Implementation of an Internet Session Invitation Protocol

by

Stefan Hoffmann

Studienarbeit

Berlin, 28. October 1997

Technische Universität Berlin
Institut für Nachrichtentechnik und Theoretische Elektrotechnik
Fachgebiet Telekommunikationsnetze

Prof. Dr. Adam Wolisz
Supervisors:
Dr. Henning Schulzrinne (Columbia University, New York)
Dorgham Sisalem (GMD FOKUS, Berlin)

Contents

Chapter 1

Introduction

In recent years, the Internet, which was initially designed as a research network has grown up to a globally available information and communication network. Nearly every user who has a computer has the ability to join the Internet.

In the same time, lots of different Internet services has been developed. Currently, multimedia services like transmission of audio and video data become more and more popular and nearly every new PC or workstation is equipped with multimedia capabilities.

One feature which combines a workstation's multimedia capability and its Internet connectivity are so-called multimedia conferences. Within such a conference, multimedia data like audio or video is transmitted in several media streams, called sessions, between the participating users. Unicast sessions perform point-to-point transmissions between two Internet hosts, but multicast sessions can be used to exchange data between several workstations simultaneously. Thereby a sender transmits data to a multicast group (specified by a multicast address and time-to-live (ttl) value) which will be received by each application which is a member of the group.

To perform the transmission of multimedia streams, special protocols like the Real-Time Transport Protocol (RTP) [14] have been developed. Based on these protocols, applications like vat or NeVoT [15] for audio data and vic [10] or NeViT [21] for video data have been implemented. Moreover, the integrated session controller isc combines different media agents to a modular, flexible Internet conference tool. Thereby, isc offers the graphical user interface to control and configure the different media streams.

The applications mentioned above only deal with mechanisms to join, leave and process media streams. To take part in a multimedia session, the session parameters must be known by all users who want to participate. These parameters consist of the address of the multicast group and the ttl value for multicast sessions or the destination host address for unicast streams, the port numbers and encodings of the media streams, etc.

One of the problems of joining multimedia sessions is how potential session members can get information about the session parameters. Currently, two basic approaches are used:

- Users obtain a session advertisement via multicast or a web page which contains the session parameters;
- Users are explicitly invited to multimedia sessions by another user.

To use the former (session advertisement), different tools like sd or sdr exist. Moreover, inviting of session members seems to be useful to initiate multimedia sessions spontaneously. This also fits the problem, that it can't be guaranteed that any user who should reach the session has seen the session advertisement.

Users could be invited explicitly via email, but the delay in delivery and reading makes telephony-like spontaneous or private conferences difficult [20]. Using the ordinary telephone service to call-up each prospect user seems to be cumbersome and could additionally become expensive.

The goal of this work is to implement a mechanism to perform session invitation over the Internet. Thereby, the implementation should fulfill the following requirements:

- The specification of the session parameters within the application should be easy to allow users who don't know much about multimedia sessions to use the application. Indeed, an expert user should also have the possibility to specify the session parameters in a flexible way.
- The caller should have the possibility to invite users to new and to existing sessions.
- The inviting user should have the possibility to specify the invitee's address in an easy way.
- After performing an invitation successfully, the multimedia sessions should be created automatically.
- The invited user should have a possibility to handle invitations automatically.

To perform session invitation, a globally standardized mechanism should be used. The Session Initiation Protocol (SIP) [7] which is currently under discussion within the IETF MMUSIC working group seems to be the best choice. Since it is still under development, one additional task of this work is to find problems within the specification during the implementation.

1.1 Structure of this Work

Chapter 2 deals with a description of the Session Initiation Protocol (SIP) [7] which specifies the invitation mechanism;

Chapter 3 gives a short overview of the Session Description Protocol (SDP) [6] which is used to describe the parameters of sessions within a SIP message;

Chapter 4 presents the software architecture of the implemented and enhanced tools;

Chapter 5 describes the Session Invitation Daemon (sipd) which is used to receive invitation requests;

Chapter 6 describes the location service mechanisms which are used to find the invited user;

Chapter 7 describes the enhancements to the Integrated Session Controller (isc); isc is the user interface to perform session invitation;

Chapter 8 offers a summary to this work.

Appendix A describes the functions of the Session Invitation Daemon (sipd);

Appendix B offers an instruction manual for the components sipd and isc;

Appendix C offers the source code of the software on a floppy disk.

Chapter 2

The Session Initiation Protocol

2.1 Motivation

In recent years, multimedia sessions over the Internet have become more and more popular. They are designed to allow users to exchange different kinds of data like audio and video streams in a comfortable way. To join such a session, the session parameters like session name, session multicast address or ports of the participating applications must be known by all users.

Session members can get these pieces of information in two different ways:

- The session and its parameters are advertised and users who see the advertisement can join the session;
- Users are explicitly invited to a session.

To initiate a multimedia session with many users spontaneously, session advertisement is not useful because it can‘t be guaranteed that every prospect participant will see the session advertisement.

Session invitation can be done differently: Users can be invited by performing telephone calls or by sending emails. Indeed, both types of invitations have disadvantages: A telephone call to many participants would be expensive and intricately; when sending emails it can't be guaranteed that each user receives this invitation during a short interval of time.

This chapter presents the Session Initiation Protocol (SIP) [7] developed by M. Handley, H. Schulzrinne and E. Schooler, which was designed to allow session invitation over the Internet in a comfortable way. Please note that this work is based on the SIP draft *"draft-ietf-mmusic-sip-02"* [7] which is currently replaced by a newer version (*"draft-ietf-mmusic-sip-03"*) [8].

2.2 Overview

The Session Initiation Protocol, which is based on the client-server model, is designed to invite users or services like video servers to multimedia sessions. In general, an inviter at the client side specifies a user to be invited to a multimedia session and the appropriate session parameters. The SIP client creates a SIP request message, tries to locate the invitee and sends the request to the called user's SIP server. The server notifies the invitee who can decide whether to participate in the session or to reject the call. Finally an appropriate SIP response message is send back to the initiating client.

To perform the functionality described above, SIP states how client and server communicate with each other, how users can be addressed and defines the messages which are transmitted. Description of the session parameters is not part of SIP. Indeed, the Session Description Protocol (SDP) [6], described in Chapter 3, could be used to describe multimedia session parameters within a SIP message.

2.3 Addressing

To invite a user to a multimedia session, a globally unique identifier like a personal telephone number must exist to address the receiver of the session invitation. In the computer realm, the equivalent to that telephone number combines the user's login id with a machine host name or numeric network address:

- `sho@rockmaster.fokus.gmd.de`
- `sho@193.175.132.225`

Indeed such a user address is difficult to memorize or becomes invalid if the user is logged in at another host. Thus a well-known, relatively location-independent address should be used to address a user.

Almost every Internet user has his own email address which consists of a user and a domain part, e.g. `hoffmann@fokus.gmd.de`.

The domain part is relatively location-independent and the email address should be well-known, thus both requirements mentioned above are fulfilled. On the other hand, getting the actual location of the user in the domain requires an additional feature, a location service.

SIP allows both forms of addressing to be used, with the latter requiring a location server to locate the user.

2.4 Call Setup

Calling a user is a multi-phase procedure. First, the initiating client tries to determine the address of the called user's SIP server. The local client checks if the user address consists of a numeric network address. If so, then that is the address used for the remote user agent. If not, the requesting client looks up the domain part of the user address in

the DNS. Therefore the client initiates a DNS query. If a new service (SRV [4]) resource record is returned, this is the address of the domain's SIP server which will be contacted next. If neither a resource nor an A but a MX record is returned, then the mail host is the address to contact next.

Presuming an address for the invitee's SIP server is found, the second and subsequent phases implement a request-response protocol. A SIP request including a session description (typically using SDP format) is sent to the appropriate SIP server which tries to locate the called user in his domain and handles appropriately.

SIP requests and responses may be sent over a TCP connection or via UDP to a well-known SIP port. A reply must be sent by the same mechanism the request was sent by. Hence if a request was sent by TCP, the SIP server must reply via TCP, too.

2.5 Locating a User

When a SIP server receives a request, it has to locate the requested user or service in his domain. Since it can't be expected that a user is always logged in at the same host, a location service is required which registers the users and their locations dynamically. If user location fails an appropriate response is send to the client, otherwise the SIP server has two different possibilities to handle the request:

Proxy mode: The server forwards the request directly to the location(s) returned by the location server

Redirect mode: The server sends a SIP response including the location(s), returned by the location server, to the client to allow him to send a new request to the new user address(es)

Whether to forward or redirect the request is up to the server itself. On firewall machines or if a multicast address is given by the location service, the former (forwarding) should be preferred.

2.6 SIP Messages

Before describing the functionality of SIP in a more detailed way, this chapter deals with the message format used by SIP. Two different types of messages are supported: SIP requests which are sent from client to server and SIP responses which are transmitted in the opposite direction. In general, all messages are text based with a close alignment to HTTP/1.1 [3]. Requests and responses consist of a start line (which characterizes the type of the message), one or more header fields, an empty line which indicates the end of the header section and optionally a message body.

Each header field, which can be distinguished in general-header (valid for all kind of massages), request-header and response-header, consist of a name followed by a colon and the field value. Like the start line, all headers are terminated by a carriage-return line-feed (CRLF) sequence.

The presence of a message body depends on the kind of message and is indicated by a "*Content-Length*" header field which specifies the length of the body in bytes and a "*Content-Type*" header indicating the format of the message body. If the payload has undergone any encoding then it must be indicated by a "*Content-Encoding*" header field.

2.6.1 Header Fields

SIP header fields are used to specify inviting and called user, the path the request has traversed so far, the kind and length of the message body, reasons to allow negotiation and other options needed to enable a successful SIP communication. *Content-Length*, *Content-Type*, *To* and *From* header fields are compulsory, other fields may be added as required. Ordering of the header fields is not of importance except of *Via* fields (see 2.7.2), reordering is not allowed.

Table 2.1 gives a short overview of existing header fields. For a complete and more detailed list see [7].

Header	*Description*
Accept-Language	indicates to the server in which language the client would prefer to receive reason phrases
Contact-Host	specifies the host the user was located on
Content-Encoding	specifies encoding of the message body
Content-Length	specifies the message body length in bytes
Content-Type	specifies the format of the message body
From	indicates the invitation initiator
Retry-After	indicates when the called party may be available again
Reason	specifies reasons why a session description cannot be supported
To	indicates the invited user
Via	indicates the path the request has traversed
Location	IP address of a callee's location

Table 2.1: SIP header fields

2.6.2 SIP Request

Requests are characterized by the first line beginning with a method token, followed by a unique request identifier (request-URI) and the SIP protocol version, each separated by

space characters. Currently two kinds of requests are supported which are distinguished by their method token:

INVITE: This method is used to invite a user or service to participate in the session and must be supported by all SIP servers.

OPTIONS: This method is used to query the users or services as to its capabilities. Support of this method is optional.

The second filed in the first request line, the request-URI, should be a string that can be guaranteed to be globally unique for the duration of the request to identify requests and responses which belong together. Using the initiator's IP-address, process id and instance (if more than one request is being made simultaneously) satisfies this requirement.

After the header section, the message body of an INVITE request deals with the description of the session to which the user or service should be invited to. Session description is not part of the SIP specification, using the Session Description Protocol (SDP) [6] is recommended.

Figure 2.1 deals with a typical SIP request.

```
INVITE 130.149.92.31/3546/1 SIP/2.0
Via:  SIP/2.0/UDP 130.149.92.31
From:  fim@lion.tu-berlin.de
To:  sho@fokus.gmd.de
Content-Type:  application/sdp
Content-Length:  187

v=0
o=user1 53655765 2353687637 IN IP4 128.2.3.4
s=Mbone Audio
i=Discussion of Mbone Engineering Issues
e=mbone@somewhere.com
c=IN IP4 224.2.0.1/127
t=0 0
m=audio 3456 RTP/AVP 0
```

SIP request from *fim@lion.tu-berlin.de* to *mailhub.fokus.gmd.de*: The first line states that it's a call to join a session (method *INVITE*). The request id is a combination of IP address, process id and a sequence number. To indicate the path the invitation has gone so far a *Via* field is present indicating the initiating client at *lion.tu-berlin.de* (IP address *130.149.92.31*). Caller and callee are characterized by the *From* and *To* headers.

Figure 2.1: SIP Request

2.6.3 SIP Response

Responses are messages which indicate the actual status of a SIP server or the success or failure of SIP requests. They are sent from the server to the client. Like SIP requests, responses are characterized by the first line. The status line consists of the SIP protocol version followed by a numeric status code, the request-URI of the corresponding request

and a textual phrase associated with the status code, each element separated by space characters.

Like in HTTP or FTP, the status code is a 3-digit integer result code which specifies success of failure to understand and satisfy the request by the server. It is intended for use by the client automata, while the reason phrase is intended for the human user. SIP responses are subdivided into 6 classes, which are defined by the first digit of the status code (Table 2.2).

Status code	*Response class*
1xx	Informational
2xx	Success
3xx	Redirection
4xx	Client Error
5xx	Server Error
6xx	Search Response

Table 2.2: SIP Response Classes

An example of a SIP response is given in Figure 2.2.

```
SIP/2.0 200 130.149.92.31/3546/1 OK
Via:  SIP/2.0/UDP 193.175.132.209
Via:  SIP/2.0/UDP 130.149.92.31
From:  fim@lion.tu-berlin.de
To:  sho@fokus.gmd.de
Contact-host:  193.175.132.184
```

SIP response from *lupus.fokus.gmd.de* to *mailhub.fokus.gmd.de*:
The first line states that it's a response to a successful invitation (Response code *200*). The id is taken from the request. The response is sent to *193.175.132.209* (*lupus.fokus.gmd.de*), indicated by the first *Via* header, where the SIP server will remove this header and forward the response to *130.149.92.31* (*lion.tu-berlin.de*). The *Contact-host* header indicates the host where the invited user was found.

Figure 2.2: SIP Response

2.7 Server Modes

As mentioned above, a SIP server has two different possibilities to handle an incoming call after locating the invited user. To explain the different modes, a typical SIP request-response scenario is given below.

Figure 2.3 shows an example configuration of user *fim*, currently logged in at host *lion.tu-berlin.de*, who wants to invite user *sho*, identified by his email address (*sho@fokus.gmd.de*). The SIP client at *lion.tu-berlin.de* makes a DNS query to *fokus.gmd.de* which results in a MX record giving the mailserver of domain *fokus.gmd.de* (*mailhub.fokus.gmd.de*). The client generates a request packet and sends it to the well-known SIP port at *mailhub* **(1)**. The SIP server at *mailhub.fokus.gmd.de* receives the call and asks his location server for the locations of the invited user *sho* **(2)**. The search results in host *lupus* **(3)** which is the actual location of the called user.

Behavior of steps **(1)** to **(3)** is equal to both server modes, following activities will be described in the respective sections.

2.7.1 Redirect Mode

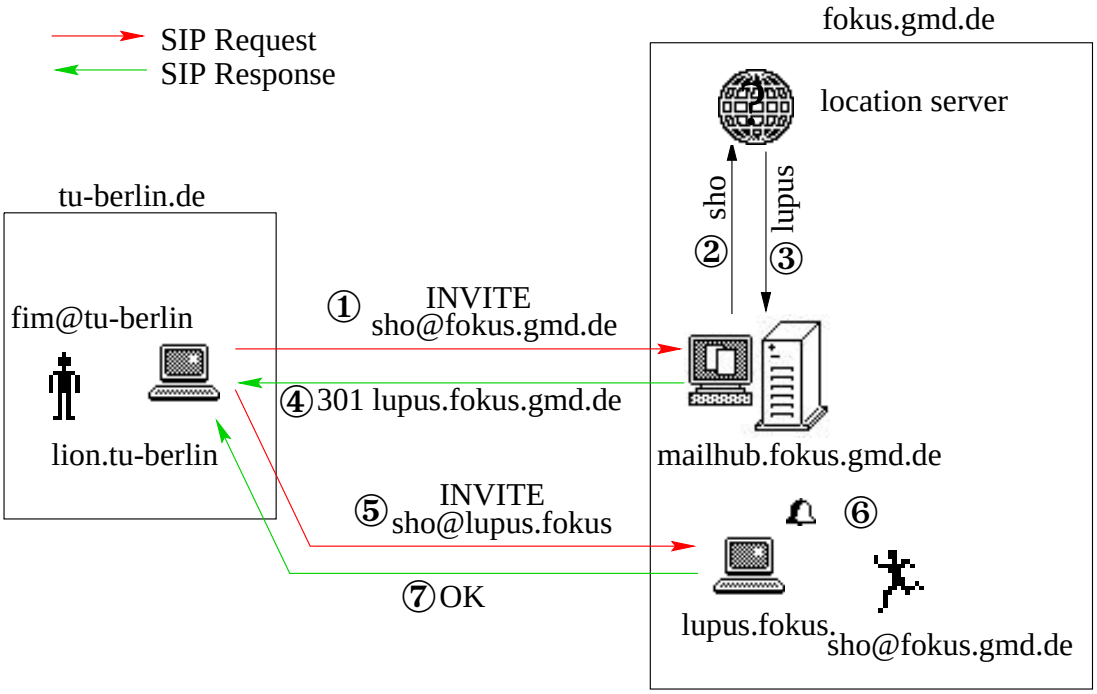


Figure 2.3: SIP Server in Redirect Mode

As shown in Figure 2.3 the SIP server at *mailhub.fokus.gmd.de* acts in redirect mode after receiving a request and locating the invited user (described above). It creates an appropriate redirect response which indicates the actual location of the invitee *sho*, and sends it to the calling SIP client **(4)**. The client itself can extract the given host out of the response packet and is able to create a new request which is addressed directly to the called user's host (*sho@lupus.gmd.de*) **(5)**. After receiving the invitation at host

lupus the invited user is notified **(6)** and can decide whether to accept or reject the call. In Figure 2.3, the invitation is accepted, indicated by a *200 OK* response **(7)**.

2.7.2 Proxy Mode

In contrast to the redirect mode, a server acting in proxy mode doesn't send a redirect response to the requesting client but forwards the SIP request to the location of the called user which is returned by the location server.

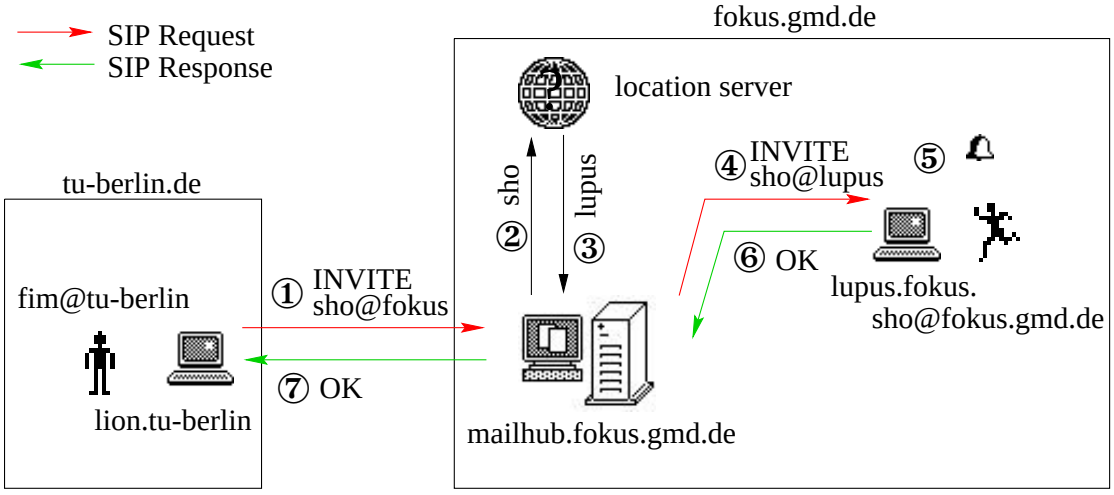


Figure 2.4: SIP Server in Proxy Mode

As Figure 2.4 shows, the SIP proxy server at *mailhub.fokus.gmd.de* locates user *sho* **(2), (3)** and creates a request addressed to *sho@lupus.fokus.gmd.de* **(4)**. The server running on *lupus* notifies the user and creates a response packet according to the user's choice (*200 OK*). This response packet is sent back to the initiating client on exactly the same way it was received **(6), (7)**.

To enable responses take the same path as requests, each proxy server adds a new *Via* header field with his own IP address to a request packet before forwarding it. Since ordering of *Via* headers is of importance, each new *Via* field is added before existing *Via* headers. When a proxy receives a response message (which travels from server to client), it removes its own *Via* header field and sends the response to the server given in the first *Via* header in the response. Another positive effect of using *Via* headers is that looping can be avoided, because forwarding to other hosts is only permitted, if the location isn't indicated by a given *Via* field.

Assuming the location server extracts several locations for the same user, the server can forward the request to these locations by using UDP transport to speed up the request. However, request implosion could result if the request is often forwarded to several locations. Thus servers that are not first hop servers in a chain of servers should not make multiple parallel requests ("*parallel search*"), but send a redirection response with multiple alternatives.

2.8 Reliability

Since SIP supports both TCP and UDP transport, the protocol must guarantee that the exchange of UDP packets becomes reliable. To perform this task, SIP uses a retransmit mechanism when using UDP transport. UDP requests are retransmitted until a definite response* was received or a maximum number of retransmissions is exceeded.

This also takes into account when the transport protocol changes at a proxy server. Assuming a SIP request was received via TCP and should be forwarded via UDP, the TCP→UDP server must retransmit the forwarded request until it receives a definite response.

2.9 Problems when Using Email Addresses

As mentioned in Section 2.3, SIP allows email addresses to specify called users. The user's email address could consist of the user's login id, indeed the local mailserver could also support an alias which is better rememberable like the real user's name. However, it is necessary to extract the user login to locate and notify him. If mapping of alias (e.g. hoffmann) to login (e.g. sho) could only be done by the local mail server, this will cause problems when redirecting or forwarding SIP requests.

2.9.1 Scenario

Assuming a scenario where user *sho* should be invited to a multimedia session. The caller doesn't know the user's login but his email address *hoffmann@fokus.gmd.de*. After extracting the domain's mail server to contact next, the SIP client sends the SIP request to *mailhub.fokus.gmd.de*. There, the SIP server extracts the user login *sho* and locates him at host *193.175.132.225* in the domain.

If the server works in redirect mode, it sends a redirect response indicating the callee's IP-Address which was returned by the location server (*Location:193.175.132.225*). The client modifies the original request's *To* header field by changing the host part but leaving the user part unchanged (*To:hoffmann@193.175.132.225*). This request reaches the SIP server on the given host which can't extract the user's login out of the given alias.

In proxy mode the server forwards the unmodified SIP request directly to host *193.175.132.225*. If aliases mapping isn't possible at this machine, location of the called user becomes impossible.

2.9.2 SIP Modification

As mentioned above, this work is based on the SIP draft *"draft-ietf-mmusic-sip-02"* [7] which is currently replaced by a newer version. In the draft *"draft-ietf-mmusic-sip-03"*[8], the problem described above, is solved by changing some aspects of the SIP

*definite responses are responses which terminate a SIP call (response code higher than 199)

message format.

- The first message line is modified

  The request-URI in the first SIP request line is replaced by a SIP-URL which consists of a user and host part and indicates the next destination of the request. This URL can be modified by SIP servers which replace the host as well as the user part. With regard to the example described above, the SIP server changes the SIP-URL in the first request line from *hoffmann@fokus.gmd.de* to *sho@193.175.132.225* whereby the next SIP server must not extract the login of the called user.

  In SIP responses the request-ID in the first line is totally canceled.

