocol.

The following methods are supported by client and server:

- INVITE;
- ACK;
- OPTIONS;
- BYE;
- CANCEL;
- REGISTER;

- INFO.

- **INVITE**: This method indicates the desired communications means (audio, video, games), parameters of those means, and addresses for receiving media from the answerer. The answer indicates which communications means are accepted, the parameters that apply to those means, and addresses for receiving media from the offerer. The offer/answer model defines restrictions on when offers and answers can be made.

- **ACK**: An ACK request is sent to a client only as an assent to a previous INVITE. This method is used by a UAC when the answer is positive, and by a proxy for other types of responses (3xx, 4xx, 5xx, 6xx). The decision of a proxy as to how to redirect such request depends on the fields From, To, CSeq and Call ID; if such header values in the request are constructed in the same way, ACK is redirect to a proxy, otherwise it must be accepted. The ACK body can contain a session description, and if the field is empty the call must consider valid the information gotten previously with INVITE.

- **OPTIONS**: This method allows the calling client entity to ask information about the behavior of a particular called user; the entity server returns capacity and methods supported with a particular header field (allow). This method permits the obtaining of information on the called state, and the INVITE acceptance. Proxy and redirect server must forward the request keeping out of the loop of further messaging.

- **BYE:** This method informs the server that the session must terminate; it is forwarded like an INVITE, and can be forwarded by the called and the caller.

- **CANCEL:** This method is used to cancel a previous request sent by a client. The following procedures are used to construct a CANCEL request: Call-id, To, CSeq, From, and must be identical to those in the request being cancelled. When the CANCEL is generated, the UAS will not generate any response. A proxy must cancel any pending transaction, while a redirect answers with a 200 (OK) response if the operation is pending, or with a 481 (Transaction Does Not Exist) otherwise.

- **REGISTER:** A client uses this method to register on a Registration Server a user identified by the address in the field To; such address can be the address of the same client or a third-party registration address. The registration can be done sending a REGISTER to a multicast servers address "sip.mcast.net" (244.0.1.75), so that the request remains in this particular area. The User Agents can listen to find out the other users' position, obviously without forwarding any answer to the requests that are sent. The requests received by the Register Server are elaborated second priority, and a client who wants to register many times must wait for the previous requests to be answered. The expiration of a request is defined by the field Expires inside the response sent to the client, a registration should be cancelled by the client with a "0" value.

- **INFO**: The SIP protocol defines used messages during a setup procedure or a call stop, but it does not define the transport mechanism of a particular session. This gap filled in with the INFO method, which is not used to change the status of a call, or a parameter, or to allow the exchange of optional information along a signaling route. The INFO method can transport:

- ¬ Signaling messages PSTN between gateway PSTN during a session;
- ¬ DTMF figures generated by a call;
- ¬ Information about wireless signals power;
- ¬ Contracts data;
- ¬ Non-streaming images and information.

The signaling route for the INFO method is established during setup; it can be direct between calling and called, or through a proxy server. The INFO server can make an answer:

- ¬ Positive: If the request is received with success for an existing call (200 OK).
- ¬ Negative:
  - ♣ If the request is referred to an unknown call (481 Call Leg/Transaction Does Not Exist);
  - ♣ If the server cannot understand the request body (415 Unsupported Media Type Message).

If the server receives a CANCEL for an INFO previously requested, it behaves as if the request had never been received (487 Request Cancelled).

Response

This a SIP message sent from a server to a client, it indicates the status of a request sent from a client to a server. Any response allows six values:

- INFORMATIONAL 1XX;
- SUCCESS 2XX;
- REDIRECTION 3XX;
- REQUEST FAILURE 4XX;
- SERVER FAILURE 5XX;
- GLOBAL FAILURE 6XX.

- INFORMATIONAL 1xx: The server contacts the called client, the necessary time is superior to 200 ms, the response is not definitive, the client waits for more information. Examples:
  - ¬ 100 Trying, server looking for user;
  - ¬ 180 Ringing, user located, waiting for an answer;
  - ¬ 182 Queued, called busy with another call, request queued.

- **SUCCESS 2xx**: The action was successfully received, understood and accepted, and a 200 OK message was sent by the server.

- **REDIRECTION 3xx:** Further action needs to be taken in order to complete the request. Examples:
  - ¬ 300 Multiple Choices, the address in the request found several choices, the user can select a preferred communication endpoint, the choices are listed as Contact fields;
  - ¬ 301 Moved Permanently, the user can no longer be found at the address in the request; the requester must retry at a new address given by the Contact header field;
  - ¬ 302 Moved Temporarily, the requesting client should retry the request at a new address given by the Contact header field;
  - ¬ 380 Alternative Service, the call was not successful, but alternative services are possible. The alternative services are described in the response message.

- **REQUEST FAILURE 4xx:** Request failure due to the client, a new request must be sent with modification. Causes of failure can be the following:
  - ¬ 400 Bad Request, the request could not be understood due to wrong syntax;
  - ¬ 401 Unauthorized, the request requires user authentication;
  - ¬ 404 Not Found, the user does not exist at the domain specified in the Request;
  - ¬ 485 Ambiguous, the request was ambiguous, the response contains a listing of possible unambiguous addresses in the Contact header field.

- **SERVER FAILURE 5xx:** The server itself has erred. Examples:
  - ¬ 500 Server Internal Error, the server encountered an unexpected condition that prevented it from fulfilling the request;
  - ¬ 501 Not Implemented, the server does not support the function required to fulfill the request.

- **GLOBAL FAILURES 6xx:** The server has definitive information about a particular user, not just the particular instance indicated in the request. Example:
  - ¬ 604 Does Not Exist Anywhere, the server has authoritative information that the user indicated in the request does not exist anywhere;
  - ¬ 606 Not Acceptable, the user's agent was contacted successfully but some aspects of the session description such as the requested media, bandwidth, or addressing style were not acceptable.

Session Description Protocol

The SDP, Session Description Protocol, is the protocol adopted to describe multimedia sessions during the various procedures of a starting session. The characteristic parameters of a multimedia session are contained in a SIP body message. For this reason the SIP header Content Type takes the application/sdp values.

The transported information can be:

- Flow type (audio, video,…);
- Transport Protocol (RTP/UDP/IP, H.320, …);
- Transmitted data format (H.261 video, MPEG video, …);
- Session name and goal;
- Time duration;
- Available band use;
- Multicast Address and Transport Port;
- Remote Address and Transport Port.

Procedures Examples

Presentation of and comment on some characteristic SIP procedures.

Registration

Registration example:

```
C◊  S:     REGISTER  sip: gustavo.com SIP/2.0
           Via: SIP/2.0/UDP  guffy.gustavo.com
           From: sip: mark@gustavo.com
           To: sip: mark@gustavo.com
           Call-ID: 70710@guffy.gustavo.com
           CSeq: 1 REGISTER
           Contact: <sip: mark@guffy.gustavo.com: 3890; transport=udp>
           Expires: 7200
S◊  C:     200 OK
```

The user mark@gustavo.com over host guffy.gustavo.com is registered via multicast on the server gustavo.com, and it waits for an answer at port 3890. The registration lasts two hours (7200 seconds), and until that moment, all INVITE for mark@gustavo.com arriving to sip.gustavo.com will be sent to mark@guffy.gustavo.com.

The party requesting registration appears in From field, the party who is registered appears in To; in case of a third-party registration, the From and To addresses are different.

Call Setup

Two people dialog INVITE procedure:

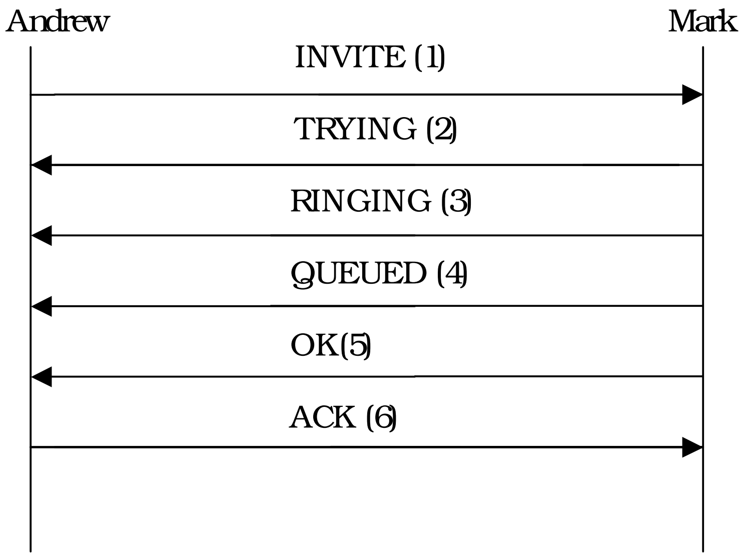

Fig. A.1 Invite procedure example

C◊ S:

```
INVITE sip: mark@guffy.gustavo.com SIP/2.0
Via: SIP/2.0/UDP  my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip: mark@gustavo.com
Call-ID: 666777888@my.gustavo.com
CSeq: 1 INVITE
Contact: <sip:Andrew@my.gustavo.com>
Subject: mark, hallo!
Content-Type: application/sdp
Content-Length: 187

v = 0
o = Andrew 53655765 2353687637 IN IP4 128.3.4.5
s = mark, hallo!
t = 3149328600 0
c = IN IP4 my.gustavo.com
m = audio 3456 RTP/AVP 0 3 4 5
a = rtpmap: 0 PCMU/8000
a = rtpmap: 3 GSM/8000
```

Andrew (Andrew@ gustavo.com) calls Mark (mark@ gustavo.com), specifying that he can receive audio flows RTP with code PCMU (0), GSM (3), G.723 (4), and DVI4 (5).