- A new *Call-ID* header filed is defined

  To identify requests and the belonging responses a *Call-ID* header is defined in the *"03"* version of the SIP draft. Like the request-URL in the old SIP version, the call-id must be globally unique during the duration of the request.

- The *Location* header doesn't specify an IP-address but a SIP-URL

  A SIP server in redirect mode sends a whole SIP-URL consisting of user and host part back to the client. If doing so, a SIP server running on the domain's mail server is be able to extract the user login out of a given alias, locate the user and send *login@host* (*sho@193.175.132.225*) back to the requesting client.

Chapter 3

The Session Description Protocol

3.1 Motivation

The Session Initiation Protocol (SIP) [7] is used to invite users to multimedia conferences. However as it only states how communication between inviter and called user, addressing and user location should be done, there is also need to describe a multimedia session within a SIP request. The Session Description Protocol (SDP) [6] is designed for this purpose.

3.2 Overview

The purpose of SDP is to describe multimedia sessions in a way to allow the recipients of a media session description to participate in the session.

Thus SDP includes:

- Session name and purpose,
- Time(s) the session is active,
- The media comprising the session,
- Information on how to receive those media (addresses, ports, formats and so on).

Additional information may be:

- Information about the bandwidth to be used by the conference;
- Contact information for the person responsible for the session.

3.3 SDP Specification

SDP session descriptions are entirely textual. They consist of a number of lines in the form
<*type*>=<*value*>.
<*type*> is exactly one case-significant character, <*value*> is a structured text string whose format depends on <*type*>.

A session description consist of several parts, a session-level section followed by zero or more media-level sections. In general, session-level values are the default for all media unless overridden by an equivalent media-level value.

To give an overview of the permitted <*type*> fields see the Table 3.1 to 3.3.

Type	*Description*
v	protocol version
o	owner/creator and session identifier
s	session name
i	session information
u	URI of description
e	email address of the person responsible for the conference
p	phone number of the person responsible for the conference
c	connection information
b	bandwidth information
z	time zone adjustments
k	encryption key
a	session attributes

Table 3.1: SDP Session Description

Type	*Description*
t	time the session is active
r	repeat times

Table 3.2: SDP Time Description

Type	*Description*
m	media name and transport address
i	media title
c	connection information
b	bandwidth information
k	encryption key
a	media attributes

Table 3.3: SDP Media Description

3.4 SDP Example

An example of a session description within an SIP request can be found in Figure 2.1 in Section 2.6.2.

Chapter 4

Software Architecture

After describing the protocols used to enable session invitation in the previous chapters, the following chapters deal with a description of the implemented software. First, the general software architecture is given, followed by a detailed description of the different components.

4.1 General Design

To enable session invitation based on the Session Initiation Protocol (see Chapter 2), a software package with different components is required. At the client side, a user interface is needed which allows the inviting user to specify request and session parameters. Moreover, SIP requests should be created and sent to the appropriate SIP server. Another function of the SIP client is handling of incoming SIP responses. After receiving an invitation response, the calling user has to be informed about success or failure of the invitation.

On the other hand, receiving SIP requests requires a more complex invitation server. After receiving the incoming SIP call, it has to locate the called user and must decide whether to redirect or to forward the request. Additionally, a user interface is required which notifies the called user and lets him specify whether to accept or reject the invitation. According to the user's choice, a SIP response must be created and sent back to the initiating SIP client.

As described above, it seems to be useful to implement a single SIP client and a SIP server, although another architecture, presented above, could be more reasonable. Assuming a SIP server which runs on the domain's mail server host, this server is queried for every invitation in which the callee is identified by his email address. Most of the time, the user will be logged in on another host, so the SIP server is only used for redirection or forwarding. A user interface which indicates an incoming call is only needed at the host of the called user. Furthermore, a SIP server should run all time and therefore has to be robust. It waits for incoming UDP packets or TCP connection requests and handles them appropriately. All these requirements are best fulfilled when running a SIP server as a daemon. A daemon is a process that executes without an associated terminal or login shell ("it runs in the background"), either waiting for some

event occur or waiting to perform some specified task on a periodic basis [22]. All parts of the software package which require user interaction are separated into another tool. This tool should fulfill requirements for user interaction for both, client and server side, which seems to be useful since it is expected that a user will use one single tool to initiate and react to SIP requests.

Location of users which is needed to perform session invitation, can be done in several ways and is not described in the SIP specification, so different kinds of location servers are conceivable. To make the SIP server daemon flexible, the location server is the third separate software component.

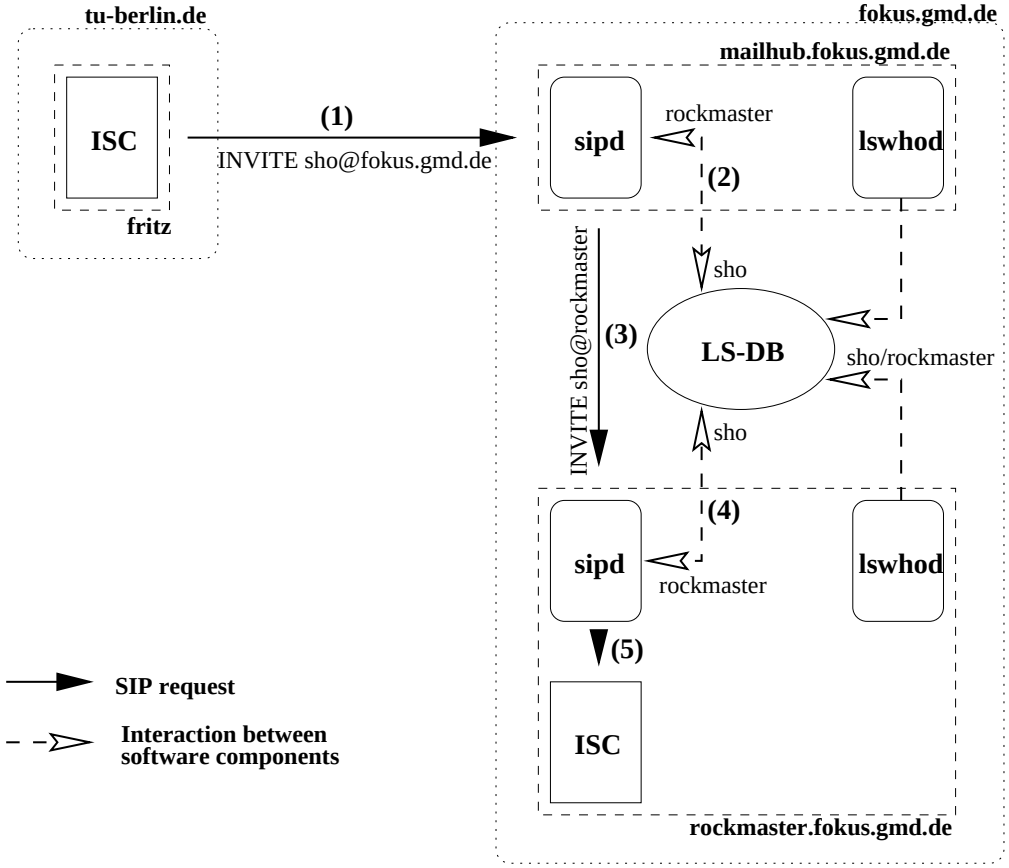


Figure 4.1: Software Architecture

Figure 4.1 shows an overview of the currently implemented software architecture and the interaction between the different components during the process of a SIP request. In detail, the three components mentioned above consist of:

1. isc

   The Integrated Session Controller (isc) includes the user interface to perform session invitations based on SIP. It is used to specify the invitation and session parameters and to send the created request to an appropriate SIP server. When receiving a request, it checks if the requested media is supported, informs the called user and creates a SIP response appropriate to the invitee's selection.

2. sipd

   The Session Invitation Daemon (sipd) is the SIP server which performs user location and call forwarding or redirection for incoming SIP requests according to the SIP specification. It has to run on each host which should be able to receive and handle SIP requests.

3. lswhod

   The location service daemon (lswhod) is used to maintain and update a location service database (LS-DB). The data base contains information about the current location (host) of any user in the domain. Therefore, lswhod has to run on each host which should be registered in the location service database.

Typically, a SIP request is created by isc which sends it to the appropriate SIP servers sipd, according to the specified invitee's address **(1)**. The Session Invitation Daemon queries the location service data base LS-DB to extract the current location of the invited user **(2)**. In proxy mode (as shown in Figure 4.1), sipd forwards the call to the next SIP server at the destination host **(3)**. Here, sipd also asks the location service data base for the location of the callee, which results in the local host name. This causes sipd to forward the request to the Integrated Session Controller which informs the callee about the incoming call.

On the reverse path, a SIP response will take exactly the same way back to the initiating client as requested by the SIP specification:

isc at rockmaster → sipd at rockmaster → sipd at mailhub → isc at fritz.

Chapter 5

The Session Invitation Daemon

5.1 Overview

The Session Invitation Daemon sipd is designed to handle session invitations based on the session initiation protocol (SIP), described in Section 2. After receiving a SIP request, it has to locate the invited user in the local domain and forwards (either to the next SIP server or to the local session controller isc (Figure 4.1)) or redirects the call. An additional feature is the automatic reply function which works similar to the ./forward file in the sendmail system. It allows the invited user to redirect or decline a request automatically according to the request's sender or subject.

When acting in proxy mode (forwarding a request), sipd also handles incoming SIP responses from the next SIP server.

To enable successful session invitation, the SIP server sipd has to run on each host where users should be able to be invited. It is expected that sipd runs as a root process because it must be able to start the session controller isc under the environment of the called user, if it is not already running. On the other hand, if root permissions are not available for the user who wants to install sipd, each user who wants to have the ability of being invited has to start the daemon on his workstation. If there are several users logged in on a single host, only the user who is logged in at the console should start sipd, because only one process can bind to the standard SIP port number and it is expected that only users at the console can participate via audio or video.

All hosts within a domain on which sipd runs can operate as a SIP server for the whole domain. Typically the daemon should also run on the domain's mail gateway host, so that callers can make use of the existing MX records to find the SIP server of a domain. When addressing an invitee by using his email address, the SIP daemon at the mail host will be queried if no SRV record was returned when the SIP client searches of the domain's SIP server (see Chapter 2).

5.2 Processing of Incoming Calls

After receiving an incoming SIP call, the SIP server sipd starts a multi-phase sequence which consists of the following steps:

1. **Map the name of the callee to his local user name**

   Before sipd is able to process the incoming request, it has to resolve the local user name of the called user since it is used for user location, enabling of the automatic reply function and if the daemon has to start the session controller in the invitee's environment.

   The Session Initiation Protocol supports usage of "login@host" or the email address to identify the called user, with the latter requiring a mechanism to resolve the local user name since the user part of an email address can be different to his login.

   To perform name mapping different mechanisms are used. The best information to resolve the local user name should be found in the same data base which is used by the local mail server. In the current version, sipd supports databases formated as "alias: user1, user2, ..." which is the standard format of the sendmail mail system. This database could be typically found at /etc/mail/aliases.

   If the mail aliases data base can't be read by sipd (because it isn't exported via NFS or only root can read it and sipd was not started as a root process), name mapping is based on the local password file and its NIS equivalent.

   In both cases, name mapping may result in exactly one or in a list of user names. If the result is unambiguous, the daemon continues in request processing, otherwise it should either send a redirect response returning a list of all user names or create an *"Ambiguous"* reply. In the current version, sipd performs the latter because of protection of data privacy. In later versions of sipd, this should be configurable.

2. **Check if the invited user wants to redirect or decline the incoming call automatically**

   Similar to sendmail using the $HOME/.forward file to determine mail handling, sipd uses the $HOME/.sip/sip_handler file to make use of SIP more comfortable for users. It is used to decline or redirect a SIP call depending on its initiator, identified by the SIP "*From*" header, or the subject of the call, identified by the first information field in the request payload. (In the next version of sipd, which will be based on the SIP draft *"draft-ietf-mmusic-sip-03"*, the subject of a request will be indicated by a SIP *"Subject"* header.)

   Besides, the $HOME/.forward file should be used to indicate that a user has moved to another domain. This could be done by initiating a handler which replies to all incoming calls with an *"Moved Permanently"* or *"Moved Temporarily"* response.

   The format of the $HOME/.sip/sip_handler file is line-based, with each line identifying one sip handler. Each line consists of four parts which are separated by *TAB* characters:

`[from|subject|*] TAB [source address|subject string] TAB [reply code] TAB [reason phrase|destination address]`

The first and second argument are used to identify calls which should be handled automatically, whereas the third and fourth argument state how to react to these calls. Three different identifiers are defined for the first argument to indicate whether to examine the call initiator (identified by a *"from"* value), the call's subject (*"subject"*) or to handle all incoming calls (specified by *"*"*). The second argument consists of a SIP address or a subject string according to the first part of a sip_handler line.

To indicate how to react to the specified call, the third argument always specifies a SIP reply code indicating the response which will be sent to the client by sipd. Depending on this reply code, the last argument of a handler line specifies an appropriate reason phrase or a SIP address if a redirect response code was given.

sipd always reacts to the first handler in the $HOME/.forward file which matches to the incoming call.

3. **User location and appropriate call handling**

   After resolving the local user name and evaluating the automatic reply function, the invitation daemon sipd has to determine at which workstation the called user is currently logged in directly. In the current version, sipd uses a location service data base which is created and updated by the location service daemon lswhod (Section 6.2) to get the callee's location.

   Depending on the number of the extracted locations, sipd handles the request differently:

   - **The callee is logged in at exactly one host**

     If the query of the location service data base results in only one single host which is not the name of the local host, the call will be forwarded to the SIP server at this remote machine. This will be done by using the same kind of transport the request was received from sipd (i.e. requests, incoming via TCP are forwarded by using TCP; requests, incoming via UDP are forwarded by using UDP).

     On the other hand, if user location results in the host at which the request is currently handled by sipd, the call has reached the destination machine and the called user has to be informed. Therefore, sipd forwards the request to the local session controller isc which will notify the callee (see Figure 4.1).

   - **The invited user is located at several hosts**

     If the called user is logged in at several hosts, and sipd is the first proxy within a chain of proxies (indicated by the number of via headers), the request will be forwarded to all given locations (including forwarding to isc if one location is the local host). This "parallel search" is done by using UDP as mentioned in the SIP specification.

     The SIP specification doesn't allow parallel searches for a SIP server which is not the first server. So, sipd sends a redirect response which indicates all possible locations of the invited user to the SIP client. To avoid that a

redirect response is sent if the user is logged in at the local host, sipd checks if it is one of the extracted locations. If so, it will send the call to isc to notify the callee. When receiving a definite response from isc, this response is sent to the SIP client; call forwarding to other locations is not necessary. Otherwise, if no definite or final response was received from isc during a specified interval of time, a redirect response will be sent to the calling client.

5.3 Interface between sipd and isc

As described in Section 5.2, sipd forwards a received SIP call to isc if the location service returns the local host as a possible location of the invitee. Afterwards, it awaits a response which belongs to the request.

Since the Integrated Session Controller (isc) uses the pattern-matching multicast mechanism (pmm) [17] to communicate with its local media agents (7), the same mechanism is chosen to enable communication between sipd and isc.

In the pattern-matching multicast mechanism, all participating applications use the same local multicast group to communicate with each other. Since all agents receive all messages which are sent to the common multicast group, a mechanism is developed to indicate the destination of a message. This is is done by using a so called "pmm header" which consists of identifiers for the conference, the media agent and media instance. For example:

"C/audio/3" specifies the third audio session within the conference C.

The message introduced by the given pmm header is only handled by the audio agent which handles the appropriate session.

To use the pmm format, the pmm header is simply modified for SIP messages between sipd and isc. The media agent identifier for SIP messages is *"sip"* and the conference identifier is replaced by the SIP call-id. An instance identifier isn't needed to transmit SIP messages, since the call-is is globally unique. To maintain the defined pmm header format the instance identifier is set to zero:

"call-id/sip/0"

To avoid conflicts with the pmm header format all slash characters ("/") in the call-id are replaced by underline characters ("_").

5.4 Forwarding Requests to isc

When forwarding a request to isc, sipd creates a pmm header and sends the received request to the local pmm multicast group. Since it can't be sure that isc is already running, a mechanism is required which starts the session controller.

After sending the call to the pmm group, sipd starts a timer. If isc runs and receives the request, it replies with a *"Trying"* response via the local multicast bus which lets sipd cancel the timer. If the timer expires (no response is received by sipd), it is expected that isc has to be started which is initiated by the invitation daemon. Afterwards the

SIP request is retransmitted to the multicast group.

5.5 Processing of Incoming Responses

When acting in proxy mode, sipd waits for responses belonging to the forwarded request and sends the reply to the calling SIP client.

If a request was forwarded to several locations (*"parallel search"*), sipd receives responses from each server the request was forwarded to, but the client which queries sipd only awaits one definite or final response. Therefore, sipd supports an internal priority mechanism which allows to classify all incoming responses by their response code due to the SIP specification. So, the daemon stores the response with the highest internal priority for each forwarded invitation. This response is sent to the client if a definite response was received form all locations or sipd receives a final reply which terminates the parallel search.

5.6 Implementation

5.6.1 Concepts

The Session Invitation Daemon is event driven, two different kinds of events can occur:

socket (file descriptor) events: Socket events are used to react to incoming messages. Each used socket is bound to an event handler which invokes an appropriate function if the socket becomes readable. Theses sockets are the TCP and UDP server socket, the socket to the local pmm multicast group and socket which are created if a request was forwarded to a remote SIP server via TCP.

timer events: Timer events are used to avoid deadlocks in sipd. They invoke timeout functions which will handle the processing request appropriately.

To allow sipd to handle several requests concurrently, the daemon supports a invitation list which stores all active invitations and their parameters which are used for a successful processing. The format of the list entries is given in Section 5.6.3.

Functions like name mapping, automatic reply and user location are separated into Tcl scripts so that the appropriate functions can be easily modified.

5.6.2 Global Variables

sipd uses global variables which are listed in Table 5.1.

Variable	*Description*
char myAddr[17]	IP address in dotted decimal notation of the local host where sipd if running
int tcp_server_socket, udp_server_socket	server sockets for incoming SIP messages
int isc_socket	local multicast socket to isc (pmm)
int isc_connections	counter for "connections" between isc amd sipd
int sip_port	standard SIP port
inv_list_t *invitation_list	pointer to the invitation list which stores settings of currently active invitations

Table 5.1: Global Variables

5.6.3 Structures

To store the settings and parameters of incoming SIP requests and responses and to store extracted locations, two structures are defined in sipd. The `inv_list_t` structure, shown in Table 5.2 stores the incoming request, parameters which are used to process the invitation successfully and incoming responses which belong to the request.

Location information of invited users is stored in the `loc_list_t` structure, shown in Table 5.3. It is used, to manage call forwarding or redirection correctly.

sipd creates an invitation list consisting of `inv_list_t` structures to store the settings of all requests which are currently in process. Each structure handles one SIP call. If a new request is received, a new `inv_list_t` structure is added to the invitation list and is removed if processing of the request is finished.

To store new locations for an invited user when performing user location or if *"Alternative Address"* responses are received, each `inv_list_t` structure points to a list of `loc_list_t` structures. If a new location was extracted an appropriate `loc_list_t` structure is added to the location list of the appropriate invitation list entry.

Identification of special list entries within the invitation list or the location list, the `inv->call_id` and `loc->addr` fields are used.

Figure 5.1 shows the structure of the invitation list.

5.6.4 Timers

To avoid deadlocks during the processing of requests, several timer mechanisms are provided. Table 5.4 gives an overview of the used times and their values.

Variable	*Description*
char *call_id	call_id of the stored request; identifier of the structure
char *user	user specified in the first SIP request line
struct passwd *userEntry	pointer to invited user's passwd structure, needed to start the session controller in the invitee's environment
char *request	incoming SIP request
char *response	SIP response created by sipd; SIP response received from next server or isc with highest priority
int response_priority	priority of the SIP response stored in `response`
char addr_of_client	IP address of the client from which a request was received if it was received via UDP
int socket_to_client	socket to the client from which a request was received (TCP)
int port_to_client	port of the client from which a request was received (UDP)
int transport_to_client	transport protocol the request was received (SOCK_STREAM or SOCK_DGRAM)
int socket_to_server	socket to the next SIP server when acting as TCP proxy
int transport_to_server	transport protocol to the next SIP server (SOCK_STREAM)
int state	current state of the request
int locations	number of possible locations of the invitee; count of locations from which no definite response has been received
loc_list *location_list	pointer to the invitation's location list
struct inv_list_t *next	pointer to the next invitation list entry

Table 5.2: inv_list_t structure

Variable	*Description*
char *addr	IP address in dotted decimal notation of the location
char *hostname	hostname of the location
char *user	user part of the SIP location (user@host)
int retransmissions	counter of transmissions of the request to the location when forwarding via UDP
int response_received	flag to indicate if a definitive response was received from this location; when flag is set (=1) incoming responses from this location are ignored
struct loc_list_t *next	pointer to next location entry
struct inv_list_t *inv_parent	pointer to inv_list entry belonging to this location

Table 5.3: loc_list_t structure

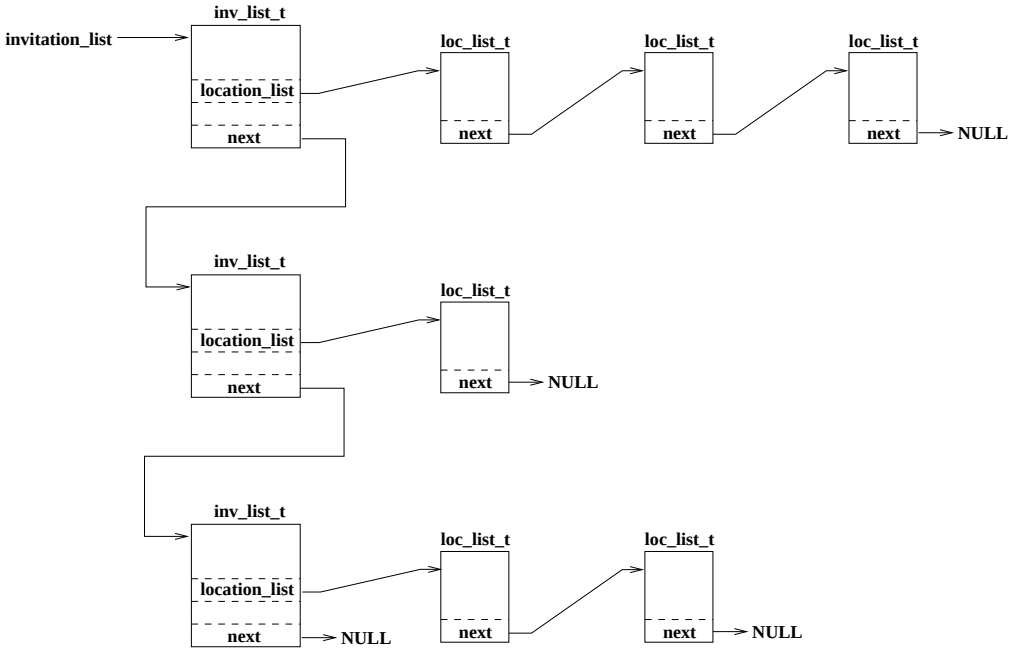


Figure 5.1: Structure of the invitation list

Timer	*Value*	*Description*
T_request	3 min - N*10 sec	Timer for the total processing time of an request forwarded to the next SIP server. N equals the number of proxies the request has reached so far. Calls *request_timeout()* if expires.
T_retransmit	10 sec	Retransmit timer used if a request is forwarded via UDP to the next SIP server. Periodically reinvoked until a definite response is received from the SIP server or a maximum number of retransmissions is reached.
T_isc_req	1 min	Timer for receiving a definite response from the session controller if a request was forwarded to isc. Invokes *isc_request_timeout()* if expires.
T_isc	2 sec	Timer to receive a *"Trying"* response from isc which indicates that the session controller is running.
T_start_isc	20 sec	Timer to wait until isc is started. Calls *start_isc_timeout()* if expires.
T_close_inv	1 min	Timer to free the invitation settings in the invitation list.