The Via field defines the request route, the Call-ID, the Content-Type (SDP information), and the Content-Length.

The empty line separates the request line and the body.

(2) S◊ C:

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip: mark@gustavo.com ;tag=37462311
Call-ID: 666777888@my.gustavo.com
CSeq: 1 INVITE
Content-Length: 0
```

(3) S◊

```
C:SIP/2.0 180 RINGING
Via: SIP/2.0/UDP my.gustavo.com
From sip: Andrew@gustavo.com
To: sip: mark@gustavo.com ;tag=37462311
Call-ID: 666777888@my.gustavo.com
CSeq: 1 INVITE
Content-0Length: 0
```

(4) S◊ C:

```
SIP/2.0 182 Queued, there is previous user
Via: SIP/2.0/UDP my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip: mark@gustavo.com ;tag=37462311
Call-ID: 333444555@gustavo.com
CSeq: 1 INVITE
Content-Length: 0
```

Reception call is 100 Trying, then the message 180 Ringing (the phone is ringing), then 182 Queued, ID changes from 666777888 to 333444555.

(5) S◊ C:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip:mark@gustavo.com ;tag=37462311
Call-ID: 333444555@my.gustavo.com
CSeq: 1 INVITE
Contact: sip: mark@guffy.gustavo.com
Content-Type: application/sdp
Content-Length: 156

v = 0
o = mark 4858949 4858949 IN IP4 192. 1.2.3
s = yes, here I am!
t = 3149329600 0
c = IN IP4 guffy.gustavo.com
m = audio 5004 RTP/AVP 0 3
a = rtpmap: 0 PCMU/8000
a = rtpmap: 3 GSM/8000
```

(6) C◊ S:

```
ACK sip: mark@guffy.gustavo.com  SIP/2.0
```

```
Via: SIP/2.0/UDP my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip: mark@gustavo.com ;tag=37462311
Call-ID: 333444555@my.gustavo.com
CSeq: 1 ACK
```

Call was accepted through 200 OK, the session parameters are returned to the body message. The called user can receive only PCMU and GSM. The INVITE procedure is terminated with ACK.

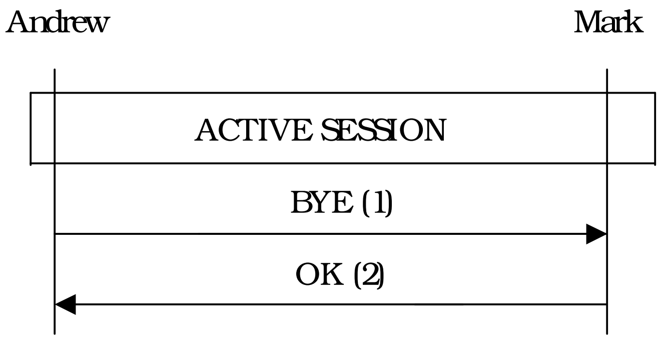

Fig. A.2 Example of session ending

C◊ S:

```
BYE sip: mark@guffy.bell-tel.com SIP/2.0
Via: SIP/2.0/UDP  my.gustavo.com
From: sip: Andrew@gustavo.com
To: sip: mark@gustavo.com ;tag=37462311
Call-ID: 333444555@my.gustavo.com
CSeq: 2 BYE
```

S◊ C:

```
200 OK
```

Bye terminates the call.

Proxy USE

This is an example of how a Proxy Server works. User 1 sends an INVITE request to User2. The address is user2@ gustavo.com; Proxy gets information about the location of user2 through Location Server (server1.david.com). Proxy sends the INVITE request to the address user2@ server1.gustavo.com.