Table 5.4: Timers

5.6.5 Procedures

The procedures implemented for sipd are listed in the Appendix A. Ordering is based on the procedure's appearance in the appropriate files. First, files containing support functions are listed (Table 5.5), whereas SIP functionality (Table 5.6) starts within the *event.c* file.

File	*Description*
notify.c	sets up event handlers
multimer.c	establishes timers
servers.c	expands a domain name in a list of SIP servers
misc.c	manipulates strings in SIP messages
list.c	maintains the invitation and location list
udp.c	UDP communication
tcp.c	TCP communication

Table 5.5: sipd files implementing support functions

File	*Description*
event.c	Event handler invoked if a server socket becomes readable, start of an invitation process
request.c	Functions which handle new SIP requests
proxy.c	Functions to perform call forwarding
response.c	Functions which handle all kinds of responses
isc.c	Function used to communicate with isc
timer.c	Event handlers invoked by timeouts
sipd.c	sipd main routine and initialization functions

Table 5.6: sipd files implementing SIP functionality

After installing event handlers for the server sockets `udp_server_socket` and `udp_server_socket`(file *sipd.c*), processing will start with the *socket_event()* function (*event.c*) if one of the server sockets becomes readable.

Incoming SIP requests are handled by the *process_request()* function (*request.c*), whereas *process_response()* (*response.c*) is invoked if a SIP response arrives.

The *isc_socket_event()* function (*isc.c*) builds the event handler for a readable `isc_socket`.

5.6.6 Scripts

As mentioned in 5.6.1, functions which should be easy to modify are extracted to Tcl or shell scripts. This section deals with a short description of the functions used to perform name mapping, check call acceptance and locate a user.

5.6.6.1 The *expand* script

The *expand* Tcl script is used to map the name of the called user, given in the first line of a SIP request, into the local user name. It is invoked by the *expand_user_name()* function (file *request.c*) with the name of the callee as argument.

First, *expand* invokes the aliases application which performs name mapping based on the local alias data base which is also used by the local mail system. Therefore, *expand* invokes aliases with the given user name. To specify the data base which should be queried by aliases, the command line option *"-a"* followed by the path to the data base can be used. If no data base is specified, */etc/aliases* is queried. If the user name can be found in the data base, the *expand* script returns the local user names to stdout and exits.

To enable name mapping if the aliases application can't match the invitee's name to the local user name, *expand* tries to get the local user id by using the namemapper application. namemapper searches in the system password files for user names that match the command line argument which specifies the name of the called user.

If name mapping was successful, namemapper returns one or more strings of the format:

- U|L|F user_name name, e.g.: *U sho Stefan Hoffmann*

The first letter (U, L, F) designates if the match was on the user name (*"namemapper sho"*), last name (*"namemapper hoffmann"*) , or first name (*"namemapper stefan"*).

expand returns the strings extracted by namemapper .

5.6.6.2 The *auto_reply* Script

The *auto_reply* script is invoked by the *automatic_reply()* function to check if an invitee wants to decline or redirect the incoming SIP call automatically. Therefore the script is invoked with the request's sender (*From* header) and subject, and checks if the they match to a line in the $HOME/.sip/sip_handler file. If so, *auto_reply* returns the appropriate handler string (response code and reason phrase or redirect location) to specify the SIP response which is sent to the calling SIP client automatically.

5.6.6.3 The *locate* Script

The *locate* script is invoked by the *locate_user()* function and searches in the location service data base, created and updated by lswhod (Section 6.2), for hosts where the called user might be logged on. The script returns the list of the extracted hosts.

Chapter 6

Location Service

6.1 Overview

One feature which differentiates between a session invitation tool based on SIP and other existing invitation tools like showme, is the ability to automatically locate invitees within a domain. To perform this task, a user location mechanism is required. Since SIP doesn't specify how user location should be done, different methods are thinkable and should be supported. Several location service mechanisms are listed by Schulzrinne in [19]:

- **File system registration:** For example, users create or remove a .location.$HOST file when loggin on or leaving a worstation. This should automatically be done by the .login and .logout file;
- **lswhod:** Location service based on the who (utmp) data base on each host;
- **tracker:** Location service program based on the finger protocol;
- **Multicast location:** SIP requests are multicast to a local address, either a global address for all users or a user-specific address;
- **Service location protocol:** It might be possible to employ the service location protocol [23] to find people.

To enable an easy way of changing the location service mechanism, the Session Invitation Daemon (sipd) uses a Tcl script as its interface to the location server. This script is described in Section 5.6.6.

In the currently implemented version, the lswhod location server is used. It is described in the following section.

6.2 The Location Server lswhod

To perform location service, the location service daemon lswhod, written by Schulzrinne and loosely based on [25], adds location information for all users logged in at the local

host to a location service data base. To get information about the users who are logged in on a host, the local who database (typically /etc/utmp) is used by lswhod. So, the location service daemon has to run on each host on which users should be found and session invitation should be possible. lswhod periodically checks who is currently logged in and updates the location service data base accordingly.

To make modification of the data base easy, it is located in a common directory d in the NFS which is readable and writable by all users in a domain via NFS. So, an update and also a query of the database only require a single NFS transfer. If a user U is logged in at the host H, lswhod creates a file $d/U/H$, where d is the data base directory. If the user is only logged in locally (typically on the console), the file is empty. Otherwise, the file consists of a list of hosts the user is logged in from. If a user is logged in remotely and locally on one workstation, the local host name and the remote hosts are listed in the file.

Figure 6.1 shows an example scenario of a location service database (LS_DB) maintained by lswhod .

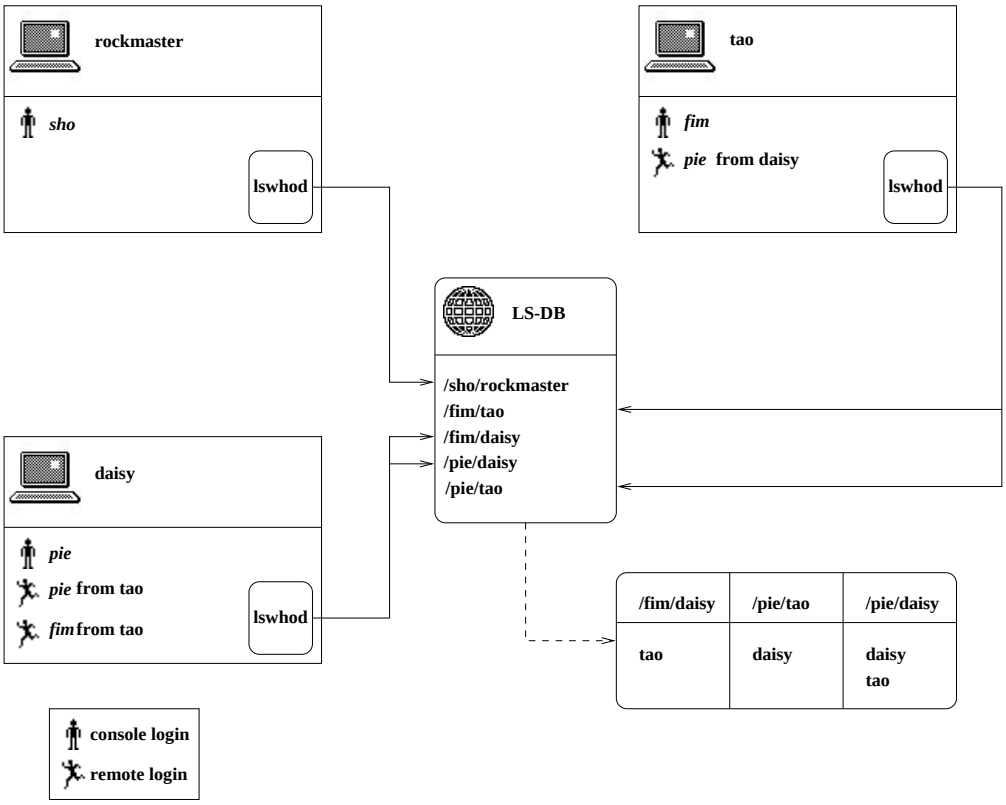


Figure 6.1: Location Service Database updated by lswhod

Chapter 7

The Integrated Session Controller

7.1 Overview

The Integrated Session Controller (isc) is designed to perform multimedia conferences over the Internet. Therefore, it can be combined with several media agents (e.g., NeVoT for audio, NeViT for video) which are used to transmit and receive media streams. isc controls the media agents and offers the graphical user interface to the different applications. The Integrated Session Controller is used to handle a single conference, consisting of several multimedia sessions, by creating and terminating multimedia sessions and by configuring the session parameters. Messages between the session controller and the media agents are exchanged via so-called pmm messages which are transmitted over a local multicast "bus".

Several enhancements are required to enlarge isc with a session invitation feature based on the Session Initiation Protocol combined with the Session Description Protocol. These enhancements can be separated into client behavior features, needed to initiate SIP calls and server behavior features to handle incoming invitations.

- **SIP client behavior**

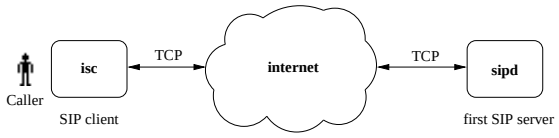


Figure 7.1: isc as SIP client

First of all, isc has to offer a graphical user interface to the caller which allows to declare invitation parameters like the names of users who should be invited and the description of the session. Afterwards, the session controller must create the SIP request message and has to find an appropriate SIP server to contact. After sending the call, isc waits for incoming responses and processes the response code returned. Success or failure of the invitation is announced to the calling user. To make session invitation with isc comfortable, sessions are created automatically after receiving a reply indicating success.

- **SIP server behavior**

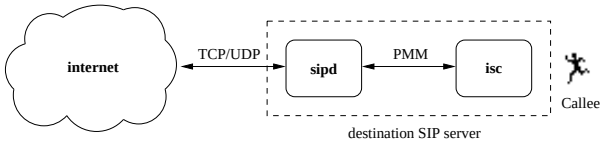


Figure 7.2: isc as SIP server

The Integrated Session Controller deals with SIP server behavior in conjunction with the Session Invitation Daemon (sipd). As described in Chapter 5, the Session Invitation Daemon performs user location, call forwarding or redirection of incoming SIP calls but doesn't offer an interface to the called user which notifies him about an invitation or lets him specify whether to accept or decline a call. The user interaction to incoming session invitations is totally task of the Integrated Session Controller isc. So, isc receives a SIP request from sipd if the daemon has located the called user at the local host. Now, the session controller checks if the media, demanded in the request payload, are supported by isc and an appropriate media agent. If so, the called user is notified and has the ability to decline or accept the call. If he wants to process the invited session, it is created by isc automatically. SIP server behavior ends with sending a SIP response due to the callee's choice to the calling SIP client.

Two additional features are added to the Integrated Session Controller which are used to make session invitation and automatic call handling more comfortable. Firstly, the "phonebook" function enables the user to store aliases for users who are invited frequently. Secondly, isc offers a simple editor to modify the $HOME/.sip/sip_handler file which is used to decline or redirect incoming SIP calls automatically with regard to their initiator or subject (see Section 5.2).

7.2 Local Conference Control Architecture

As mentioned above, isc is one component of a multimedia conferencing application which consists of controllers and media agents. Figure 7.3 shows an overview of the given architecture.

The Integrated Session Controller controlls the media agents which handle the media streams of the conference. Communication between session controller and the media agents is managed by the so called pattern-matching multicast mechanism [16]: All components of the multimedia conference application are members of the same local multicast group. This multicast group is called *"local"* because all messages to this multicast address and port are sent with a time-to-live value of zero which guarantees that no pmm message leaves the local host. To indicate the destination and session to which a pmm message belongs, each message contains a pmm header which consists of the name of the conference *C*, an identifier of the session's media type and a media instance identifier:

"C/audio/3" specifies a message belonging to the third audio session within the

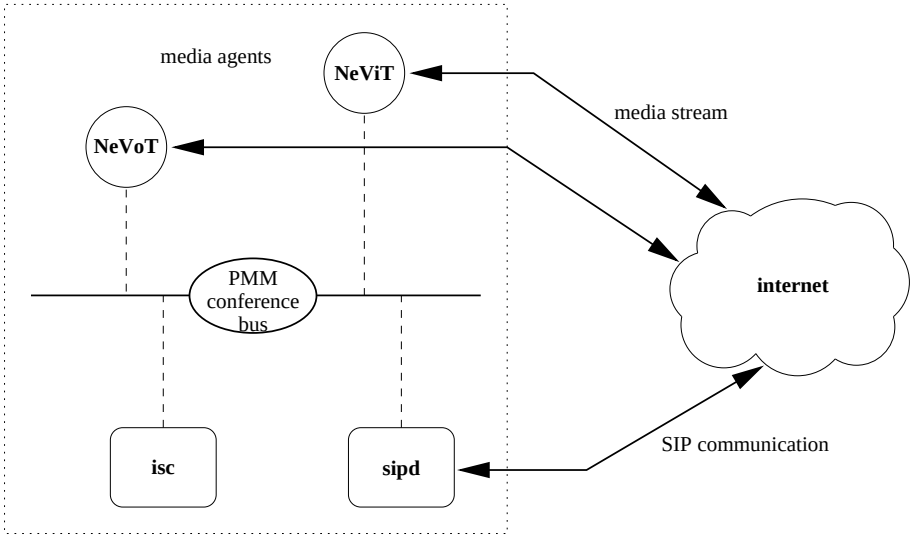


Figure 7.3: Local Conference Control Architecture

conference *C*

The pmm specification also deals with a short protocol wich defines messages for creation, opening, leaving or other actions for mutimedia sessions.

Since the pmm mechanism is used to exchange messages between different applications on the same host, it can also be used to transmit messages between the Session Invitation Daemon (sipd) and isc as required if the local sipd has located an invited user at the local host (Section 5.4). Therefore, an appropriate pmm header must lead the exchanged SIP requests and responses. It consists of the call-id of the SIP message and a *"sip"* identifier. The instance identifier is not needed because of the globally uniqueness of the call-id, but it is set to zero to avoid conflicts with the pmm header format:

"call-id/sip/0"

7.3 SIP Client Enhancements

7.3.1 Initiating of SIP Requests

The SIP client enhancements in isc are used to invite other users to multimedia sessions. Figure 7.4 shows the steps which are performed to initiate a session invitation.

First, isc checks if any multimedia sessions are currently in process. If so, the called *get_active_session_parameters* procedure extracts the session parameters and stores the settings in the globally available `invite` array. The parameters of the active sessions are extracted to allow the caller to invite to these sessions without specifying the parameters manually. Depending on the number of active sessions, the standard (procedure *std_invite*) or advanced (*adv_invite*) invitation window appears. It is expected that a user mostly only uses one audio and/or one video session within a conference. Therefore, the standard invitation window handles these two sessions and makes it easy to

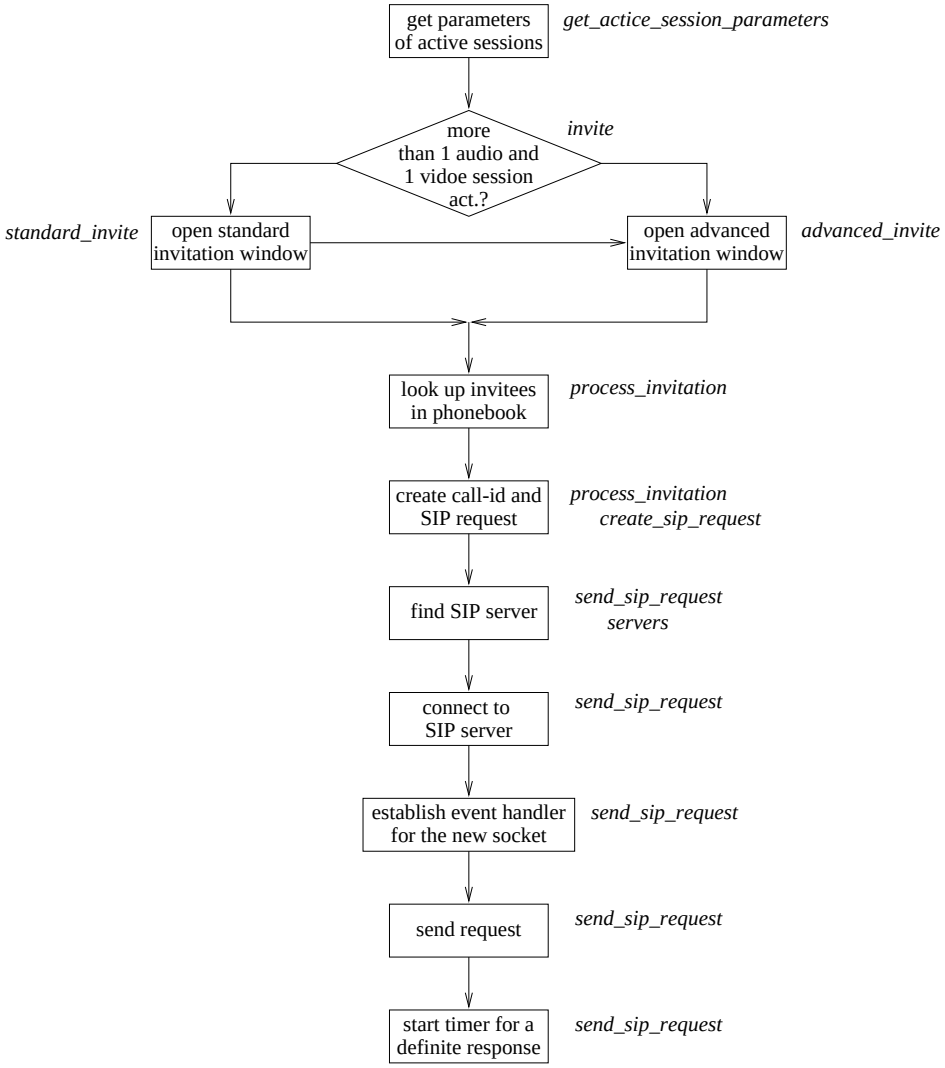


Figure 7.4: Initiating a Session Invitation

specify the appropriate parameters. To specify a more complex conference with a larger number of media streams, the advanced invitation window must be used which enables to specify an unlimited number of sessions.

After modifying the selected parameters needed to perform an invitation and pressing the *"Send Invitation"* button, the *process_invitation* procedure handles creation and sending of the SIP request. First, the procedure checks if an alias for any specified invitee exists in the SIP phonebook ($HOME/.sip/phonebook) and replaces the specified name of the callee, stored in `invite(guest)` by the name(s) in phonebook.

Afterwards, a SIP request has to be created for each invitee with a globally unique call-id. Request creation is managed by the *create_sip_request* procedure. All settings of the created invitation are stored in the `request($call_id,...)` array where the first field of the array specifies the call-id of the SIP request. The settings must be stored to allow modification of the request to enable session creation after receiving a response

which indicates the success of the invitation.

Before the request can be sent to an appropriate SIP server like sipd, isc has to extract an appropriate SIP server host for the specified invitee. Therefore, the *send_sip_request* procedure examines the domain part of the callee's address and invokes the *servers* function, offered by *libservers*, which returns a list of SIP servers. isc tries to connect to the well known SIP port at the first SIP server via TCP, and sends the request. If connection establishment or request sending failes, the next server in the list is queried. To enable isc to receive SIP responses on the new created TCP file descriptor, an event handler is created which invokes the *read_sip_response* procedure if the socket to the server becomes readable.

Finally, a timer, which invokes the *sip_timeout* procedure if it expires is established after sending the request to guarantee that isc doesn't wait an unlimited amount of time for a final response. The timer is canceled if a final response is received by isc.

7.3.2 Receiving SIP Responses

As mentioned above, initiating of SIP requests is only one task of a SIP client. An additional goal is handling of SIP responses which are caused by invitations. The process which handles incoming SIP responses is started by the *read_sip_response* procedure. It is invoked if the TCP socket to which the SIP request was sent becomes readable, which indicates an incoming response from the contacted SIP server. Figure 7.5 gives an overview of the functions initiated by in incoming SIP reply.

After reading the response (procedure *read_sip_response*), the *process_sip_response* procedure which coordinates response handling, calls the *extract_sip_parameters* function which reads the specified settings in the response and stores them in the `response ($call_id,...)` array. The first field of the array specifies the call-id of the SIP message.

If the received response is a final response, the `sip_timeout` timer is canceled and the TCP socket to the server is closed because process of the invitation is finished and no more responses for this call-id are expected.

Since the calling user has to be informed about the response received, an appropriate window is created which notifies the inviter about the information given in the SIP reply. Depending on the response code, different action are invoked next: If the invitation was successful, the sessions the callee was invited to are created automatically by the *sip_open_sessions* procedure. Otherwise, the invitation failed but some modifications in the invitation could result in a successful call. This is indicated by a redirect or *"Not Accaptable"* response which offers some alternative values for a new modified invitation. To make request modification easy, the caller has the ablilty to press the *"Modify Request"* button in the appropriate window which invokes the *modify_sip_request* procedure. If doing so, the original selected invitation values are extracted out of the `request($call_id,...)` array and are stored in the `invite` array (procedure *get_original_invite_settings*). Afterwards, the appropriate invitation settings are replaced by the alternative values specified in the received SIP response. These settings are stored in the `response($call_id,...)` array. Finally, the called *invite*

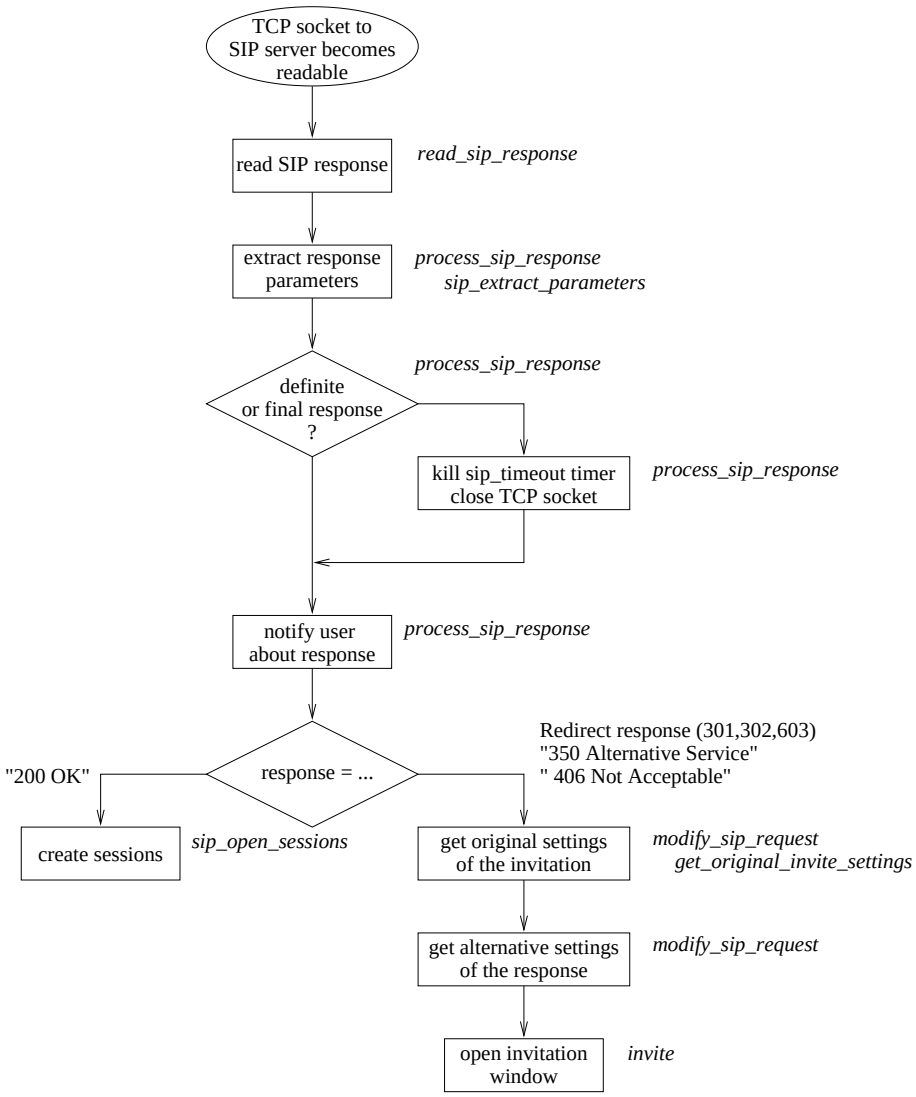


Figure 7.5: Handling of an Incoming SIP Response

procedure invokes the standard or advanced invitation window where the alternative values are selected automatically. The calling user has now the ability the send the modified request directly or to do some further modifications.

7.4 SIP Server Enhancements

The SIP server enhancements of isc deal with the server's user interface component. As mentioned in Section 5.4, the invitation daemon sipd forwards a SIP request to isc if the invited user is located at the local host. Therefore, sipd starts the Integrated Session Controller if isc is not already running.

The *control_read_udp* procedure which is invoked if the pmm socket becomes readable starts processing incoming SIP requests if a SIP message indicated by a SIP pmm

header is received. Figure 7.6 shows the typical handling of new SIP requests by isc.

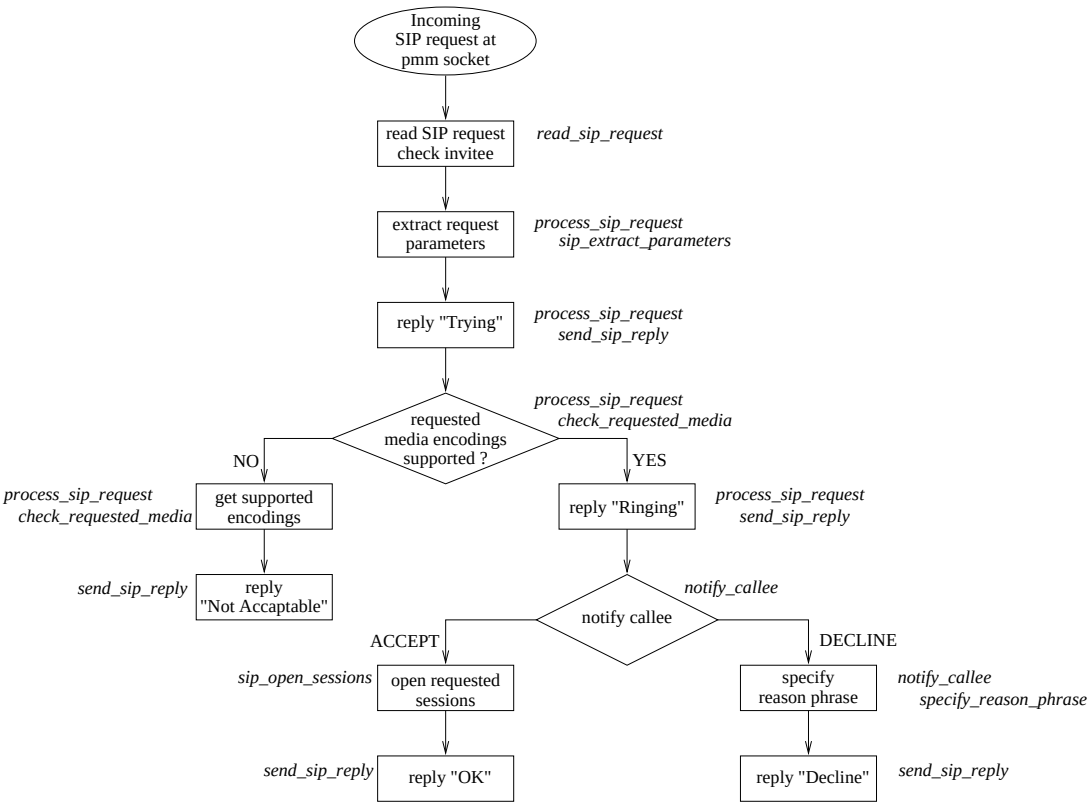


Figure 7.6: Receiving a SIP Request

Initially, the *read_sip_request* procedure reads the received SIP message and checks the request's method and call-id. The *process_sip_request* procedure which is called next manages the further processing of the SIP call. Therefore, the *sip_extract_parameters* procedure extracts the specified invitation and session parameters and stores them in the `request(call_id,...)` array. Besides, the task of this procedure is to check if the invitation specifies the same conference as the one which is currently handled by isc. This must be done because the Integrated Session Controller only can handle one single conference. If the specified `session_name` in the SDP payload is not equal to the `conf(name)` variable, this invitation must be handled by another copy of isc. So, the SIP message is ignored which will cause sipd to start a new copy of isc.

Otherwise, the *process_sip_request* procedure initiates a *"Trying"* response to sipd to indicate that the request was received and is handled by isc. As mentioned in Section 5.4, the *"Trying"* response causes sipd to cancel its *isc_timeout* or *start_isc_timeout* timer.

After reading all parameters of the SIP call, isc checks if the requested media types and their encodings are supported. The *check_requested_media* procedure searches for the media encodings in the user's $HOME/.mailcap file which contains a list of supported encodings. If any of the requested encoding isn't supported, an appropriate *"Not Acceptable"* response is created which also deals with a list of all supported encodings

for the requested media types. Otherwise, if all encodings are accepted, the callee is notified about the incoming request and a *"Ringing"* response is sent to the calling client to indicate the current status of the invitation. To avoid that the notify window is present an unlimited amount of time if the user is not sitting in front of this workstation, a timer is started which will invoke the *notify_timeout* procedure if the invitee doesn't react to the call.

In the notify window which shows the invitee the request's sender and the requested media types, the user has the possibility to accept or decline the call. Both actions initiate the belongingSIP response to the client. If the call is accepted, the *open_sip_sessions* procedure starts the invited sessions automatically.

7.5 Additional Enhancements

This section deals with a short description of additional enhancements which are not required to enable SIP and SDP functionality, but makes the usage of session invitation more comfortable.

7.5.1 The SIP Phonebook

To invite other users to multimedia sessions, the inviter has to address each callee by his SIP address which was described in detail in Section 2.3. Since it is expected that a caller invites several users frequently, it should be possible to store short aliases for these often called users. This should make the usage of isc easier because the caller only has to type the known alias to specify the invitee and not the whole SIP address.

Additionally, aliases could be used to store groups of users, identified by their SIP address, which makes sense when a caller wants to invite several users to the same miltimedia session(s). The inviter only has to specity the name of the group alias and isc sends a SIP request to each participant of the group.

To store the aliases, isc uses the $HOME/.sip/phonebook file which lines are formated as *"aliases: address1 address2 address3 ..."*. Before creating and sending a SIP request, isc looks up if the specified invitee(s) could be found in the phonebook file. If so, the invitee's alias is replaced by the list of addresses given in the file.

Since it seems not convenient to edit the phonebook file manually, isc offers a simple phonebook editor. It could be used to add, remove and modify aliases for invitees. Usage of the phonebook editor is given in the instruction manual in Appendix B.2.3.4.

7.5.2 The SIP Handler Editor

As described in Section 5.2, the Session Invitation Daemon (sipd) supports a mechanism to reply to incoming SIP calls automatically with respect to the call's subject or sender. To specify the appropriate SIP handlers, sipd uses the $HOME/.sip/sip_handler file. As for the phonebook file, a comfortable way to modify the stored handlers is needed. Therefore, isc deals with a simple SIP handler editor which could be used to install or

remove handlers. Thereby, the editor gives the user the possibility to install handlers which react to each incoming request or to specify handlers which only react to request's which consist of a specified subject or sender header. To response to the request, the user can choose between redirect and decline responses.

7.6 Variables

isc uses three arrays to store the invitation settings and session description parameters of SIP requests and responses. This section gives an overview of the `invite`, `request` and `response` array and the fields used in the arays.

7.6.1 The `invite` Array

The `invite` array is used to store the settings specified in the standard and advanced invitation window. Table 7.1 shows a list of the most important fields in the array.

Variable	*Description*
guest	space separated list of invitees
information	subject of the conference
session_addr	unicast or multicast IP address of the conference
ttl	time to live value for the multicast group specified by `session_addr`
media_list	list of the media descriptions of the conference (see 7.6.1.2)
audio|video	flag which indicates whether the *"audio"* or *"video"* check-button in the standard invitation window is selected
lpc|gsm|dvi4|...	flag which indicates whether the beloging encoding check-button in the media configure or advanced invitation window is selected
audio|video, information	information of the media stream
audio|video, session_addr	unicast or multicast IP address of the belonging media stream
audio|video,port	port of the belonging media stream
audio|video,ttl	time-to-live value for the multicast group specified by `audio|video,session_addr`
audio|video,code_list	list of the selected encodings of the media stream (see 7.6.1.1)
audio|video, profile_list	list of the RTP audio video profiles of the belonging `code_list` (see 7.6.1.1)

Table 7.1: Fields of the `invite` Array

7.6.1.1 The `code_list` and `profile_list` Field

The Session Description Protocol allows to specify different alternative media encodings within a media description whereas ordering of the encodings specifies their priority. So, isc has to enable the user to select several media encodings for each media stream description in an invitation. Therefore, the configure windows of the appropriate media type can be used. They allow the user to specify several encodings and also shows the user the order of their priority. The list of the specified encodings is stored in the appropriate `code_list` field of the invite array.

The `profile_list` field is also a list which stores the selected encodings, but instead of storing their names like `code_list`, it consists of the media payload type as defined in the RTP Audio/Video Profile (RTP/AVP) [13]. This information is needed by SDP because it uses the RTP/AVP media payload type to specify the requested media encodings in a session description.

Manipulation between encoding name and RTP/AVP payload type is done by the *get_profile* and *get_coding* procedures.

7.6.1.2 The `media_list` Field

The `invite(media_list)` variable consists of a list which stores parameters of all media streams (sessions) specified for the invitation. Each element of the list is built of 6 parts which are sperated by a colon character:

- *media:port:session_addr:ttl:code_list:information*
- e.g. audio:3456:224.2.0.1:16:pcmu lpc gsm:german audio stream

In the standard invitation window, the `media_list` is created after pressing the *"Send invitation"* button by the *create_media_list* procedure. The advanced invitation window modifies the `media_list` when the inviter adds or removes media streams to the invitation by pressing the appropriate *"Add"* (procedure *advanced_media_add*) or *"Remove"* (procedure *advanced_media_remove*) button.

7.6.2 The `request` and `response` Array

An overview of most of the fields used by the `request` and `response` array is given in Table 7.2. Both arrays are used to store parameters of incoming or outgoing SIP messages. To distinguish between different invitations, the first field of the `request` and `response` array specifies the call-id of the SIP call. When acting as a SIP client, isc stores the settings of outgoing requests in `request(call-id,...)`, the `response(call-id,...)` array handles incoming responses. On the other hand, parameters of incoming requests at isc acting as server, are also stored in `request(call-id,.)`.

Variable	*Description*
via	list of the SIP message's *"Via"* headers
from	user specified in the *"From"* header
to	user specified in the *"To"* header
length	payload length
host	host specified in the *"Contact-Host"* header
location	list of SIP locations specified in the *"Location"* header
reason	list of reasons specified in the *"Reason"* header
payload	payload of the received message
session_addr	uni- or multicast IP address of the conference
ttl	time to live value for the multicast group specified by `session_addr`
information	subject of the conference
media_list	list of the media descriptions of the conference (see 7.6.1.2)
session_name	name of the conference
sessions_to_create	list of media stream descriptions (formated as in `media_list`) which sould be created by isc

Table 7.2: Fields of the `request` and `response` Array

7.7 Procedures

This section deals with an overview and a short description of the procedures implemented to enhance the intergrated session controller with SIP and SDP functionality. The source code is separated into several files to get a better overall view. Table 7.3 gives a short list of the files.

7.7.1 The *rtp_avp.tcl* File

7.7.1.1 The *rtp_avp_load* Procedure

The *rtp_avp_load* procedure deals with a list which enables a matching between the name of an audio or video encoding and its RTP profile payload type or vice versa. Each list entry consist of two parameters. The first one specifies the name of the encoding (followed by channel count and sample rate for audio encodings), the second one gives the corresponding RTP profile payload type. The list is stored in the `rtp_avp_list` variable.

7.7.1.2 The *get_code* Procedure

The *get_code* procedure tries to match a RTP profile payload type, given as argument to the procedure, to the encoding name based on the `rtp_avp_list` offered by *rtp_avp_load*. The procedure returns the encoding name (followed by channel count and sample rate for audio encodings) if a match occurs.

File	*Description*
rtp_avp.tcl	helping procedures to change a media encoding to the RTP audio/video profile and vice versa
sip_media_list.tcl	helping procedures which handle the `media_list`
invite.tcl	procedures to initiate a SIP call
std_invite.tcl	procedures which handle the standard invitation window
adv_invite.tcl	procedures which handle the advanced invitation window
sip.tcl	procedures to create requested sessions and extract parameters of incoming SIP messages
sip_request.tcl	procedures to handle incoming SIP requests
sip_response.tcl	procedures to handle incoming SIP responses
sip_file.tcl	procedures to handle file operations for the sip_handler and phonebook functions
sip_handler.tcl	simple editor for the automatic reply function
sip_phonebook.tcl	simple phonebook editor

Table 7.3: isc Files Implementing SIP and SDP Functionality

7.7.1.3 The *get_profile* Procedure

The *get_profile* procedure tries to match an encoding name, given as argument to the procedure, to the RTP profile payload type based on the `rtp_avp_list` offered by *rtp_avp_load*. The procedure returns the profile if a match occurs.

7.7.2 The *sip_media_list.tcl* File

7.7.2.1 The *media_checkbutton_list* Procedure

The *media_checkbutton_list* procedure creates a checkbutton list which allows to select several list items. A line, located below the checkbutton list, shows the slected items in order of their choice. Encodings which are given to the procedure, but are not supported due to the specification in the .mailcap file, could not be selected by users but are disabled. The *media_checkbutton_list* is used in the media configuration or the advanced invitation window to select media encodings.

7.7.2.2 The *update_code_list* Procedure

The *update_code_list* procedure updates the encoding lists stored in the `invite($media, code_list)` and `invite($media,profile_list)` variable with respect to the specifications in the appropriate *media_checkbutton_list* window. It is invoked if a media encoding checkbutton is selected. The first argument of *update_code_list* specifies the media type *audio, video* whereas the second one gives the name of the selected encoding.

7.7.2.3 The *media_list_add* Procedure

The *media_list_add* procedure adds a new item to the `invite(media_list)` variable. The first and only argument of the procedure specifies the media type for which a new entry should be added to the list. The new list entry is composed out of the appropriate `invite` settings. On failure, the procedure shows an appropriate message to the user and returns with *1*, otherwise zero is returned.

7.7.2.4 The *media_list2invite* Procedure

The *media_list2invite* procedure together with the *media2invite* procedure converts the settings of the media descriptions stored in the `invite(media_list)` variable to an `invite` array. It is used to display the settings, stored in `invite(media_list)` in the invitation windows.

7.7.2.5 The *media2invite* Procedure

The *media2invite* procedure converts the settings of one media description stored in the `invite(media_list)` variable to an `invite` array.

7.7.3 The *invite.tcl* File

7.7.3.1 The *invite* Procedure

The *invite* procedure checks whether to call the standard or advanced invitation window with respect of the settings stored in `invite(media_list)`. Since the standard invitation window can only handle one audio and/or one video session, the advanced invitation window is invoked if more sessions are specified in `invite(media_list)`. Before invoking *standard_invite* or *advanced_invite*, the *media_list2invite* procedure is called to automatically display the appropriate settings in the invitation window.

7.7.3.2 The *get_active_session_parameters* Procedure

The *get_active_session_parameters* procedure is invoked if the invite button is pressed. It checks if any sessions are currently in process by isc. If so, the appropriate session settings are stored in `invite(media_list)`. Afterwards the *invite* procedure is invoked.

7.7.3.3 The *process_invitation* Procedure

The *process_invitation* procedure manages creation and sending of SIP requests. It is invoked if the *"Send invitation"* button of an invitation window is pressed. First, it checks if any of the specified invitees in `invite(guest)` is specified in the phonebook file. If so, the callee is replaced by the unaliased address(es) of the phonebook. Afterwards, *process_invitation* creates a call-id for each invitee and calls the *create_sip_request*

procedure. The finally invoked *send_sip_request* procedure transmits the request to the appropriate SIP server.

7.7.3.4 The *create_sip_request* Procedure

The *create_sip_request* procedure is invoked by the *process_invitation* procedure to generate a SIP request with respect to the `invite` settings. The two arguments of the procedure are the invitee and the globally unique call-id of the request. *create_sip_request* stores all parameters of the generated request in the `request($call_id,...)` array. If creation of the request was successful zero, otherwise *1* is returned.

7.7.3.5 The *send_sip_request* Procedure

The *send_sip_request* procedure is invoked by the *process_invitation* procedure to send a SIP request to an appropriate SIP server. The only argument to *send_sip_request* specifies the call-id of the request which also specifies the belonging `request($call_id,...)` array.

First, the procedure invokes the *servers* procedure which returns a list of SIP servers to contact. *send_sip_request* tries to establish a TCP connection to the first server and tries to send the request, stored in `request($call_id)`. If connection establishment or request sending failes, the next server in the list is queried.

To enable that isc will receive a SIP response on the connected socket, an event handler is established which will invoke the *read_sip_response* procedure if the file descriptor becomes readable.

Finally, a timer is created which calls *sip_timeout* if no definite response was received during a timeout interval of one minute.

7.7.3.6 The *sip_timeout* Procedure

The *sip_timeout* procedure, with call-id as argument, is invoked if no definite response for in invitation was received during a timeout interval. The procedure notifies the caller about the timout and terminates the active invitation, specified by its call-id.

7.7.4 The *std_invite.tcl* File

7.7.4.1 The *standard_invite* Procedure

The *standard_invite* procedure offers the inviter a graphical user interface to specify the invitees and session parameters of the invitation. The standard invitation window shown in Figure 7.7 lets the user invite to an audio and/or video session, since it is expected that mostly a caller only uses these two sessions within a conference. If an invitation for more sessions should be sent, the advanced invitation window (Figure 7.9), which is invoked if the *"Advanced Session Configuration"* button is pressed, must be used.

Figure 7.7: The Standard Invitation Window

Configuration of the requested media streams and their encodings could be done by calling the media configuration window by pressing the appropriate *"Configure"* button(s). If doing so, the *standard_media_config* procedure is invoked and opens the audio or video configuration window (Figure 7.8).

To create and send a SIP request message, the *"Send invitation"* button has to be pressed to invoke the *create_media_list* procedure.

7.7.4.2 The *standard_inv_display* Procedure

The *standard_inv_display* procedure updates the display of the standard invitation window (Figure 7.7). It is invoked if the audio or video checkbutton in the invitation window is pressed.

7.7.4.3 The *standard_media_config* Procedure

The *standard_media_config* procedure is invoked if one of the *"Configure"* buttons in the standard invitation window (Figure 7.7) is pressed. It opens a media configure window (shown in Figure 7.8) which lets the user to specify parameters of the appropriate media stream.

Besides, the caller has the ability to choose the requested media encodings for the session. The belonging list of media encoding is created by the *media_checkbutton_list* procedure.

7.7.4.4 The *create_media_list* Procedure

The *create_media_list* procedure is invoked if the *"Send invitation"* button in the standard invitation window is pressed. It is used to store the specified invitation settings in the `invite(media_list)` variable. Afterwards the *process_invitation* procedure is invoked to continue with creation and sending of the SIP request.

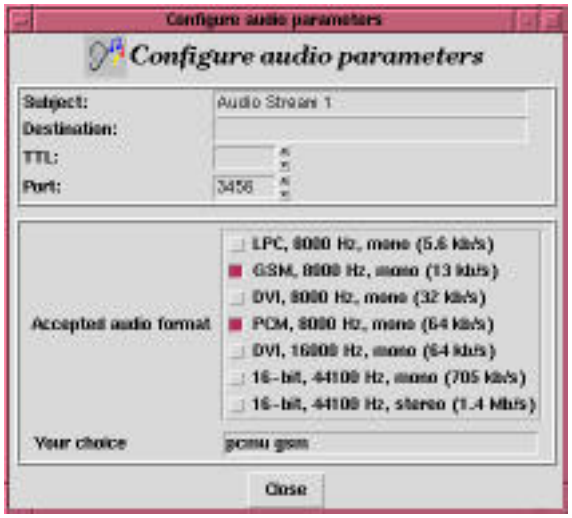


Figure 7.8: The Audio Configuration Window

7.7.5 The *adv_invite.tcl* File

7.7.5.1 The *advanced_invite* Procedure

The *advanced_invite* procedure creates the andvanced invitation window which is shown in Figure 7.9. It is invoked by the *invite* procedure of if the *"Advanced Session Configuration"* button in the standard invitation window was pressed. It gives the caller the ability to specify a large number of sessions within a SIP request. All settings could be specified within one single window, no additional configure window must be used.

Pressing the *"Send invitation"* button will start to create and send the SIP request by invoking the *process_invitation* procedure.

7.7.5.2 The *andvanced_inv_display* Procedure

The *andvanced_inv_display* procedure updates the display of the advanced invitation window. It is invoked if one of the displayed media descriptions is selected.

7.7.5.3 The *advanced_media_add* Procedure

The *advanced_media_add* procedure adds a new media description to the `invite(media_list)` variable by invoking the *media_list_add* procedure and updates the displayed media list items in the advanced invitation window. It is invoked if one of the *"Add"* buttons in the advanced invitation window was pressed.

7.7.5.4 The *advanced_media_remove* Procedure

The *advanced_media_remove* procedure removes the media description, selected in the advanced invitation window, out of the list stored in the `invite(media_list)` variable.