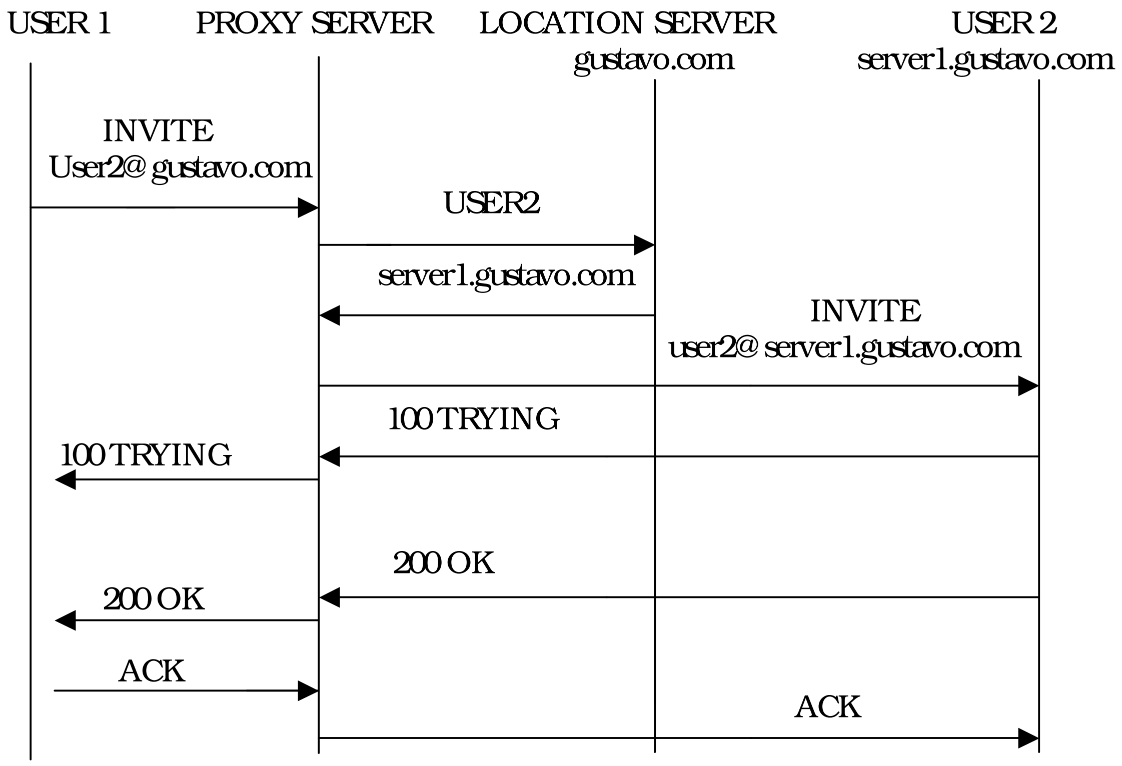

Fig. A.3 INVITE with Proxy

Redirect Server Behavior

Redirect Server behavior during an INVITE procedure

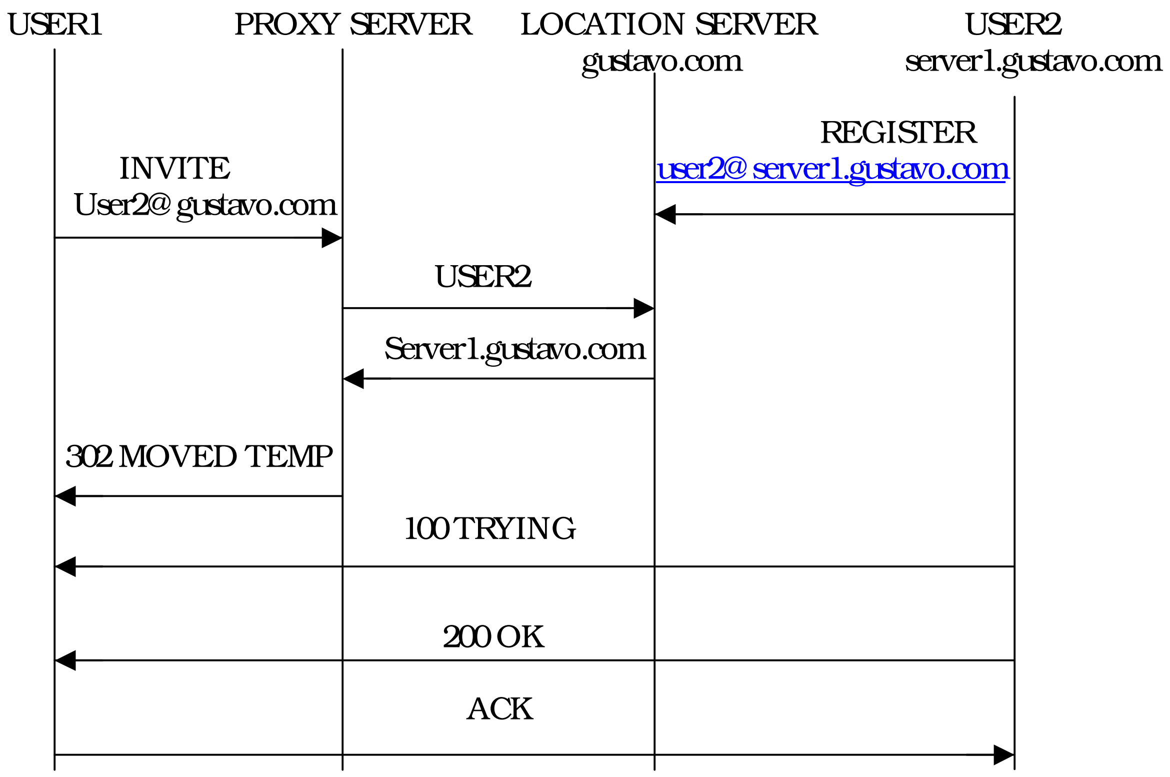

Fig. A.4 INVITE with redirect

User 1 sends an INVITE request, Redirect contacts Location Server. User 2 is already registered at Location Server. Redirect inserts the address of User 2 at 302 Moved Temporarily and passes it to User 1, who sends the request to the new address.

Appendix B

SIP APIs

The APIs used on the PDA in order to manage and process SIP were provided by Siemens Mobile. The architecture of this SIP Stack resembles very much that standardized in JSR32, but it is not a true JAIN SIP implementation. The main differences from the JAIN SIP APIs are:

¬ No support of the JAIN interface;
¬ No support of the SipFactory;
¬ Limited support of SIP headers concerning the SIP parser (but full read/write access to any SIP header);
¬ Support to only UDP as transport protocol.

The reasons that at the time of development brought to this choice, as reported from the developer, are:

¬ The focus of JSR32 is Java 2, and not PersonalJava or J2ME;
¬ Necessity to keep the stack footprint as small as possible, since the limited environment it runs on (PDA).

The choice, within the project, to stick to this stack was forced by above reasons (the only existing SIP Lite release, NIST SIP Lite, is still under testing and development) and by the decision of building further services on the existing IMS client.

In the following is given a description of JAIN SIP, since the differences with the stack used in the project are limited.

JAIN SIP

The JAIN SIP specification was the first SIP specification standardized through the JCP (Java Certification Process). The JAIN SIP specification is a general purpose transaction based Java interface to the SIP protocol. It is rich both semantically and in definition to the SIP protocol. The motivation behind JAIN SIP is to develop a standard interface to the SIP protocol that can be used independently or by higher level programming entities and environments. JAIN SIP can be used in multiple ways:

- As a specification for the J2SE platform that enables the development of stand alone user agent, proxy and registrar applications;
- A base SIP implementation for a SIP Servlet container that enables the development of user agent, proxy and registrar applications in a Servlet based environment;
- A base SIP implementation for an Enterprise JavaBeans™ (EJB™) container that enables the development of user agent, proxy or registrar applications in an EJB environment.

JAIN SIP provides a standardized interface that can be used by communications developers as a minimum to support SIP in their applications. The JAIN SIP reference implementation provides a fully functional SIP implementation that can be used by developers to talk SIP from the Java environment. The target developer community for JAIN SIP is developers that are familiar with the SIP protocol and require transactional control over the SIP implementation.

- **Architecture Overview:** JAIN SIP supports RFC 3261 functionality and the following SIP 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) and the Message method (RFC 3428) defined for instant messaging. JAIN SIP standardizes the interface to the generic transactional model defined by the SIP protocol, providing access to dialog functionality from the transaction interface. The architecture is developed for the J2SE environment therefore is event based utilizing the Listener/Provider event model. The specification is asynchronous in nature using transactional identifiers to correlate messages. It defines various factory classes for creating Request and Response messages and SIP headers. JAIN SIP defines an interface for each Header supported, which can be added to Request or Response messages respectively. These messages are passed to the SipProvider with a transaction to be sent onto the network, while the SipListener listens for incoming Events that encapsulate messages that may be responses to initiated dialogs or new incoming dialogs. JAIN SIP is extensible by design. It defines a generic extension header interface that can be used by applications that utilize headers that are not supported directly by JAIN SIP. The design also defines a mechanism to support future dialog creation methods in a JAIN SIP environment via the use of Java Properties. JAIN SIP can be managed statically with regards to IP addresses and router function, and dynamically specific to ports and transports. The default handling of message retransmissions in JAIN SIP is dependent on the application. Stateful proxy applications need not be concerned with retransmissions as these are handled by JAIN SIP. Typically User Agent applications must handle retransmissions of ACK's and 2xx Responses, however JAIN SIP provides a convenience function that ensures all retransmissions are handled by the JAIN SIP implementation, reducing the complexity for applications acting as user agents. This function may also be useful if JAIN SIP is used as a base implementation for a SIP Servlet container or an EJB implementation.

SIP Lite

The SIP Lite specification is an abstracted view of the SIP protocol that provides a SIP programming environment for developers who are not SIP literate. The API specification isprimarily developed for the J2SE platform, however as the specification is quite small it can also be implemented on the J2ME platform. The motivation behind SIP Lite on the J2ME platform is to provide a rich object model that may be suitable for midsize devices with more processing power and memory than mobile handsets, i.e. PDA's and SIP phones. SIP Lite is most common as a J2SE platform based user agent or a programming interface to a SIP Phone.