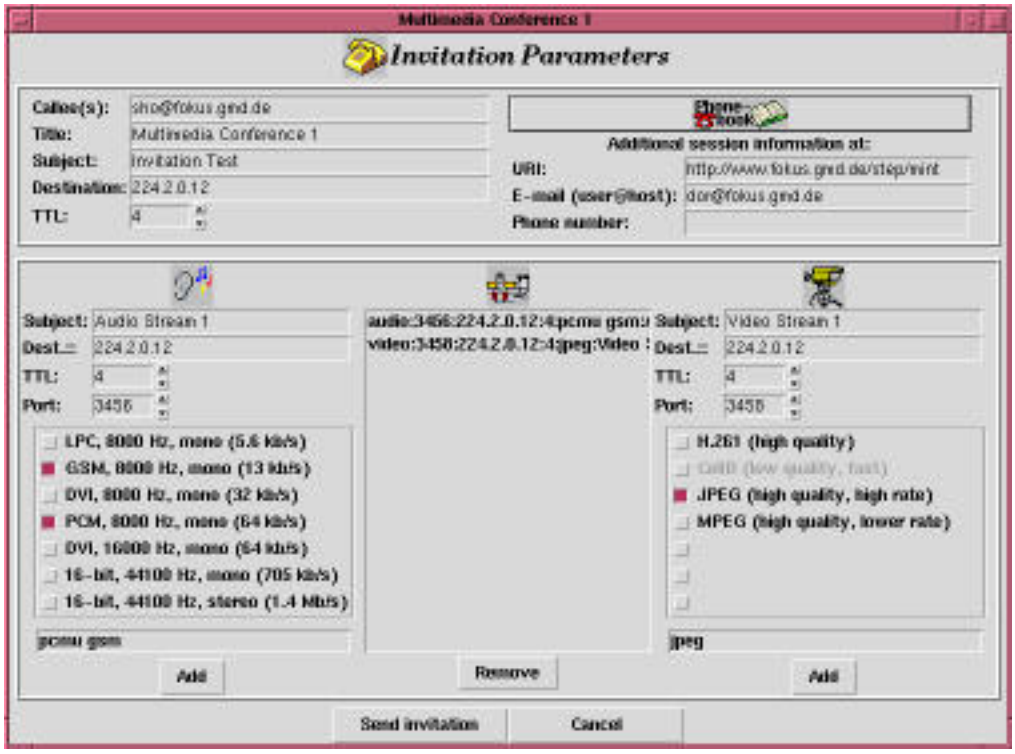


Figure 7.9: The Advanced Invitation Window

Afterwards, it updates the displayed media list items in the advanced invitation window.

7.7.6 The *sip.tcl* File

7.7.6.1 The *sip_extract_parameters* Procedure

The *sip_extract_parameters* procedure is used to extract the SIP and SDP parameters specified in an incoming request or response message. It is invoked by the *process_response* or *process_request* procedure. The extracted parameters are stored in the `request($call_id,...)` or `response($call_id,...)` array.

If an incoming SIP request is examined, *sip_extract_parameters* also checks if the requested sessions belong to the conference which is currently handled by isc. Therefore, isc checks if the session name specified in the SDP payload is equal to the name of the active conference (stored in `conf(name)`). If the request deals with information about a new conference, the *sip_extract_parameters* procedure returns with a value of one, which lets isc ignore the SIP message. Otherwise zero is returned.

7.7.6.2 The *sip_open_sessions* Procedure

The *sip_open_sessions* procedure is invoked to create sessions requested by an incoming and accepted SIP request or if an invitation was successful. The `request(sessions_to_create)` variable stores a list of call-ids of received or sent invitations for which sessions

should be created. The description of the media streams are stored in the belonging `request($call_id,media_list)` variable.

The *sip_open_sessions* procedure opens all sessions belonging to the first call-id in `request(sessions_to_create)`. Depending on isc acting as SIP client or server and if a unicast or multicast address was specified in the media description, isc has to handle the specified session address differently. If a caller wants to invite a remote user to an unicast session, he normally specifies his own host as session address. The session controller at the server side opens the media stream to the specified address in the SDP payload. On the other side, isc acting as client must not create the session to his IP address but to the address of the replying server host.

Therefore, the *sip_open_sessions* procedure checks if the session address in the appropriate media description is equal to the local host address. If so, isc creates the specified session not to his own IP address but to the host specified by the *"Contact-Host"* header of the received SIP response (stored in `response($call_id,host)`).

If the session address is a multicast adderess, isc opens the session to this address, independent of acting as a SIP cilent or server.

After creating the sessions for each media description stored in the `request($call_id,media_list)` variable, the first call-id in the `request(sessions_to_create)` variable is removed and *sip_open_sessions* checks if there are aditional call-ids in the `request(sessions_to_create)` list. If so, the *sip_open_sessions* procedure is invoked again to create the sessions of the next successful call.

This is method is choosen, because the *session_create* procedure which is invoked by *sip_open_sessions* only can process one session creation after the other since it uses the same variables to open new media streams. Since isc is bulit on an event driven basis, a mechanism is required which enables to receive sveral requests or responses concurrently but also to open the specified sessions one after another and not at the same time.

7.7.6.3 The *sip_sound* Procedure

The *sip_sound* procedure is used to play short audio files to notify the user about incoming SIP requests or about newly received SIP responses.

7.7.7 The *sip_request.tcl* File

7.7.7.1 The *read_sip_response* Procedure

isc invokes the *read_sip_response* procedure is invoked if it receives a SIP response. The procedure reads the response and extracts the reply code and reason phrase and stores them in the appropriate `response($call_id,...)` array. Afterwards, the *process_sip_response* procedure is invoked which is the main procedure to handle SIP responses.

7.7.7.2 The *process_sip_response* Procedure

The *process_sip_response* procedure is invoked by *read_sip_response* to handle incoming SIP responses. It first calls the *sip_extract_parameters* procedure to get the values specified in the response message. These values are stored in the `response($call_id, ...)` array.

If a final response was received by isc, the timer started when sending the corresponding SIP request is cancelled. Additionally, the TCP connection to the remote SIP server is disconnected since the request is finished.

The *process_sip_response* procedure notifies the caller about the received response and handles differently due to its reply code:

If the invitation was successful (indicated by a *"200"* reply code), the call-id is added to the `request(sessions_to_create)` list which will indicate to open all sessions specified in the `request($call_id,media_list)`. If the call-id is the only one in `request(sessions_to_create)`, the *sip_open_sessions* procedure is invoked. Otherwise the *sip_open_sessions* procedure currently is creating sessions for other calls and is reinvoked until the `request(sessions_to_create)` list is empty (see 7.7.6.2).

If the response code specifies a redirect response or a response which indicates that the request should be modified and retransmitted, the *modify_sip_request* procedure is invoked after pressing the *"Modify Request"* button in the notification window.

7.7.7.3 The *get_original_invite_settings* Procedure

The *get_original_invite_settings* procedure is invoked by the *modify_sip_request* procedure to extract the parameters specified for an invitation out of the appropriate `request($call_id,...)` array and stores them in the `invite` array. The procedure is called if a response was received which indicates that some modifications to the original SIP request are required to enable a sccessful invitation.

7.7.7.4 The *modify_sip_request* Procedure

The *modify_sip_request* procedure is invoked if the *"Modify Request"* button in the notification window was pressed. It is used to extract the original invitation settings of the sent SIP request by calling *get_original_invite_settings* and to overwrite the extracted parameters by the alternative parameters given in the received response message. Afterwards, the called *invite* procedures opens the standard or advanced invitation window to let the caller modify and retransmit the invitation. The settings which are automatically loaded to the invitation window are the parameters of the original sent request, modified with the values which are given in the received response message. So, the inviter normally only has to press the *"Send invitation"* button to start the new modified request which should have a better chance of being accepted by the callee. Nevertheless, the user can make additional modification if he decides to do so.

7.7.8 The *sip_response.tcl* File

7.7.8.1 The *read_sip_request* Procedure

The *read_sip_request* procedure is invoked by the *control_read_udp* procedure if a SIP pmm messages is received via the local pmm multicast group. The procedure checks if the received message is a SIP request and if it is addressed to the owner of isc. If so, it reads the request, otherwise the message is ignored. Before calling the *process_sip_request* procedure which is the main procedure to handle incoming SIP requests, *read_sip_request* extract the call-id of the invitation.

7.7.8.2 The *process_sip_request* Procedure

The *process_sip_request* procedure is invoked by *read_sip_request* to handle new incoming SIP requests. After calling the *sip_extract_parameters* procedure which extracts the SIP and SDP parameters of the invitation and checks if the requested sessions belong to the active conference, the *process_sip_request* procedure initiates a *"Trying"* response to the calling client. Therefore, it sets the appropriate `request($call_id,reply_code)` and `request($call_id,reason_phrase)` variables and calls the *send_sip_reply* procedure. This response notifies sipd that the SIP request was received by isc. Afterwards the called *check_requested_media* procedure checks, if the media and their encodings specified in the request are supported by the invitee. If so, a *"Ringing"* response is sent to the inviter and the *notify_callee* procedure is called, otherwise an appropriate respone, which indicates a list of the supported media is sent to the SIP client.

7.7.8.3 The *check_requested_media* Procedure

The *check_requested_media* procedure is used to check if the media and their encodings, requested in the received SIP call, are supported. Therefore, the user's .mailcap file is queried since it deals with a list of the supported media types and encodings used by isc and its media agents.

If one of the media encodings or types which are specified in the media description of the SIP request are not supported by isc, a *"Not Acceptable"* response is initiated by *check_requested_media*. This is done by setting the `request($call_id,reply_code)` and `request($call_id,reason_phrase)` variables appropriately. Moreover, the `request ($call_id,media_list)` variable is modified and now stores the description of the supported media types and encodings. These will be listed in the response message to indicate the caller how to modify the invitation to make the call successful. The procedure returns with one which notifies the calling *process_sip_request* procedure that the request can't be handled successfully.

If the media descriptions are accepted, the procedure returns with zero.

7.7.8.4 The *ring* Procedure

The *ring* procedure is used to notify the called user about the incoming invitation. Therefore, it calles the *sip_sound* procedure periodically until the callee reacts to the request.

7.7.8.5 The *notify_callee* Procedure

The *notify_callee* procedure which is invoked by *process_sip_request* notifies the callee about an incoming invitation. This is done by opening an appropriate notify window and calling the *ring* procedure. In the *"Incoming Call"* window, shown in Figure 7.10, different parameters like the sender of the invitation, the conference name and the requested media streams are presented to the callee. The invited user has the ability to accept or decline the call. After pressing the *"Accept"* button, the appropriate `request($call_id,reply_code)` and `request($call_id,reason_phrase)` variables are set and the *send_sip_reply* procedure is invoked which sends the response to the SIP client and initiates the creation of the requested sessions.

Figure 7.10: The "Incoming Call" Window

The *"Decline"* button invokes the *specify_reason_phrase* procedure to enable the user to specify a reason, why he wants to decline the call.

Since it is possible that a user is logged in at a workstation but is absent and can't recognize the incoming invitation, the notify window must be deleted after a defined interval of time. Therefore the *notify_callee* procedure initiates a timer which calls the *notify_timeout* procedure if neither the *"Accept"* button nore the *"Decline"* button was pressed during the timeout interval of 1 minute.

7.7.8.6 The *notify_timeout* Procedure

The *notify_timeout* procedure is invoked if the callee doesn't react to the call notification during a timeout interval. The procedure initiates a *"Not Currently Here"* reply.

7.7.8.7 The *specify_reason_phrase* Procedure

The *specify_reason_phrase* procedure is invoked if the *"Decline"* button in the notification window was pressed. It enables the callee to specify a reason why he declines the call. When pressing the new created *"Send"* button, the *send_sip_reply* procedure is invoked and sends the response to the initiation SIP client.

7.7.8.8 The *show_additional_info* Procedure

The *show_additional_info* procedure which is invoked if the *"Additional Info"* button in the notification window was pressed opens a new window which displays how the callee can get additional information about the session.

7.7.8.9 The *send_sip_reply* Procedure

The *send_sip_reply* procedure which is invoked whenever a SIP response to the calling SIP client should be sent, creates a SIP response message due to the reply code specified in the `request($call_id,reply_code)` variable. Moreover, the procedure calls the *sip_open_sessions* procedure if the response which should be sent indicates a successful, accepted invitation. If the callee doesn't neither accept nor declines the incoming call, the *send_sip_reply* procedure terminates isc, if the session controller was started by the SIP daemon sipd.

7.7.9 The *sip_file.tcl* File

7.7.9.1 The *save_check* Procedure

The *save_check* procedure is invoked if the user has modified settings in the phonebook or sip_handler window and has pressed the *"Dismiss"* button. The procedure creates a window which enables the user to save or dismiss the settings.

7.7.9.2 The *save_file* Procedure

The *save_file* procedure is used to save the settings in the phonebook or sip_handler file.

7.7.9.3 The *read_file* Procedure

The *read_file* procedure reads the settings of the phonebook or sip_handler file. It returns the list of the read lines.

7.7.10 The *sip_handler.tcl* File

7.7.10.1 The *sip_handler* Procedure

The *sip_handler* procedure offers a simple editor for the $HOME/.sip/sip_handler file which is used by sipd to reply automatically to incoming SIP calls. The format of the file is described in Section 5.2. The sip_handler window (given in Figure 7.11) shows the currently specified handlers. To add or remove handlers the appropriate button has to be pressed.

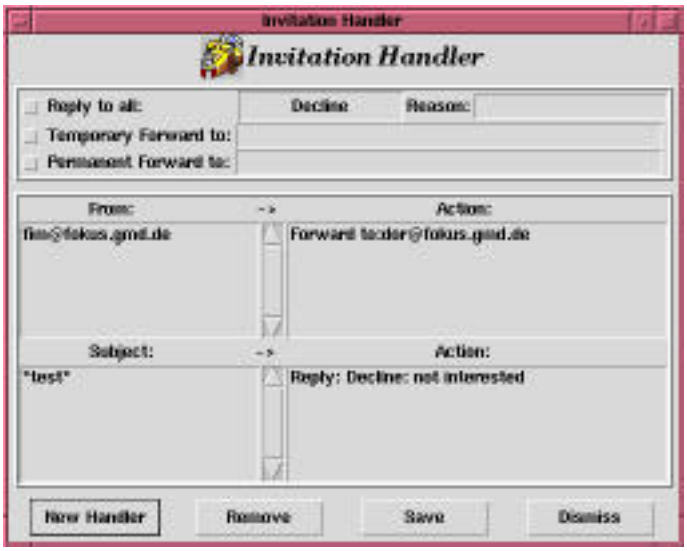


Figure 7.11: The sip_handler Window

7.7.10.2 The *all_display* Procedure

The *all_display* procedure updates the display of the sip_handler window. It is invoked if one of the checkbuttons of the handlers for all incoming requests is selected.

7.7.10.3 The *new_handler* Procedure

The *new_handler* procedure offers a window (shown in Figure 7.12) to specify new sip_handlers. The user has to specify whether to create a handler which examines the incoming call's subject or sender. Pressing the *"Add"* button calls the *add_handler* procedure.

7.7.10.4 The *add_handler* Procedure

The *add_handler* procedure adds a new sip_handler to the sip_handler list. To update the display in the sip_handler window, the *insert_handler* procedure is invoked.

Figure 7.12: The Add sip_handler Window

7.7.10.5 The *remove_handler* Procedure

The *remove_handler* procedure, invoked after pressing the *"Remove"* button in the sip_handler window, removes the selected handler out of the handler list and updates the display.

7.7.10.6 The *insert_handler* Procedure

The *insert_handler* procedure invoked by *add_handler* updates the display in the sip_handler window after adding a new handler.

7.7.10.7 The *save_sip_handler* Procedure

The *save_sip_handler* procedure is invoked if the *"Save"* button of the handler window was pressed and manages the saving in the sip_handler file. Therefore it invokes the *save_file* procedure.

7.7.11 The *sip_phonebook.tcl* File

7.7.11.1 The *sip_phonebook* Procedure

The *sip_phonebook* procedure offers the user a simple editor (shown in Figure 7.13) to modify aliases of invitees stored in the $HOME/.sip/phonebook file. Pressing the appropriate buttons will call the procedures to add or remove aliases and to save the phonebook file.

7.7.11.2 The *alias_selection* Procedure

The *alias_selection* procedure is invoked if one alias in the alias list is selected. It updates the display of the addresses frame which shows the stored addresses for the selected alias.

7.7.11.3 The *add_alias* Procedure

The *add_alias* procedure adds a new alias to the phonebook and updates the display. It is invoked if the *"Add"* button under the aliases frame was pressed.

Figure 7.13: The Phonebook Window

7.7.11.4 The *remove_alias* Procedure

The *remove_alias* procedure removes the selected alias from the phonebook. It is invoked if the *"Remove"* button under the aliases frame was pressed.

7.7.11.5 The *add_addr* Procedure

The *add_alias* procedure adds a new address to the selected alias and updates the display. It is invoked if the *"Add"* button under the addresses frame was pressed.

7.7.11.6 The *remove_addr* Procedure

The *remove_alias* procedure removes the selected address from the phonebook. It is invoked if the *"Remove"* button under the aliases frame was pressed.

7.7.11.7 The *save_phonebook* Procedure

The *save_phonebook* procedure is invoked if the *"Save"* button of the phonebook window was pressed and manages the saving in the phonebook file. Therefore, it also invokes the *save_file* procedure.

Chapter 8

Summary and Future Work

The goal of this work was to implement a mechanism to enable an easy way to invite users to multimedia sessions over the internet. The Session Initiation Protocol deals with a request-response mechanism and defines the message and address format used to invite users. The features of SIP include session invitation based on the user's email address which requires a location mechanism within the user's domain.

To describe the session parameters within a SIP message, the Session Description Protocol was chosen.

The software implemented is separated into two different parts. The Session Invitation Daemon (sipd) is a per-host daemon which waits for incoming SIP requests and handles them according to the SIP specification. It performs user location by using a location service database and redirects or forwards the SIP call. Moreover, an automatic reply function is implemented to enable the user to redirect or decline invitations with respect to their subject or sender.

The Integrated Session Controller (isc), which is the second part within the implementation was enhanced with the ability to send session invitations and to act as a SIP server in combination with sipd. It offers a graphical interface to the user which makes it easy to specify invitation parameters for both novice and expert users. It offers the possibility to invite several users to the same existing or new session(s). Additionally, isc was equipped with an invitation phonebook and a simple editor to specify SIP handlers for the automatic reply feature of sipd. So, a single tool can be used to invite users to multimedia sessions, to perform these sessions and to specify handlers which are used to automatically react to incoming calls. The Integrated Session Controller together with different media agents becomes to an universal multimedia session tool.

During the implementation, a problem when using the email address to specify an invitee occurs (described in Section 2.9). This results in modifications of the SIP specification in the current sip-03-draft [8].

The tester of isc and sipd criticized that it is impossible to terminate a sent call before a definite response is received or timeout interval expires. The implemented draft of SIP doesn't provide this feature. Indeed, the current draft of the Session Initiation Protocol defines additional SIP methods which are also used to terminate sent calls.

8.1 Current Status

The currently implemented versions of sipd and isc are based on the SIP version 2, draft-02 [7] including modifications described in Section 2.9.2. These modifications are also part of the current draft-03 [8]. The *OPTIONS* method is not supported.

The Session Description Protocol is implemented in the draft-03 version. Time descriptions of sessions, session attributes and payload encryption are not implemented in the current version.

8.2 Future Work

As mentioned above, the implemented versions of sipd and isc follow a SIP draft which is currently updated by a new one. Therefore, it seems to be useful to update the applications according to the actual SIP draft. Moreover, testing of sipd and isc results in additional suggestions how to improve the invitation daemon and the Integrated Session Controller:

sipd:

- Update sipd to the current SDP and SIP drafts
- Enable to configure the Session Invitation Daemon by a configuration file which allows to specify:
  - file location for call logging
  - file location for error logging
  - port on which sipd listens for incoming requests
  - supported protocols (TCP or UDP)
  - operating mode of sipd (redirect or proxy mode)
  - timeout intervals
- Ability to call sipd from inetd which makes operation more robust

isc:

- Update sipd to the current SDP and SIP drafts
- Redesign the invitation windows to the new isc design based on TIX
- Extract the SIP handler editor to a separate application and make it more comfortable
- Add new features to the notification window like call forwarding/redirection and default decline reasons

Bibliography

[1] Douglas Comer and David L. Stevens. *Internetworking with TCP/IP – Client-Server Programming and Applications*, volume 3. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[2] Peter A. Darnell and Philip E. Margolis. *C: A Software Engineering Approach.* Springer-Verlag, 1991.

[3] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext transfer protocol – HTTP/1.1. Request for Comments (Proposed Standard) RFC 2068, Internet Engineering Task Force, January 1997.

[4] A. Gulbrandsen and P. Vixie. A DNS RR for specifying the location of services (DNS SRV). Request for Comments (Experimental) RFC 2052, Internet Engineering Task Force, October 1996.

[5] M. Handley, J. Crowcroft, C. Bormann, and J. Ott. The internet multimedia conferencing architecture. Internet Draft, Internet Engineering Task Force, July 1997. Work in progress.

[6] Mark Handley and Van Jacobson. SDP: Session description protocol. Internet Draft, Internet Engineering Task Force, March 1997. Work in progress.

[7] Mark Handley, Henning Schulzrinne, and Eve Schooler. SIP: session initiation protocol. Internet Draft, Internet Engineering Task Force, March 1997. Work in progress.

[8] Mark Handley, Henning Schulzrinne, and Eve Schooler. SIP: Session initiation protocol. Internet Draft, Internet Engineering Task Force, July 1997. Work in progress.

[9] Brian W. Kernighan and Dennis M. Ritchie. *The C Programming Language.* Prentice-Hall, Englewood Cliffs, New Jersey, 1978.

[10] Steve McCanne and Van Jacobson. vic: A flexible framework for packet video. In *Proc. of ACM Multimedia '95*, November 1995.

[11] Frank Oertel. Aufbau und Konfiguration lokaler Multimedia Konferenz-Umgebungen (Set-up and configuration of local multimedia conferencing environments). Studienarbeit, Department of Communication Networks, TU Berlin, Berlin, Germany, November 1995.

[12] John K. Ousterhout. *Tcl and the Tk Toolkit.* Addison-Wesley, Reading, Massachusetts, 1994.

[13] H. Schulzrinne. RTP profile for audio and video conferences with minimal control. Request for Comments (Proposed Standard) RFC 1890, Internet Engineering Task Force, January 1996.

[14] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson. RTP: a transport protocol for real-time applications. Request for Comments (Proposed Standard) RFC 1889, Internet Engineering Task Force, January 1996.

[15] Henning Schulzrinne. Voice communication across the Internet: A network voice terminal. Technical Report TR 92-50, Dept. of Computer Science, University of Massachusetts, Amherst, Massachusetts, July 1992.

[16] Henning Schulzrinne. Dynamic configuration of conferencing applications using pattern-matching multicast. In *Proc. International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*, Lecture Notes in Computer Science (LNCS), pages 231–242, Durham, New Hampshire, April 1995. Springer.

[17] Henning Schulzrinne. Dynamic configuration of conferencing applications using pattern-matching multicast. *Multimedia Systems*, 2, March 1996.

[18] Henning Schulzrinne. Simple conference invitation protocol. Internet Draft, Internet Engineering Task Force, February 1996. Work in progress.

[19] Henning Schulzrinne. SIP server architecture. July 1997.

[20] Dorgham Sisalem and Henning Schulzrinne. The multimedia internet terminal. In *accepted for publication in the Special I issue on Multimedia of the Journal of Telecommunication Systems*, June 1997.

[21] Dorgham Sisalem, Henning Schulzrinne, and Christian Sieckmeyer. The network video terminal. In *HPDC Focus Workshop on Multimedia and Collaborative Environments (Fifth IEEE International Symposium on High Performance Distributed Computing)*, Syracuse, New York, August 1996. IEEE Computer Society.

[22] W. Richard Stevens. *UNIX Network Programming.* Prentice Hall, 1990.

[23] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol. RFC 2165, Internet Engineering Task Force, June 1997.

[24] Brent Welch. *Practical Programming in Tcl and Tk.* Prentice Hall PTR, Upper Saddle River, New Jersey, 1995.