- **Architecture Overview:** SIP Lite supports the functionality defined in RFC 3261. SIP Lite can be implemented both on the J2SE platform and the J2ME platform, therefore it does not mandate the Listener/Provider event model like JAIN SIP; however, SIP Lite utilizes the Listener/Provider naming convention. SIP Lite defines a three-tier architecture, where the concept of a Listener exists for a Dialog, a Call and a CallProvider. These three interfaces in essence listen for incoming messages, dialogs and calls respectively. SIP Lite does not define specific Request and Response interfaces; rather, they are combined using a single Message interface. Messages are identified based on Request and Response constants defined within the API specification. Messages are created from and sent via a specific Dialog, where a Response is created based on a specific Request message. A Request is created specific to a method, content type and content body. A generic interface is also defined for SIPHeaders, containing generic SIP values and parameters. The SIP Lite architecture defines the concept of a Call and Dialog interface within which a Call may contain multiple Dialogs. There is no model of transactions within a Dialog; transactions are handled in the underlying implementation hidden from the application developer. SIP Lite defines a single factory class which is used to create addresses, users, headers and parameters, while a CallProvider may be viewed as a factory class for Calls and a Call may be viewed as a factoryclass for Dialogs. SIP Lite is an API specification designed specifically for User Agent applications. It does not define or support any proxying capabilities and will always behave statefully. All retransmission semantics will be handled by the implementation, further simplifying the programming environment for the application developer.

JAIN SIP and SIP Lite

JAIN SIP and SIP Lite are overlapping technologies as they are primarily designed for the J2SE platform; however they are distinctly different in programming model and scope. SIP Lite is focused solely on User Agent functionality, while JAIN SIP also supports proxy capabilities. SIP Lite defines a call centric architecture, while JAIN SIP defines a transaction centric architecture which provides more powerful control over the protocol. SIP Lite has an abstract simple look and feel i.e. SIP Lite has no concept of a Request or Response or specific SIP Headers. SIP Lite may be easier to understand compared to JAIN SIP for enterprise application developers, however for SIP application developers SIP Lite may not provide the required control over the protocol. JAIN SIP targets SIP application developers that have a reasonable understanding of the SIP protocol, i.e. message components, headers and the transaction model. SIP Lite on the contrary is focused on enabling communication capabilities to non telecom developers, such as enterprise developers with minimal SIP knowledge.

APPENDIX C

Multimedia

More than mere signalling and transmission of static contents, one of the targets of the project was that of achieving real-time streaming communications. A tool able to capture, process, transmit and present multimedia was necessary. At the moment the solutions that are proposed by the java environment are two, MMAPI (Mobile Multimedia API) and JMF (Java Media Framework) packages.

The JMF package is very well known by developers, since its use on Windows platforms (9x, NT), Solaris and Linux. This package provides the developer with very powerful tools to process multimedia and it supports, since version 2.0, presentation and transmission of RTP (Real-Time Transmission Protocol) packets. The drawback of this solution is that, being born for the world of PCs and workstations, many features available on the PC are not supported on a PDA. On a PDA, downloading the cross-platform release of this package, it is possible to use the player and RTP packets presentation. The great strength of this solution is that it is modular, it is possible to program and add platform dependent plugins and so it is very well extensible.

MMAPI libraries were born for the CDLC/MIDP platform, they are so part of a smartphone oriented solution. Their advantages are their performance (they are targeted for limited devices) and the many similarities with the JMF package, in fact they are often refered at as JMF Lite. Their limitations are no possibility to capture voice/video or to transmit RTP.

The solution deployed in the project is called SMF (Simple Media Framework) and is an extension of the JMF APIs for the PDA. This extension has been developed by Siemens in its Munich labs. More details are given on the JMF package and on SMF's extensions in the following chapters.

JMF

The Java Media Framework (JMF) is a large and versatile API used to process time-based media. Currently at version 2.2, it is Sun' s initiative to bring timebased media processing to Java. Time-based media is data that changes meaningfully with respect to time, such as audio and video clips, MIDI sequences, and animations. JMF has been built so:

- To be easily used by developers;
- To permit the development of conferencing and streaming applications;
- To support captured multimedia data;
- To let developers extend and improve its functionalities.

Among other uses, JMF can:

- Play various multimedia files in a Java applet or application. The formats supported include AU, AVI, MIDI, MPEG, QuickTime, and WAV;

- Play streaming media from the Internet, capture audio and video with your microphone and video camera, then store the data in a supported format;
- Process time-based media and change the content-type format;
- Transmit audio and video in real-time on the Internet;
- Broadcast live radio or television programs.

To be able to write a JMF application, it is needed tofully understand the JMF architecture, its interfaces, and its classes.

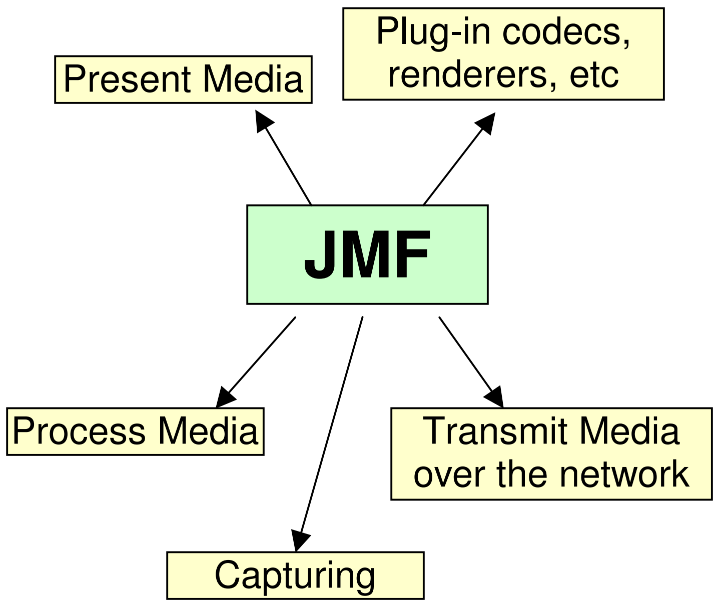
Present Media
Plug-in codecs, renderers, etc
JMF
Process Media
Transmit Media over the network
Capturing

Fig. C.1 JMF Functionality

- **JMF architecture:** the JMF is built on the following elements:

  ¬ **Data Source:** a data source encapsulates the media stream much like a music CD. In JMF, a DataSource object represents the audio media, video media, or a combination of the two. A DataSource can be a file or an incoming stream from the Internet. The good thing about this class is, once you determine its location or protocol, the DataSource encapsulates both the media location, and the protocol and software used to deliver the media. Once created, a DataSource can be fed into a Player to be rendered, with the Player unconcerned about where the DataSource originated or what was its original form. Media data can be obtained from various sources, such as local or network files, or live Internet broadcasts. As such, DataSources can be classified according to how a data transfer initiates:

    ♣ **Pull data source:** the client initiates the data transfer and controls the data flow from the source. HTTP and FILE serve as examples of established protocols for this type of data;

- ♣ **Push data source:** the server initiates the data transfer and controls the data flow from a push data source. Push data source examples include broadcast media and video on demand.

¬ **Capture Device:** a capture device represents the hardware you use to capture data, such as a microphone, a still camera, or a video camera. Captured media data can be fed into a Player to be rendered, processed to convert the data into another format, or stored for future use. Capture devices can be categorized as either push or pull sources. With a pull source, the user controls when to capture an image. An example is a still camera where a user clicks a button to take the shot. In contrast, a microphone acts as a push source because it continuously provides a stream of audio data.

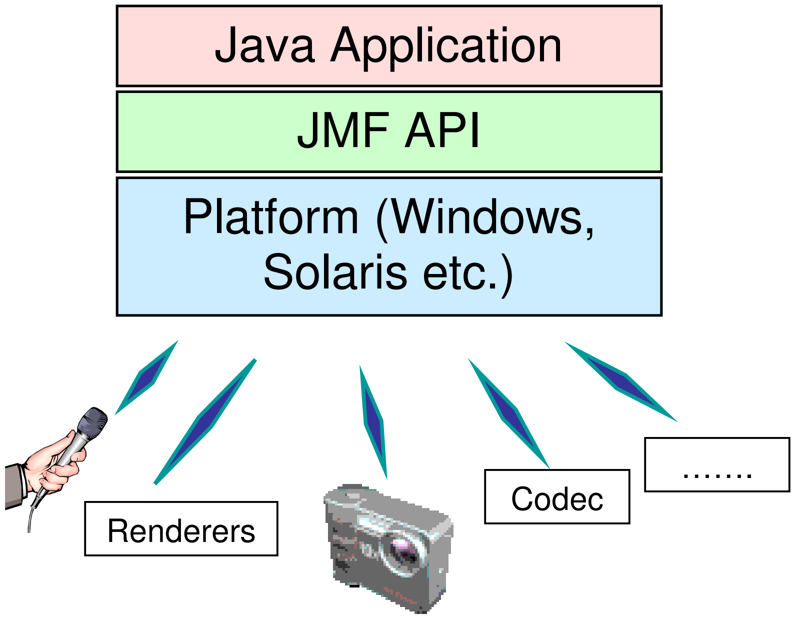

Fig. C.2 JMF Architecture

¬ **Player:** a Player takes as input a stream of audio or video data and renders it to a speaker or a screen. A Player can have states, which exist naturally because a Player has to prepare itself and its data source before it can start playing the media. In normal operations, a Player steps through each state until it reaches the final state. JMF defines six states in a Player:

- ♣ **Unrealized:** in this state, the Player object has been instantiated. A newly instantiated Player does not yet know anything about its media.
- ♣ **Realizing:** a Player moves from the unrealized state to the realizing state when you call the Player' s realize() method. In the realizing state, the Player is in the process of determining its resource requirements. A realizing Player often downloads assets over the network.