[25] Christian Zahl. Entwicklung einer Internet-Multimedia-Telekommunikations-Anlage (Development of an internet multimedia telecommunications system). Studienarbeit, Department of Communication Networks, TU Berlin, Berlin, Germany, November 1995.

Appendix A

Procedures of sipd

A.1 The *notify.c* file

Procedures in the *notify.c* file (©by AT&T Bell Laboratories) are used to establish and remove event handlers.

A.1.1 The *notify_set_input_func()* Function

Notify_func_input *notify_set_input_func* (Notify_client client, Notify_func_input func, int fd)

Arguments:

client: identifier of the event handler; argument passed to `func` if event occurs

func: name of the procedure which is invoked if `fd` becomes readable

fd: file descriptor of the event handler

Returned value:

-

Description:

The *notify_set_input_func()* function establishes an event handler for the file descriptor `fd`. The handler function `func`, with `client` and `fd` as arguments, is invoked whenever `fd` becomes readable. Calling *notify_set_input_func()* with `func` = *NOTIFY_FUNC_NULL* removes the handler.

A.1.2 The *notify_start()* Function

Notify_error *notify_start*(void)

Arguments:

-

Returned value:

-

Description:

The *notify_start()* function starts the event handler loop. It is terminated by calling *notify_stop()* .

A.1.3 The *notify_stop()* Function

Notify_error *notify_stop*(void)

Arguments:

-

Returned value:

-

Description:

The *notify_stop()* function terminates the event handler loop.

A.2 The *multimer.c* file

Procedures in the *multimer.c* file (©by the University of Southern California) are used to establish or remove timer events.

A.2.1 The *timer_set()* Function

struct timeval **timer_set*(struct timeval *interval, Notify_func func, Notify_client client, int relative)

Arguments:

***interval:** time interval

func: function to be called when time expires

client: first argument passed to the handler function `func`

relative: flag; set relative to current time

Returned value:

-

Description:

The *timer_set()* function sets a timer event for the specified client. The `client` pointer is opaque to this routine but must be unique among all clients. Each client may have only one timer pending. If the interval specified is zero, the pending timer, if any, for this client will be cancelled. Otherwise, a timer event will be created for the requested amount of time in the future, and will be inserted in chronological order into the queue of all clients' timers.

A.3 The *servers.c* File

Procedures in the *servers.c* file (written by Schulzrinne) are used to expand a domain name in a list of SIP servers.

A.3.1 The *servers()* Function

server_t *servers(char *domain)

Arguments:

***domain:** domain in which a SIP server should be found

Returned value:

list of server entries, with last entry having a host value of NULL

Description:

The *servers()* function expands a domain name into a list of suitable SIP servers, in the following order:

1. SRV records
2. MX record
3. A and RR records

A.4 The *misc.c* File

Procedures in the *misc.c* file are helping functions to manipulate and search for strings in SIP requests and responses. Partially based or taken from cd by Christian Zahl.

A.4.1 The *str_tolower()* Function

char **str_tolower*(char *s)

Arguments:

***s:** pointer to string which should be converted to lower case

Returned value:

pointer to the newly allocated lower case string

Description:

The *str_tolower()* function converts all characters in a given string to lower case. It returns the pointer to newly allocated lower case string.

A.4.2 The *str_search()* Function

char **str_search*(char *s1, char *s2)

Arguments:

***s1:** pointer to string in which `s2` should be searched

***s2:** pointer to string which should be searched in `s1`

Returned value:

- pointer to the located string in `s1`
- NULL if no match occurs

Description:

The *str_search()* function locates the first occurrence of a string `s2` in another string `s1`. It returns a pointer to the located string in `s2`, or the null pointer if no match occurs. The only difference to *strstr()* is, that the search is not case sensitive. Therefore, both strings (`s1` and `s2`) are converted to lower case (procedure *str_tolower()*) before *strstr()* is invoked.

A.4.3 The *str_cmp()* Function

int *str_cmp*(char *s1, char *s2)

Arguments:

***s1:** pointer to string which should be compared with `s2`

***s2:** pointer to string which should be compared with `s1`

Returned value:

- int < 0, if *str_tolower(s1)* $<$ *str_tolower(s2)*
- int $= 0$, if *str_tolower(s1)* $=$ *str_tolower(s2)*
- int > 0, if *str_tolower(s1)* $>$ *str_tolower(s2)*

Description:

The *str_cmp()* function compares string `s1` with string `s2` not case sensitive. Therefore, both strings are converted to lower case (procedure *str_tolower()*) before *strcmp()* is invoked. If both lower case strings are equal, zero is returned.

A.4.4 The *skip_white_spaces()* Function

char **skip_white_spaces*(char *p)

Arguments:

***p:** string in which leading white spaces should be skipped

Returned value:

pointer to the first character in sting `p` which is no white spaces

Description:

The *skip_white_spaces()* function returns a pointer to the first character in string `p` which is no white space.

A.4.5 The *get_field()* Function

char **get_field*(char *p, char *str, int len)

Arguments:

***p:** pointer to string in which next field should be extracted

***str:** pointer to string in which extracted field will be stored, allocated by caller of the function

len: length of string `str` in which extracted field will be stored

Returned value:

- pointer to the end of the extracted field in string `p`
- NULL on failure

Description:

The *get_field()* function tries to extract the next field in string `p` and stores the extracted string in `str`. Therefore, it first skips leading white spaces in string `p` (procedure *skip_white_spaces()*) and copies characters from string `p` to string `str` until the filed is terminated by *"space"*, *"tab"*, or any newline character. The string copying is also terminated if the string terminating null character *"\0"* comes up or the length of `str` exceeded. *get_field()* returns a pointer to the end of the extracted field in string `p` if the field could be extracted successfully or NULL, if field extraction failed.

A.4.6 The *get_line()* Function

char **get_line*(char *p, char *str, int len)

Arguments:

***p:** pointer to the buffer from which the next line should be extracted

***str:** pointer to string in which extracted line will be stored

len: length of string `str` in which extracted line will be stored

Returned value:

- pointer to the end of the extracted line in string `p`
- NULL on failure

Description:

The *get_line()* function extracts a string terminated by an end of line character out of given string `p`. It is used to extract a single line or SIP parameter out of a SIP message. The procedure works like *get_field()* except that string copying isn't terminated by whitespace characters.

A.4.7 The *search_SIP_header()* Function

char **search_SIP_header* (char *msg, char *header_name, char *short_name)

Arguments:

***msg:** pointer to the SIP message in which a SIP header should be searched

***header_name:** name of the SIP header

***short_name:** short name of the SIP header

Returned value:

- pointer to the SIP header value
- NULL, if header not found

Description:

The *search_SIP_header()* function searches for a SIP header specified by `header_name` or `short_name` in a given SIP message (`msg`). It returns a pointer to the found header value (to the end of the header identifier specified by `header_name` or `short_name`). If the SIP can't be found in `msg` the null pointer is returned.

A.4.8 The *delete_SIP_line()* Function

int *delete_SIP_line*(char *s)

Arguments:

***s:** pointer to a line in a SIP message header which should be deleted

Returned value:

- 0, if line was deleted successfully
- -1, if line was not removed

Description:

The *delete_SIP_line()* function removes a SIP header line, indicated by `s`, in a SIP message. Therefore, `s` has to point to the beginning of a SIP header line which should be deleted.

A.4.9 The *insert_SIP_line()* Function

void *insert_SIP_line*(char *p, char *s)

Arguments:

***p:** pointer to the place in a SIP message where a new header line `s` should be inserted

***s:** pointer to the new SIP header which should be inserted in a SIP message at place `p`

Returned value:

-

Description:

The *insert_SIP_line()* function inserts a new SIP header in a SIP message. The first argument `p` specifies a pointer to the place in the SIP message where the new header should be added. The procedure adds the given string `s` and terminates the new header by a "*CRLF* " sequence.

A.5 The *list.c* File

Procedures in the *list.c* file are used to handle both, the invitation and location list. They perform adding and removing of list entries and deal with functions to search specified list entries.

A.5.1 The *inv_list_search()* Function

inv_list_t **inv_list_search*(char *call_id)

Arguments:

***call_id:** call-id of the searched invitation list entry

Returned value:

- pointer to the found invitation list entry
- NULL, if no entry specified by `call_id` was found in the invitation list

Description:

The *inv_list_search()* function locates the `inv_list_t` structure identified by `call_id` in the invitation list. If an appropriate list entry is found, the pointer to this structure is returned, otherwise *inv_list_search()* returns the null pointer.

A.5.2 The *loc_list_search()* Function

loc_list_t **loc_list_search*(inv_list_t *inv, char *search_str)

Arguments:

***inv:** pointer to the invitation list entry in which a location list entry should be searched

***addr:** IP address in dotted decimal notation; identifier of the searched location list entry

Returned value:

- pointer to the found location list entry
- NULL, if no entry specified by `addr` was found in the location list

Description:

The *loc_list_search()* function tries to find a pointer to the `loc_list_t` structure identified by `addr` in the location list of the invitation list entry specified by `inv`. If an appropriate list entry is found, the pointer to this structure is returned, otherwise *loc_list_search()* returns the null pointer.

A.5.3 The *inv_list_add()* Function

int *inv_list_add*()

Arguments:

-

Returned value:

- 0, if a new entry was added to the invitation list
- -1, adding a new item to invitation list failed

Description:

The *inv_list_add()* function adds a new entry to the top of the invitation list. The procedure returns 0 if a new structure was added, on failure -1 is returned. The global variable `invitation_list` will point to the new inserted list entry. *inv_list_add()* is invoked after receiving a new SIP request.

A.5.4 The *loc_list_add()* Function

int *loc_list_add*(inv_list_t *inv)

Arguments:

***inv:** pointer to the entry in invitation list to which a new location list entry should be added

Returned value:

- 0, if a new entry was added to the location list
- -1, adding a new item to location list failed

Description:

The *loc_list_add()* function adds a new entry to the top of the location list of an invitation list entry specified by `inv`. The procedure returns 0 if a new structure was added, on failure -1 is returned. The `inv->location_list` variable of structure `inv` will point to the new inserted list entry. *loc_list_add()* is invoked if a new location was extracted by the location server or out of a received *"Alternative Address"* response.

A.5.5 The *inv_list_remove()* Function

void *inv_list_remove*(char *call_id))

Arguments:

***call_id:** identifier of the invitation list entry which should be removed

Returned value:

-

Description:

The *inv_list_remove()* function removes an `inv_list_t` structure specified by `call_id` out of the invitation list. The procedure is invoked if the process of a SIP request is finished either by sending a definite or final response to the requesting client, or by a timeout.

A.6 The udp.c File

Procedures in the udp.c file are used to perform communication based on UDP.

A.6.1 The *UDP_connect()* Function

int *UDP_connect*(char *address, int port, u_int8 ttl)

Arguments:

***address:** pointer to a uni- or multicast IP address in dotted decimal notation to which the new socket should be bound

port: port number of the new socket

ttl: ttl value if a multicast group was specified by `address`

Returned value:

- new allocated UDP socket, if socket allocation and binding was successful
- -1, if socket creation failed

Description:

The *UDP_connect()* function allocates and binds a new socket to a given IP address and port. The IP address can either be an uni- or a multicast address, the latter requires specification of a ttl value.

A.6.2 The *UDP_send()* Function

int *UDP_send*(char *address, int port, int socket, int arg_num, ...)

Arguments:

***address:** destination IP address in dotted decimal notation

port: destination port

socket: socket to sent UDP packet

arg_num: length of the following variable argument list; number of strings which will build the message

... : variable argument list which consists of pointers to char, specifying the strings which should be sent

Returned value:

number of bytes sent by *UDP_send()*

Description:

The *UDP_send()* function sends a message via UDP to a specified destination. To specify the destination of the UDP packet, the IP address in dotted decimal notation and the appropriate port are given as the first two arguments to the procedure. The message is compound of several strings which are specified as the last arguments of the *UDP_send()* function. The number of strings which should be sent must be specified by the `arg_num` variable before the variable string list.

A.6.3 The *UDP_read()* Function

char **UDP_read*(int fd, int *port, char *addr)

Arguments:

fd: socket which becomes readable and from which message was received

***port:** pointer to int in which the destination port of the message is stored

***addr:** pointer to char in which the destination address (doted decimal notation) of the received message is stored

Returned value:

- pointer to the read message
- NULL, if reading of the message failed

Description:

The *UDP_read()* function reads new incoming UDP messages like SIP requests or responses. After reading a new message it will return the pointer to this message or the null pointer if reading failed. To read the message, the *recvfrom()* function is used which can be also used to extract the source address and port of the incoming UDP packet. To make this data available to the calling function, *UDP_read()* needs a pointer to `int` and a pointer to `char` as second and third argument in which the port and the IP address (in dotted decimal notation) will be stored. The function is invoked if an UDP socket becomes readable.

A.7 The *tcp.c* File

Procedures in the *tcp.c* file are used to perform communication based on TCP.

A.7.1 The *alarm_handler()* Function

void *alarm_handler*()

Arguments:

-

Returned value:

-

Description:

The *alarm_handler()* function sets a timeout flag for the *TCP_read()* function and is invoked if *TCP_read()* can't receive a whole TCP message during an interval of ten seconds. It is used to avoid deadlocks in *TCP_read()* .

A.7.2 The *TCP_read()* Function

char *TCP_read*(int fd, inv_list_t *inv)

Arguments:

fd: socket which becomes readable and from which message will be received

***inv:** pointer to an entry in invitation list if the incoming message belongs to an existing `inv_list_t` structure or NULL is a new invitation arrives

Returned value:

- pointer to the read message
- NULL, if reading of the message failed

Description:

The *TCP_read()* function reads new incoming TCP messages like SIP requests or responses. After reading a new message it will return the pointer to this message or the null pointer if reading failed. The procedure is invoked if a TCP socket becomes readable.
TCP_read() reads data available at file descriptor `fd` until an empty line was received which indicates the end of a the SIP header within a SIP message. A *"Content-Length"* header in the currently read message should specify the message's payload length, which is read afterwards. If no *"Content-Length"* is found, it is expected that no payload is available.
To avoid deadlocks, the *alarm_handler()* function is called, if *TCP_read()* doesn't read the whole message during a timeout interval of 10 seconds. This is done by initiating a alarm signal which will invoke *alarm_handler()* .

A.7.3 The *TCP_send()* Function

int *TCP_send*(int sock, char *msg)

Arguments:

sock: socket in connected state which specifies the destination of the TCP message

***msg:** pointer to the string which should be sent

Returned value:

number of bytes sent by *TCP_send()*

Description:

The *TCP_send()* function sends a message via TCP. The destination is specified by the `sock` argument which has to be a TCP socket in connected state. `msg` is a pointer to the string which should be sent.

A.7.4 The *TCP_connect()* Function

int *TCP_connect*(loc_list_t *loc)

Arguments:

***loc:** location list entry which includes the IP address (`loc->addr`) to which the connection should be established

Returned value:

- new socket to the established connection
- -1, if connection establishment failed

Description:

The *TCP_connect()* function establishes a TCP connection to a remote SIP server specified by `loc->addr`. First, *TCP_connect()* tries to allocate a new socket which should be connected to the destination. The destination port is the standard SIP port stored in the global variable `sip_port`. If the connection establishment is successful, the new socket to this connection is returned, otherwise -1 is given to the calling function.

A.8 The event.c File

A.8.1 The *socket_event()* Function

Notify_value *socket_event*(Notify_client client, int fd)

Arguments:

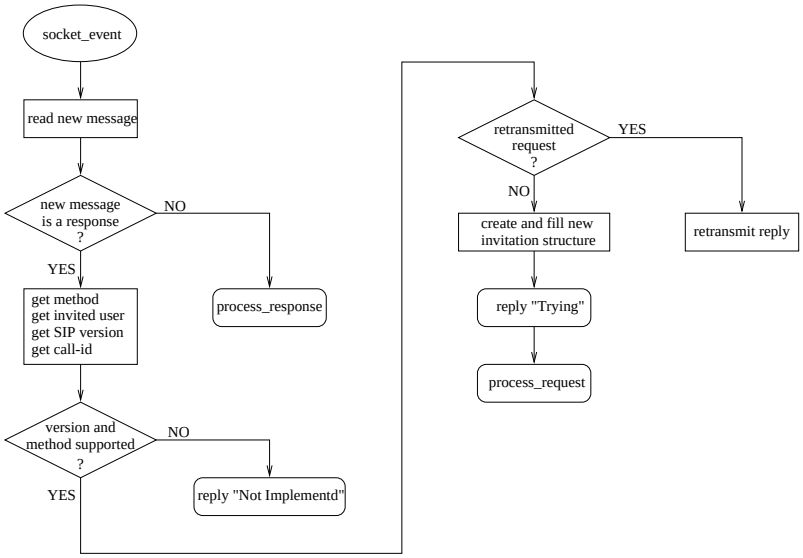
client: pointer to belonging entry in invitation list if the socket of an existing TCP connection becomes readable; NULL, if `tcp_server_socket` or `udp_server_socket` becomes readable

fd: socket which becomes readable

Returned value:

-

Diagram:



Description:

The *socket_event()* function is the event handler function invoked if a TCP of UDP socket becomes readable, except of the `isc_socket` which invokes *isc_socket_event()*. The function handles new incoming SIP messages, decides if these messages are requests or responses and calls the appropriate functions to go on. →

Notify_value *socket_event()*
As the diagram shows, *socket_event()* first reads the new incoming SIP message and stores it in the variable `msg`. This is done by calling *TCP_read()* or *UDP_read()* depending on the kind of the file descriptor `fd` which is the readable socket. If the new message is a SIP response, the *process_response()* procedure is called which handles it. Otherwise, the message is a request and *socket_event()* extracts and checks different parameters specified in the request. Because UDP requests are retransmitted until a definite response was received by the SIP client, it is possible that the incoming message was a retransmitted request. Therefore, the *socket_event()* function searches for an entry in the invitation list which has the same call-id like the new message. If an appropriate entry can be found, the incoming message was a retransmitted request which is ignored after the last response which was sent to the calling client is retransmitted. (This is done by invoking *check_response_priority()* . To handle a new request, the event handler creates a new invitation list entry (*inv_list_add()*) in which the message parameters are stored and invokes the *process_request()* function after sending a *"Trying"* response to the calling client which indicates that the request was received.

A.9 The *request.c* File

Procedures in the *request.c* file are used to handle new incoming requests. They perform name mapping, automatic reply or user location functions.

A.9.1 The *locate_user()* Function

int *locate_user*(inv_list_t *inv)

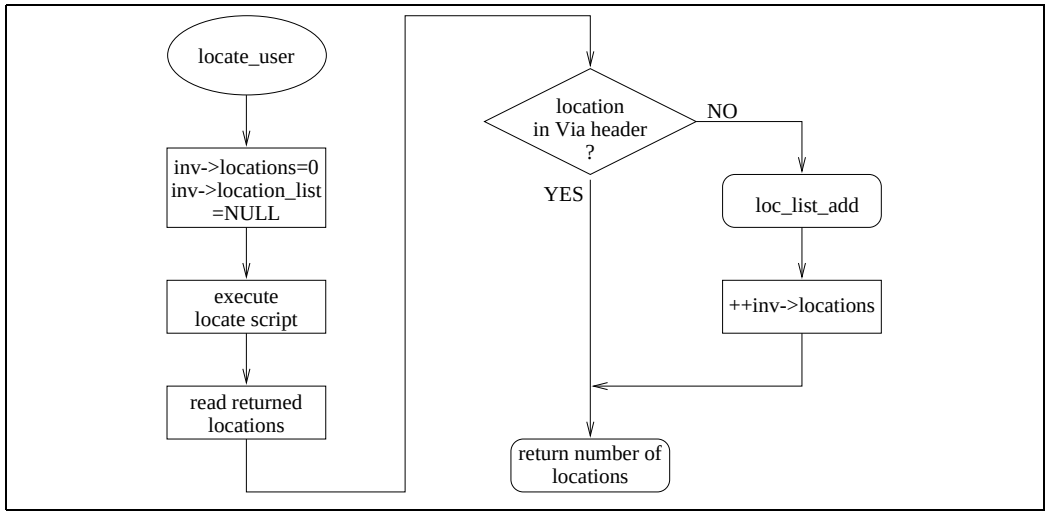
Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

Returned value:

- number of locations where the called user is possibly located
- -1, on failure

Diagram:



Description:

The *locate_user()* function searches the invited user in the local domain. The number of hosts on which the callee is currently logged in is returned to the calling procedure. →

int *locate_user()*
The only argument of *locate_user()* is the pointer to the invitation list entry of the SIP request which is currently in process. As the diagram shows, the *locate_user()* procedure first resets the `inv->locations` variable, which is a counter for the extracted locations. Now, the Tcl script *locate* is executed with the local user name of the callee (stored in `inv->userEntry->pw_name`) as argument. The script returns a list of hostnames at which the invitee was found. Since looping of requests is forbidden by SIP, *locate_user()* checks if the addresses of the extracted locations are already given in the request's *via* header which indicates that the request was forwarded to this location. If so, the extracted location is ignored. Otherwise the new location is added to `inv->location_list` by calling *loc_list_add()* and the `inv->locations` counter is incremented by one. The function returns with `inv->locations`.

A.9.2 The *automatic_reply()* Function

int *automatic_reply*(inv_list_t *inv)

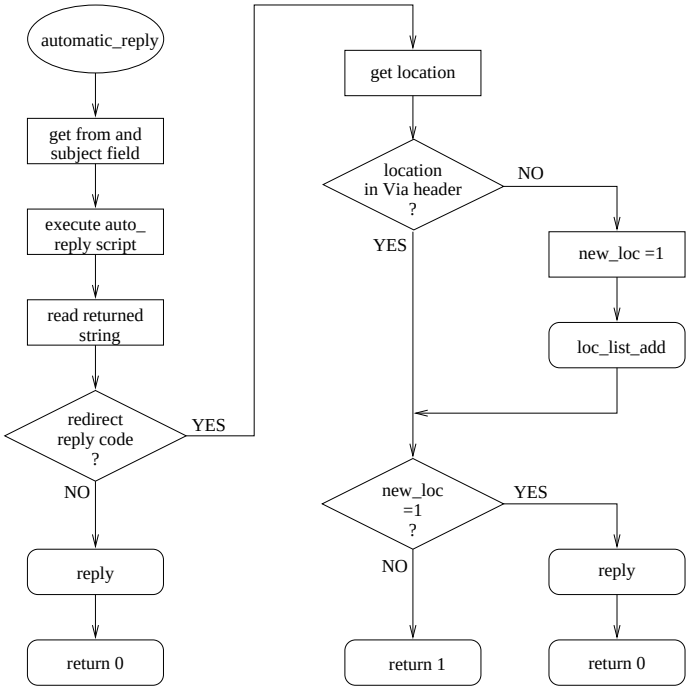
Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

Returned value:

- 0, if an automatic reply was sent
- 1, if no automatic reply was sent

Diagram:



Description:

The *automatic_reply()* function checks if the invited user wants to accept the call or if he wants to send a SIP reply automatically. If so, the function initiates the sending of the appropriate response and returns with 0. →

int *automatic_reply()*
As shown in the diagram, *automatic_reply()* first extracts the sender of the invitation (*"From"* header) and the call's subject out of the received request which is stored in `inv->request`. The *auto_reply* Tcl script which is executed, checks if the invited user wants to accept the call or if a reply should be sent automatically. This is indicated by a returned line which consists of a reply code followed by a string. If the reply code specifies a redirect response, the given string is interpreted as a list of space-separated locations, otherwise it is handled as a reason phrase and sending of the reply is initiated by calling the *reply()* function. Handling of redirect responses requires some additional action. First, *automatic_reply()* has to check if the specified locations consist of a user and host (*"user@host"*) or only a host part and handles it appropriately. Afterwards, it checks if the locations are already queried (by examination the *via* headers of the request) and appends new locations to the `inv->location_list` (calling *loc_list_add()*). Finally it calls the *reply()* function to create and send the redirect response. The *automatic_reply()* function terminates with a return code of 0 if a response was sent, otherwise the return code is 1.

A.9.3 The *expand_user_name()* Function

char **expand_user_name*(inv_list_t *inv)

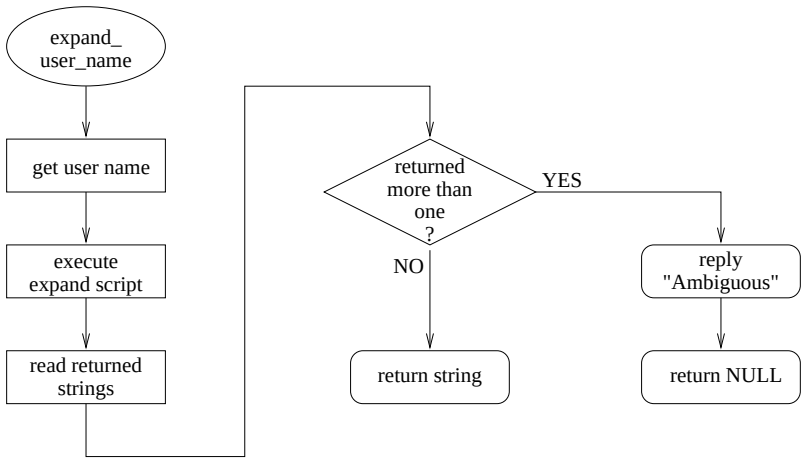
Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

Returned value:

- string which includes the expanded local user name
- NULL, if local user name can't be extracted or was ambiguous

Diagram:



Description:

The *expand_user_name()* function tries to match the name of the invited user, given in the first SIP request line, to the local user name. Therefore, it executes the Tcl script *expand* with the given user name as argument which tries to map the name to the local user name. The script returns a list of strings which include the extracted user names. If more than one user name string is extracted by the script, name mapping can't be done ambiguously and an appropriate response is sent to the calling client (invoked by function *reply()*). The format of the returned strings are described in Section 5.6.6.1.
expand_user_name() returns the string, given by *expand*, if an unambiguous match was found, otherwise it sends an appropriate response to the calling client and returns the null pointer.

A.9.4 The *process_request()* Function

void *process_request*(inv_list_t *inv)

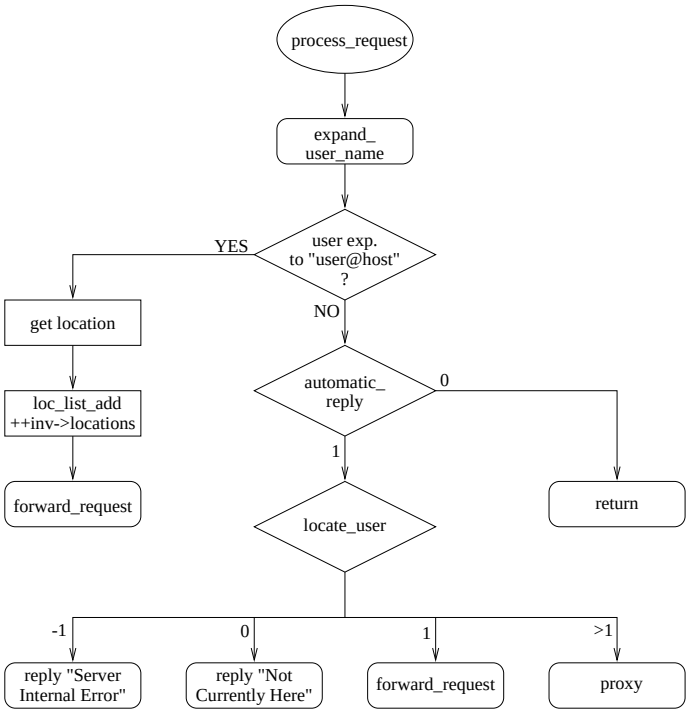
Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

Returned value:

-

Diagram:



Description:

The *process_request()* function is invoked by *socket_event()* after receiving a new incoming SIP request and is the main function to handle new incoming SIP requests. It manages the calling of functions to get the local user name of the invitee (procedure *expand_user_name()*), to check if the called user wants to accept the call (procedure *automatic_reply()*) and to initiate user location (procedure *locate_user()*). Depending on the returned value of *locate_user()* , the *process_request()* function calls different procedures to go on with handling the call.

→

void *process_request()*
An overview of the *process_request()* function is shown in the diagram which starts with calling the *expand_user_name()* procedure which returns a string if name mapping can be done unambiguously. As described in Section 5.6.6.1, the string can consist of different forms. If a user name including a location (user@location) is returned, the SIP request is forwarded to this location. Therefore a location list entry is created and the *forward_request()* procedure is invoked. Otherwise, the local user name is extracted out of the returned string and request processing continues. Call acceptance is checked by the function *automatic_reply()* which returns 1 if the request was not automatically answered by the called procedure. Depending on the returned value of the *locate_user()* function (which is called next), *process_request()* sends an appropriate reply or calls the *forward_request()* or *proxy()* function. If the user is located only at one host (return value 1, the request is forwarded to this location (= proxy mode, function *forward_request()*), but if the invitee was found at several locations, the *proxy()* function is called which handles call forwarding or redirection with respect to the SIP specification.

A.10 The *proxy.c* File

Procedures in the *proxy.c* file are used to decide whether to forward requests (proxy-mode) or to send redirect responses and to perform call forwarding.

A.10.1 The *get_via_header_no()* Function

int *get_via_header_no*(char *req)

Arguments:

***req:** pointer to a SIP request

Returned value:

number of *via* headers in `req`

Description:

get_via_header_no() function calculates and returns the number of given *via* headers within a SIP request. It is called to check if a SIP server is the first one in a chain of servers and to calculate request timeout intervals.

A.10.2 The *forward_request()* Function

void *forward_request*(loc_list_t *loc)

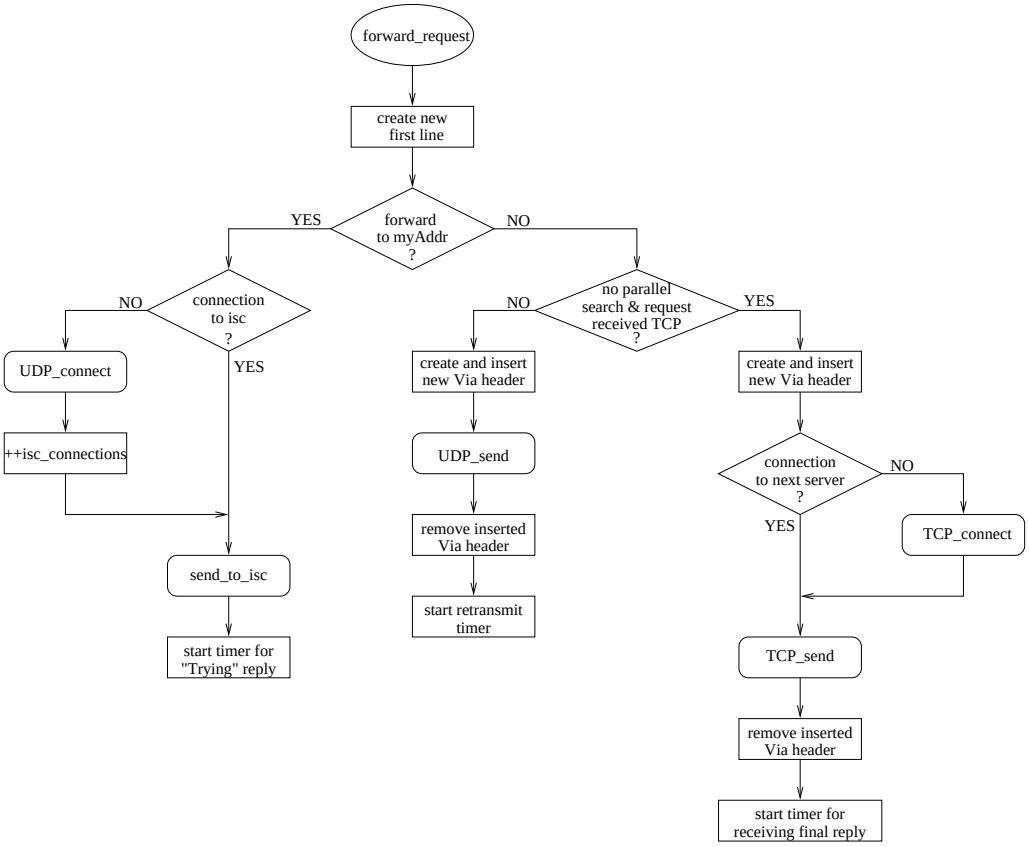
Arguments:

***loc:** pointer to the location list entry which stores the destination to which a SIP request should be forwarded

Returned value:

-

Diagram:



Description:

The *forward_request()* function initiates SIP request forwarding to a new location specified by `loc`. It calls functions to send a call as well to the local session controller isc as to other SIP servers located on remote hosts. →

void *forward_request()*
Before forwarding a request, the *forward_request()* procedure creates a new first request line. This is necessary because the next destination of a SIP request is one part of the first request line. The new location is built of `loc->user` and `loc->addr`. Afterwards the procedure has to check whether to send the request to isc, or to a remote host. To forward the request to isc, a connection to the local pmm multicast group of isc must exist. This is indicated by the global `isc_connections` variable which is greater than 0 if sipd is a member of the local multicast group. Otherwise the *UDP_connect()* function establishes the appropriate connection and the `isc_connections` variable is incremented by one. Forwarding the request to the local multicast group is done by calling the *send_to_isc()* procedure. If the *UDP_connect()* or the *send_to_isc()* function return with a value which indicates an error condition, a *"Server Internal Error"* response is initiated (calling the *reply()* function) or the *isc_request_timeout()* function is called, if sipd awaits responses from additional destinations (indicated by a `inv->locations` variable greater than one). After sending the call to the local multicast group, it can't be sure that isc was already running on the local host. To make reliable that isc receives the request, a timer is started. If a *"Trying"* response from isc is received by sipd, the timer is canceled because isc was already running and has received the call. Otherwise, the timer expires and invokes the *isc_timeout()* function. This function starts isc and retransmits the request to the local multicast group (see Section 5.4). To send the request to a remote host, *forward_request()* decides whether to send the SIP call via TCP or UDP. The former transport protocol (TCP) is used, if sipd received the request via TCP, and the request is not forwarded to several destinations (parallel_search). Otherwise, UDP forwarding is performed. In both cases, the *forward_request()* function creates a new *via* header which is inserted to the request in front of existing ones. Before sending the request via TCP, *forward_request()* establishes a TCP connection (*TCP_connect()* function) to the SIP servers at the remote workstation if both servers are not already connected. Afterwards an event handler for the new socket is established which invokes *socket_event()* if the socket file descriptor becomes readable (i.e. a SIP response from the called server is received). After sending the request to the destination host, the inserted *via* header is removed to rebuild the original incoming SIP request. If forwarding of the request failed, the `inv->locations` counter is decremented by one, since no response is expected from the location specified by `loc`. Moreover, *forward_request()* initiates an appropriate error message (reply *"Server Internal Error"*). →

void forward_request
Depending on the transport protocol, different timer functions are invoked to handle retransmitting or deadlocks. If the request is sent by UDP, a retransmit mechanism is invoked since it can't be sure that the remote server has received the request. Therefore a timer is started which invokes the the *retransmit_timeout()* function after a given interval of time. This timer is canceled if a definite or final response belonging the forwarded request is received. Since TCP is a reliable transport protocol, no retransmit mechanism is used, but a timer which avoids waiting an unlimited amount of time for a response from the remote server is established. If the timer expires, the *request_timeout()* procedure is called. Like in UDP, a definite or final response cancels the timer.

A.10.3 The *proxy()* Function

void *proxy*(inv_list_t *inv)

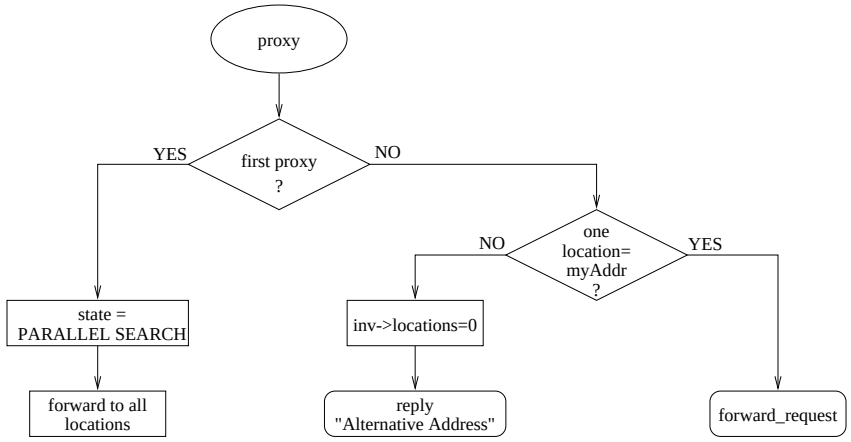
Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

Returned value:

-

Diagram:



Description:

The *proxy()* function decides whether to forward or to redirect a SIP call and calls the appropriate procedures. It is invoked, if *locate_user()* extracts more than one prospect location of the invited user.
As the SIP specification states (see Section 2.7.2), a parallel search is only permitted if a proxy server is the first one in a chain of proxies. To decide whether to forward the request to several locations or send a redirect response, *proxy()* examines the number of *via* headers (procedure *get_via_header_no()*) which indicates the path, the request has taken so far.
If only one *via* header is found in the request, *proxy()* initiates a parallel search and forwards the request to all locations stored in `inv->location_list`.

→

void proxy
Otherwise, it checks if the request should be forwarded to the local session controller isc (indicated by a location equal to the local host (stored in `myAddr`)). If so, the request is forwarded to isc (calling *forward_request()* with the `myAddr` location). If no response is received from isc after a timeout interval, an *"Alternative Address"* response is created to inform the calling SIP client about the other extracted locations. The third scenario (no parallel search and no forwarding to isc) results in an *"Alternative Address"* response which is initiated by the *reply()* function. Before calling *reply()* , the *proxy()* procedure sets the `inv->locations` variable to zero, because no responses from other SIP servers belonging to the processing call are expected.

A.11 The response.c File

Procedures in the request.c file are used to create and send replies to the calling SIP client and to handle incoming responses from called SIP servers.

A.11.1 The *get_response_priority()* Function

int *get_response_priority*(int code)

Arguments:

code: response code of a SIP response

Returned value:

priority of the response code

Description:

The *get_response_priority()* function returns a priority code appropriate to the given response code which is used to classify the response.
The response priority is used to decide if a given response is an informational, definite or final response. An informational response (code 1xx) only deals with information that a request was received or a server tries to notify a callee, whereas a definite response (code 5xx, 6xx) indicates that a server has finished processing a SIP request and no additional responses are expected from this server. A final response indicates that the request was successful (code 2xx), has definitely failed (4xx) or should be redirected (3xx). Final responses terminate a parallel search regardless of outstanding responses from other locations.
Moreover, the response priority is used to order definite responses received from several locations when performing a parallel search. If no final response but different 5xx and 6xx responses are received, the SIP specification deals with a ordered list, which states which response should be received to the client.

A.11.2 The *reply()* Function

void *reply*(inv_list_t *inv, int code, char *phrase)

Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the processing SIP request

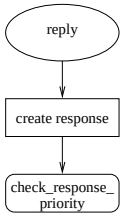
code: reply code

***phrase:** reason phrase

Returned value:

-

Diagram:



Description:

The *reply()* function creates a SIP response due to the specified reply code and reason phrase. It doesn't send the reply to the SIP client but calls *check_response_priority()* which checks if the response should be sent immediately and initiates the appropriate function.
To create a SIP response, the *reply()* procedure first creates a new first line with respect to the given arguments. Afterwards the SIP header of the request (without the first line) is appended and modified. If `code` specifies a redirect response, new *location* headers are added to the response.

A.11.3 The *check_response_priority()* Function

void *check_response_priority*(inv_list_t *inv, char *msg, int priority)

Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the appropriate SIP request

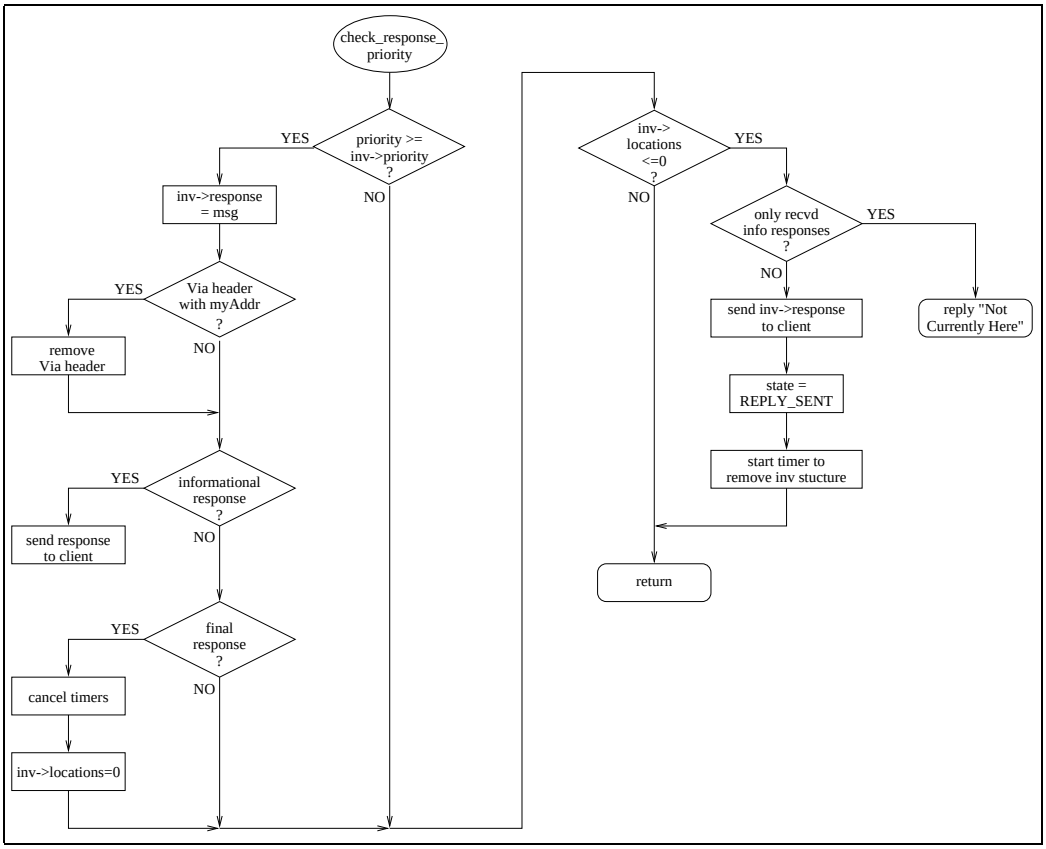
***msg:** response which should be sent to the calling SIP client

priority: priority of the response stored in msg

Returned value:

-

Diagram:



Description:

The *check_response_priority()* function checks if the response given by `msg` should be sent immediately to the calling SIP client or cached until a response was received from all queried locations when performing a parallel search. It handles responses created by sipd and responses which are received from other SIP servers or from isc. →

void *check_response_priority()*
When performing a parallel search, sipd has to wait until it received a definite response from every server the request was forwarded to, or until it received a final response which terminates the parallel search. Moreover, it should only send one definite or final response to the SIP client and thus has to cache the response with the highest response priority (see Section 5.5). After a SIP request was received or a reply was created, the *check_response_priority()* function is invoked by *process_response()* or *reply()*. First the procedure checks if the new response, stored in `msg`, has a higher priority than the currently cached one (`inv->response`) whose priority is given in `inv->response_priority`. If so, `msg` and `priority` are stored in `inv->response` and `inv->response_priority`. If the new response is a final response, all retransmit and request_timeout timers for the invitation specified by `inv` are canceled. Besides the `inv->locations` variable is set to zero because no responses from other location are expected. This will terminate the parallel search. Now, the *check_response_priority()* function checks if the response, stored in `inv->response`, should be sent to the SIP client. The `inv->locations` variable counts the number of location from which a response is expected. If it is zero, every server to which the request was forwarded has sent a definite response, or one server sent a final response. This indicates, that the response should be sent to the initiating client. If no definite of final responses are received by sipd, it initiates a *"Not Currently Here"* reply, otherwise the response stored in `inv->response` is sent to the client. After sending the response, the process of the SIP request is finished and the `inv` structure could be removed from the invitation list. This is not done immediately because it can't be sure that the response was received by the client when using UDP transport. So, the `inv` list entry is cached for a interval of time. If the calling SIP client doesn't receive the response, it retransmits his SIP request. sipd receives the retransmitted call and sends the response, stored in `inv->response`, to the client once more.

A.11.4 The *forward_to_alternative_loc()* Function

int *forward_to_alternative_loc*(inv_list_t *inv, char *l)

Arguments:

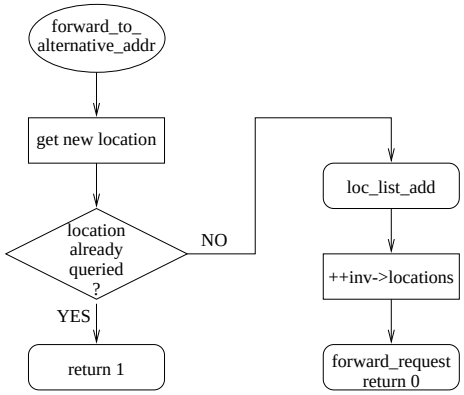
***inv:** pointer to the invitation list entry which stores the parameters of the appropriate SIP request

***l:** pointer to string which stores the new alternative SIP location

Returned value:

- 0, if the request was forwarded to the alternative location
- 1, if the request was not forwarded to the alternative location

Diagram:



Description:

The *forward_to_alternative_loc()* function is invoked if an *"Alternative Address"* response was received. After extracting the new locations, the procedure checks if the request was already forwarded to the given location by calling *loc_list_search()*. If no appropriate location list entry is found, the new location is added to the location list (*loc_list_add()*), the `inv->locations` counter is incremented by one and the *forward_request()* function is called to send the request to the new location.