- ♣ **Realized:** transitioning from the realizing state, the Player comes into the realized state. In this state the Player knows what resources it needs and has information about the type of media it is to present. It can also provide visual components and controls, and its connections to other objects in the system are in place.
- ♣ **Prefetching:** when the prefetch() method is called, a Player moves from the realized state into the prefetching state. A prefetching Player is preparing to present its media. During this phase, the Player preloads its media data, obtains exclusive-use resources, and does whatever else is needed to play the media data.
- ♣ **Prefetched:** the state where the Player has finished prefetching media data, it' s ready to start.
- ♣ **Started:** this state is entered when the start() method I called. The Player is now ready to present the media data.

¬ **Processor:** a Processor is a type of Player. In the JMF API, a Processor interface extends Player. As such, a Processor supports the same presentation controls as a Player. Unlike a Player, though, a Processor has control over what processing is performed on the input media stream. In addition to rendering a data source, a Processor can also output media data through a DataSource so it can be presented by another Player or Processor, further processed by another Processor, or converted to some other format. Besides the six aforementioned Player states, a Processor includes two additional states that occur before the Processor enters the realizing state but after the unrealized state:

- ♣ **Configuring:** a Processor enters the configuring state from the unrealized state when the configure() method is called. A Processor exists in the configuring state when it connects to the DataSource, demultiplexes the input stream, and accesses information about the format of the input data.
- ♣ **Configured:** from the configuring state, a Processor moves into the configured state when it is connected to the DataSource and the data format has been determined.

As with a Player, a Processor transitions to the realized state when the realize() method is called.

¬ **DataSink: t**he DataSink is a base interface for objects that read media content delivered by a DataSource and render the media to some destination. As an example DataSink, a file-writer object that stores the media in a file may be considered.

¬ **Format: a** Format object represents an object' s exact media format. The format itself carries no encoding-specific parameters or global-timing information; it describes the format' s encoding name and the type of data the format requires. Format subclasses include AudioFormat and VideoFormat.

¬ **Manager:** Managers are used to create Players, Processors, DataSources, and DataSinks. For example, to render a DataSource, its possible to use Manager to create a Player for it.

SMF

The Simple Media Framework has been developed by Siemens Mobile with the Technical University of St. Petersburg and is a JMF based environment. The motivations that have led to the development of SMF come from two directions:

- Necessity on the user's side to be able to work with multimedia data on a PDA, careless of about the underlying platform;
- Necessity on the developer's side to:
    - ¬ Have access to a common multimedia application framework available on any common OS platform;
    - ¬ Be able to develop multimedia applications in an easy and comfortable way;
    - ¬ Write an application that runs on any Java-enabled device.

The above reason have brought to the birth of the SMF, its primary uses are in Conferencing Applications (PDA to PDA) and Real-Time Streaming Applications (Server to PDA), optimized for WinCE devices.
The use of the SMF is the same as the JMF to the developer, since Java acts as a wrapper of the platform dependent DLLs.

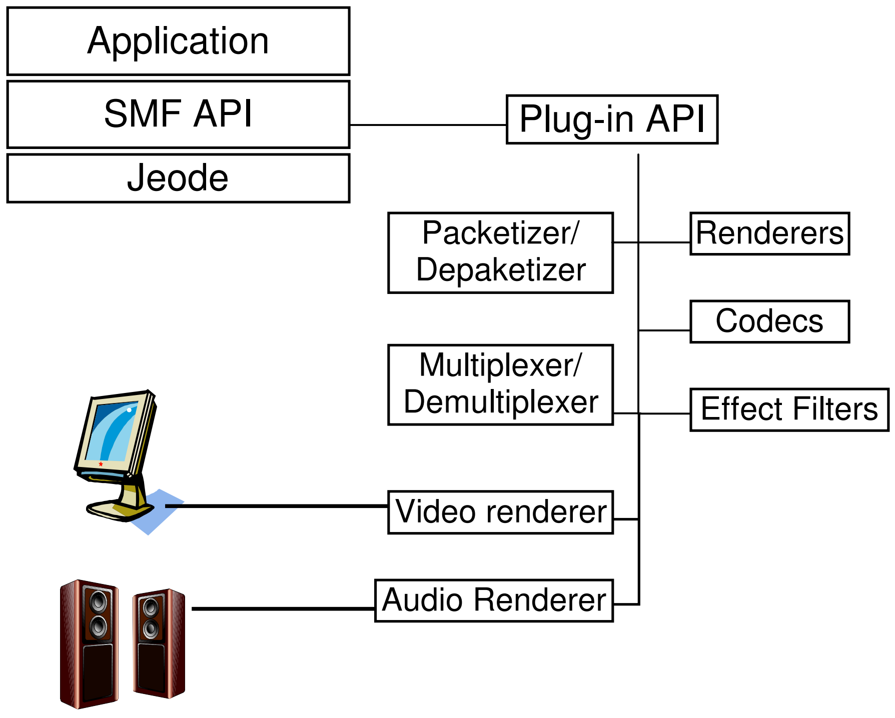

Fig. C.3 SMF Framework