A.11.5 The *process_response()* Function

int *process_response*(char *msg, char *addr)

Arguments:

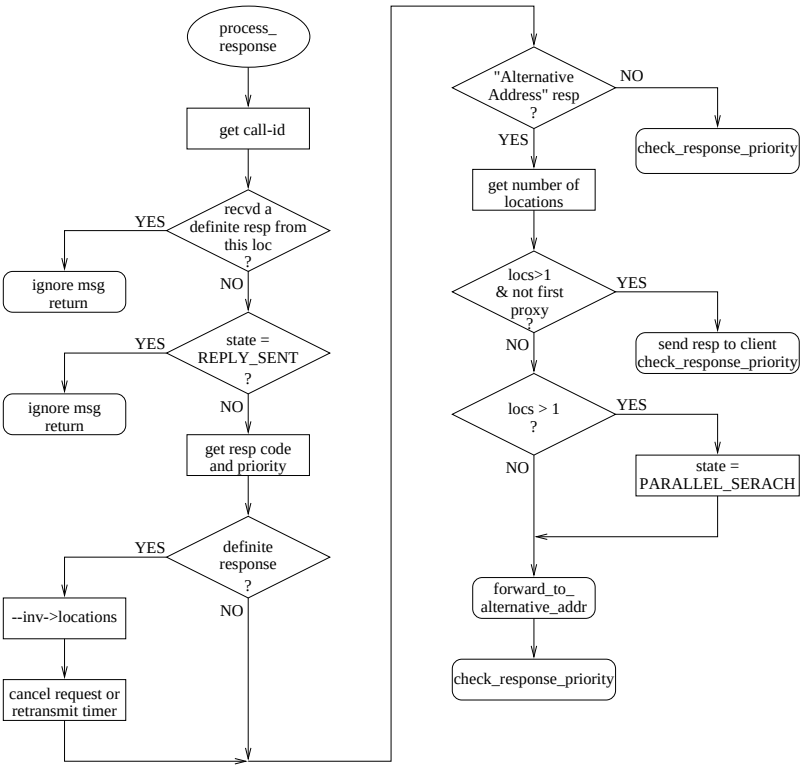
***msg:** pointer to the received response

***addr:** pointer to the destination IP address of the received response

Returned value:

-

Diagram:



Description:

The *process_response()* function is invoked if a new SIP response was received to handle it appropriately. First, it extracts the call-id, searches for the appropriate invitation list entry (*inv_list_search()*) and checks if the request should be ignored. Therefore, the function checks if the `response_received` flag of the belonging location list entry is set or if a definite or final response for the invitation was already sent to the calling SIP client (indicated by `inv->state` = *REPLY_SENT*). This must be done to make sure that duplicated response packets are ignored.

→

int *process_response()*
Afterwards, *process_response()* extracts the response code and its priority (function *get_response_priority()*). If the response is a definite or final response the `inv->locations` counter is decremented by one since a response from this location is no longer expected. To make sure that other responses from this location are ignored, the `loc->response_received` flag is set. Retransmit and request_timeout timers are canceled for this location. If the reply isn't an *"Alternative Address"* response, the *check_response_priority()* function is invoked which decides whether to send a response to the calling client or not. An *"Alternative Address"* response initiates some additional functions: After counting the number of alternative user locations given in the response, the *process_response()* function has to decide whether to forward the SIP request (stored `in inv->request`) to the new given location(s) or to send the response to the SIP client which initiates the call. If the response specifies more than one location, but `sipd` is not the first proxy within a chain of servers it calls the *check_response_priority()* function. Otherwise `inv->state` is set to initiate a parallel search. Invoking the *forward_to_alternative_loc()* function for each alternative location given in the response, sends the SIP request to these locations if they are not already queried. Afterwards the *check_response_priority()* procedure is invoked with an priority of zero. This must be done to make sure that the response is handled correctly if the request was not forwarded by *forward_to_alternative_loc(). check_response_priority()* won't store the *"Alternative Address"* response in `inv->response` (because of the priority of zero) but checks if responses from all servers, the request was forwarded to, was received (`inv->locations counter`). If so, the received response with the highest priority will be sent to the calling SIP client.

A.12 The *isc.c* File

Procedures in the *isc.c* file are used to enable communication between sipd and isc.

A.12.1 The *isc_socket_event()* Function

void *isc_socket_event*(Notify_client client, int fd)

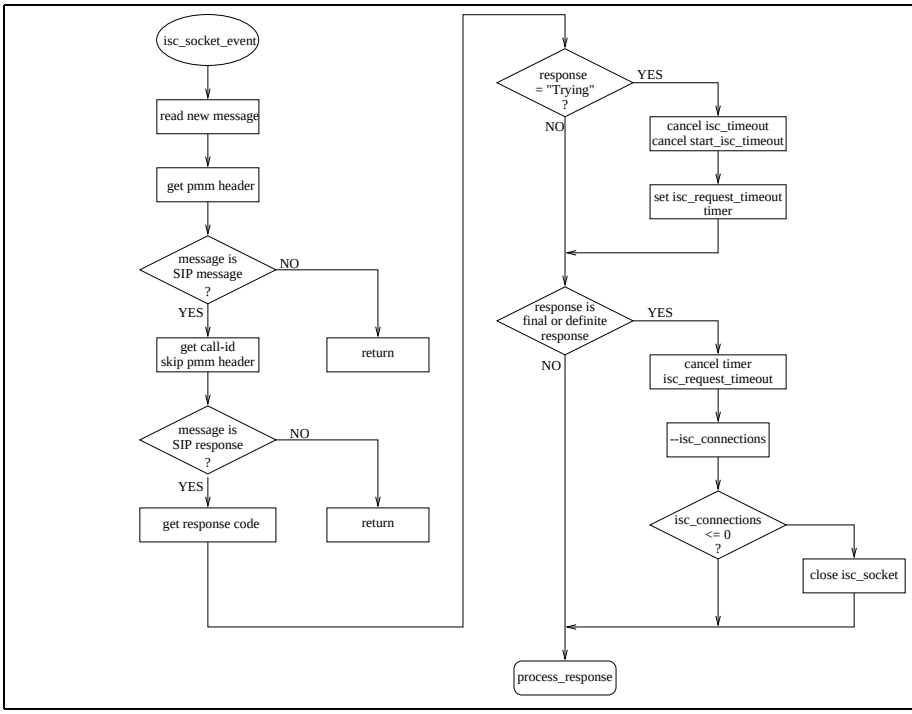
Arguments:

fd: socket which becomes readable; local multicast socket to isc

Returned value:

-

Diagram:



Description:

The *isc_socket_event()* function is invoked if the local pmm multicast socket to the isc multicast group becomes readable. It is used to receive and handle SIP responses from the local session controller isc. →

void *isc_socket_event()*
After reading the incoming message and extracting the pmm header, *isc_socket_event()* checks if the received message was a SIP message which is indicated by the *"sip"* identifier in the pmm header. Afterwards the message's call-id which is part of the pmm header is extracted and the appropriate invitation list entry is searched (calling *inv_list_search()*). Since only SIP responses are expected from isc, *isc_socket_event()* checks if the message was a reply and extracts its response code. A *"Trying"* response indicates that the SIP request was received by isc and lets sipd cancel the isc_timeout and start_isc_timeout timers. Moreover, a new timer is started which is used to wait for a specified duration of time for a definite or final response from the session controller. Expiring of the timer invokes the *isc_request_timeout()* function. If the received response is a definite or final response the *isc_request_timeout()* timer is canceled and the `isc_connections` counter is decremented because the SIP request is finished. If no other invitation which needs a connection to the local multicast group is in process (indicated by a `isc_connection` counter of zero), the `isc_socket` is closed. Finally the *process_response()* procedure is invoked which continues with processing the response.

A.12.2 The *send_to_isc()* Function

int *send_to_isc*(inv_list_t *inv, char *msg)

Arguments:

***inv:** pointer to the invitation list entry which stores the parameters of the appropriate SIP request

***msg:** message which should be sent to the local session controller isc

Returned value:

number of bytes sent to the local multicast group

Description:

The *send_to_isc()* function is used to send a SIP request to the local session controller isc. Since isc uses a local multicast group to communicate with its media controllers, local multicast could also be used to transmit SIP requests between sipd and the local session controller. A pmm header must be created to indicate the receiver of the pmm message. It is built of the request's call-id and the identifier *"sip"* : *"call-id/sip/0"*. To avoid conflicts with the pmm header format, all slash characters ("/") in `call_id` are replaced by underline characters ("_"). Moreover the SIP request which should be sent to isc has to be modified because the routine of isc which receives pmm messages doesn't accept carriage return or line feed characters in a single pmm message. Therefore all CR or LF characters are replaced by "tab" characters which indicate the end of a SIP line. The invoked *UDP_send()* function sends the pmm header and the modified SIP request to the local multicast group.

A.12.3 The *start_isc()* Function

int *start_isc*(struct passwd *userEntry)

Arguments:

***userEntry:** stores the environment of the invited user

Returned value:

- 0, if the *start_isc* script is executed
- 1, if the *start_isc* script can't be executed

Description:

The *start_isc()* function is invoked to start the local session controller isc in the invited user's environment. It is called if the *isc_timeout()* timer expires which indicates that isc is not running for the called user or is not able to handle the request.
After setting the environment of the invitee and changing the user and group of the process according to the invitee, the *start_isc* script is executed which should start the local session controller isc.

A.13 The *timer.c* File

Procedures in the *timer.c* file are invoked if an appropriate timer expires.

A.13.1 The *close_isc_socket()* Function

Notify_func *close_isc_socket*(Notify_client client)

Arguments:

-

Returned value:

-

Description:

The *close_isc_socket()* function is invoked if the global `isc_connections` counter is set to zero. The `isc_socket` is closed.

A.13.2 The *request_timeout()* Function

Notify_func *request_timeout*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *request_timeout()* function is invoked if no definite or final response was received from the called SIP server during a defined interval of time (*T_request*). The `inv->locations` counter is decremented and the event handler for the socket to this server (if the request was forwarded via TCP) is removed. Finally a *"Not Currently Here"* response is initiated.

A.13.3 The *isc_request_timeout()* Function

Notify_func *isc_request_timeout*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *isc_request_timeout()* function is invoked if no definite or final response was received from isc during a defined interval of time (*T_isc_req*).
The `isc_connections` and `inv->locations` counter are decremented. If there are no more invitations in process which communicate with isc, the `isc_socket` is closed (calling *close_isc_socket()*). To avoid that sipd reacts to incoming responses for this request from isc, the appropriate `loc->response_received` flag is set. If there are additional locations to query and sipd isn't performing a parallel search, an *"Alternative Address"* response, otherwise a *"Not Currently Here"* response is initiated.

A.13.4 The *start_isc_timeout()* Function

Notify_func *start_isc_timeout*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *start_isc_timeout()* function is invoked if the local session controller isc was started by the *start_isc()* function, the SIP request was sent to the local multicast group and isc doesn't send a *"Trying"* response after a specified interval of time. It is expected that isc can't be started or doesn't work correctly. The *isc_request_timeout()* function is invoked which handles appropriately.

A.13.5 The *isc_timeout()* Function

Notify_func *isc_timeout*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *isc_timeout()* function is invoked if a SIP request was sent to the local multicast group of isc but no response was received until the *T_isc* timeout interval. It is expected that isc was not running and thus the *start_isc()* function is called. After waiting a short duration of time to give isc some time to start, the SIP call is retransmitted to the local multicast group and the *T_start_isc* timer is invoked which calls *start_isc_timeout()* if no *"Trying"* response was received.

A.13.6 The *retransmit_timeout()* Function

Notify_func *retransmit_timeout*(Notify_client client)

Arguments:

client: pointer to the location list entry which stores the parameters of the appropriate location

Returned value:

-

Description:

The *retransmit_timeout()* function is periodically invoked until a definite or final response was received from the SIP server at the given location or a maximum number of retransmissions is reached.
The retransmit mechanism is used if SIP requests are forwarded via UDP since it can't be guaranteed that the call reaches the destination. First, the *retransmit_timeout()* procedure checks if the request was forwarded too often to the next SIP server. If so, the `inv->locations` counter is decremented, the `loc->response_received` flag is set and a *"Not Currently Here"* response is initiated.
Otherwise the request is retransmitted once more and the *T_retransmit* timer is established again.

A.13.7 The *close_invitation()* Function

Notify_func *close_invitation*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *close_invitation()* function is invoked if the process of a SIP request is totally finished. The sockets stored in the invitation list entry are closed and the `inv_list_t` structure is removed in the invitation list.

A.13.8 The *close_event_handlers()* Function

Notify_func *close_event_handlers*(Notify_client client)

Arguments:

client: pointer to the invitation list entry which stores the parameters of the appropriate SIP request

Returned value:

-

Description:

The *close_event_handlers()* function is invoked if the process of a SIP request is totally finished. All used event handlers and timers are canceled. Finally the *close_invitation()* function is invoked.

A.14 The *sipd.c* File

Procedures in the *sipd.c* file are invoked if the session invitation daemon is started. Initialization, server socket allocation and starting of the the main server loop are processed. The sipd *main()* function is also part of the *sipd.c* file.

A.14.1 The *Exit()* Function

void *Exit*()

Arguments:

-

Returned value:

-

Description:

The *Exit()* function is invoked to terminate sipd. *Exit()* removes the server socket event handlers, terminates the event loop (*notify_stop()*) and closes the server sockets. Finally sipd is terminated.

A.14.2 The *create_server_socket()* Function

int *create_server_socket*(char *protocol, int qlen)

Arguments:

***protocol:** specifies the transport protocol of the new server socket; "*tcp*" or "*udp*"

qlen: length of the listen queue for the TCP server socket

Returned value:

new server socket

Description:

The *create_server_socket()* function allocates and binds SIP server sockets for TCP or UDP.

A.14.3 The *server_main_loop()* Function

void *server_main_loop*()

Arguments:

-

Returned value:

-

Description:

The *server_main_loop()* function enables the daemon mode, calls *create_server_socket()* to get a TCP and UDP server socket, installs the server socket event handlers and starts the event loop (*notify_start()*). Now, sipd is able to react to incoming SIP messages.

A.14.4 The *main()* Function

main()

Arguments:

-

Returned value:

-

Description:

The *main()* function is invoked if `sipd` is started. First, it stores the IP address of the host on which is is executed in `myAddr`. After verifying the command line options, the *server_main_loop()* function is invoked.

Appendix B

Instruction Manual

This section deals with instructions how to install and use the Session Invitation Daemon (sipd) and the Location Service Daemon (lswhod), which are part of the Session Invitation Terminal (SIT). Moreover, user instructions of the SIP enhancements to the Integrated Session Controller (isc) are given.

B.1 The Session Invitation Terminal

The Session Invitation Terminal (SIT) is a software package which contains the Session Invitation Daemon (sipd) and the Location Service Daemon (lswhod). Moreover, the namemapper and aliases applications are part of SIT.

B.1.1 Introduction

The Session Invitation Daemon (sipd) is a per-host daemon which waits for incoming SIP requests and handles them with respect to the SIP specification. It manages user location and redirects or forwards incoming calls. If an invitee is located at the local host, sipd forwards the SIP call to the Integrated Session Controller (isc) which builds the user interface to the callee. So, it is necessary to install isc additionally.

The Location Service Daemon (lswhod) which is also part of SIT, maintains a location service data base which is used by sipd to locate the invitee.

Both, the Session Invitation Daemon and the Location Service Daemon should run on each host where session invitation should be possible. Particularly, sipd should be installed on the domain's mail server host, since a SIP request whose callee was specified by the email address is sent to the SIP server at the mail server host.

B.1.2 Installation

1. Switch into the sit directory.
2. Run ./configure and answer the following questions:
   - *"Directory, where to install the sit binaries"*
     The directory in which the sipd, lswhod, namemapper and aliases binaries will be installed.
   - *"Directory, where to install the sipd scripts"*
     The directory in which the Tcl scripts, used by sipd, will be installed.
   - *"Directory, where to put the location service data base"*
     The directory which will contain the location service data base.
   - *"Path, where to find the aliases data base"*
     The path to the file which deals with the mail aliases (Typically */etc/mail/aliases*).
   - *"Directory, where to find the binary of the Session Controller ISC"*
     The directory which contains the binary of the Integrated Session Controller isc which will be started by sipd to notify the callee.
   - *"Directory, where to find the images and audio directory of the Session Controller ISC"*
     The directory which contains the *images* and *audio* directories of the Integrated Session Controller isc.
   - *"Directory, where to find the tclsh"*
     The directory which contains a tclsh used by the sipd Tcl scripts.
3. Type make to compile the binaries.
4. Type make install to install the components in the directories specified in the configure script.

B.1.3 Usage

Login as root and start sipd and lswhod on each host which should have the ability to handle SIP calls.

B.1.3.1 Command Line Arguments

- sipd
  - -p *SIP port*
    Port number at which sipd awaits SIP requests and to which SIP messages are forwarded by sipd.

- lswhod
  - -d *directory*
    Directory of the location service data base
  - -D
    Starts lswhod in daemon mode
  - -i *interval*
    Update interval of the data base in minutes.
  - -v
    Starts lswhod in verbose mode.

B.2 The Integrated Session Controller

This section describes the usage of the SIP enhancements to the Integrated Session Controller (isc). It builds an expansion to the isc handbook.

B.2.1 Introduction

The SIP enhancements to the Integrated Session Controller enable isc to act as a SIP client which allows to invite users to multimedia sessions and to act as a SIP server in conjunction with the Session Invitation Daemon (sipd).

B.2.2 Installation

1. Switch into the isc directory and modify the Makefile according to your local environment.
2. Type make to compile isc.

B.2.3 Usage

B.2.3.1 Command Line Options

- -s *SIP port*
  Port number on which isc awaits SIP requests and to which SIP messages are forwarded by sipd.

B.2.3.2 Inviting Users to New or Existing Sessions

The Integrated Session Controller offers the user the ability to invite remote users to existing sessions or to initiate new multimedia conferences. To perform a session invitation, the *"Invite"* button of isc has to be pressed.

If there are already sessions active, their session parameters are automatically loaded to the appearing standard or advanced invitation window. This enables the user to invite callees to these sessions or to modify the settings.

The Standard Invitation Window

The standard invitation window shown in Figure B.1 can be used to invite remote users to one audio and/or one video session. Therefore, the caller has to select the audio or video checkbutton. Additionally, the following parameters can be specified in the standard invitation window:

Figure B.1: Standard Invitation Window

Callee(s): A list of remote users who should be invited to the session. Each user must be specified by *user@host*, his email address or an alias, stored in the SIP phonebook (see Section B.2.3.4). If a user name without a host part is specified, the local host name is appended automatically which requires that a SIP server runs on the local host.

Users in the callee list must be separated by a single space character.

Title: Name of the conference.

Subject: Information about the conference which will be shown to the invitee.

Destination: Unicast or multicast address of the conference. (If several media streams use different addresses, these address has to be specified in the appropriate configuration window.)

TTL: Time-to-live value with respect to the address specified in *"Destination"*.

Port: Port of the belonging audio or video stream.

To configure the media streams in a more detailed way, the *"Configure"* button opens an appropriate window which lets the user to select the requested audio or video encodings. The *"Configure audio parameters"* window is shown in Figure B.2 which offers the following parameters:

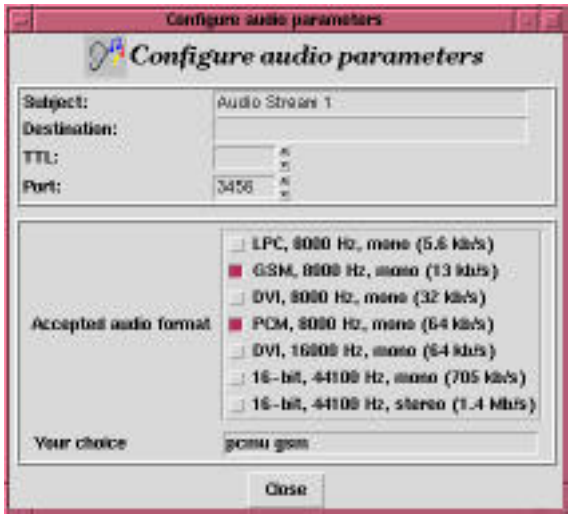


Figure B.2: Audio Configuration Window

Subject: Information about the media stream which will be shown to the invitee.

Destination: Unicast or multicast address of the media stream. Only required if the address differs from the one specified for the whole conference.

TTL: Time-to-live value with respect to the address specified in *"Destination"*.

Port: Port of the belonging audio or video stream.

Media encodings: The requested media encodings are specified by clicking on the belonging checkbuttons.

After specifying the invitation and session parameters, pressing the *"Send invitation"* button starts session invitation.

The Advanced Invitation Window

The advanced invitation window shown in Figure B.3 occurs after pressing the *"Advanced Invitation Configuration"* button in the standard invitation window or if the *"Invite"* button of isc was selected and more than one audio and one video session is active. It enables to specify a large number of media streams within a session invitation.

The window is separated into two parts. The upper part deals with the invitation parameters as in the standard invitation window plus some additional one:

URI: URI where the invitee can get more information about the invited session.

E-mail: E-mail address where the invitee can get more information about the invited session.

Phone number: Phone number where the invitee can get more information about the invited session.

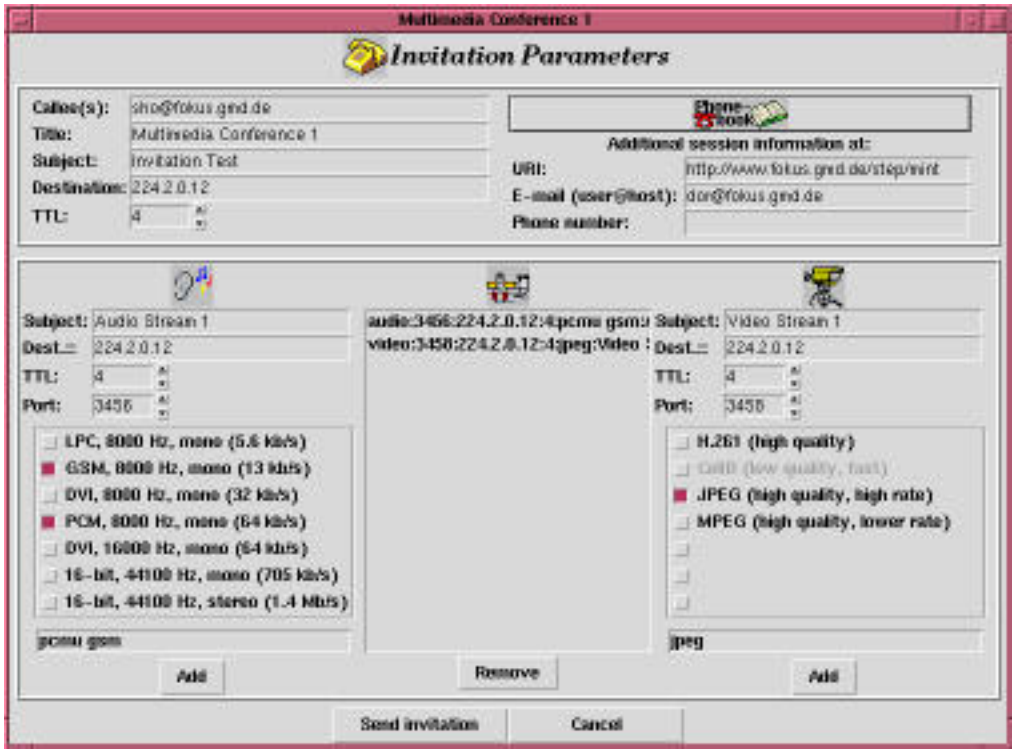


Figure B.3: Advanced Invitation Window

The lower part of the advanced invitation window is used to select the requested media streams. Therefore, the user has to select the appropriate settings of a single media stream in the left (audio) or right (video) frame. Pressing the *"Add"* button appends the specified parameters to the listbox in the middle. This listbox shows the media streams to which the callee(s) will be invited to. To remove a media stream out of the listbox, the appropriate entry has to be selected. Pressing the *"Remove"* button deletes the entry.

To send the invitation, the *"Send invitation"* button has to be pressed finally.

Results of Session Invitations

After sending an invitation, the call results in one or more responses which deal with the current status, success or failure of the call. To present the caller this information, an appropriate window occurs (Figure B.4).

If an invitation fails but may be successful with some modifications given in the response packet, an appropriate window comes up (Figure B.5). Pressing the *"Modify request"* button opens the standard or advanced invitation window which lets the user to modify and retransmit the call. The alternative settings given in the response are automatically selected in the invitation window.

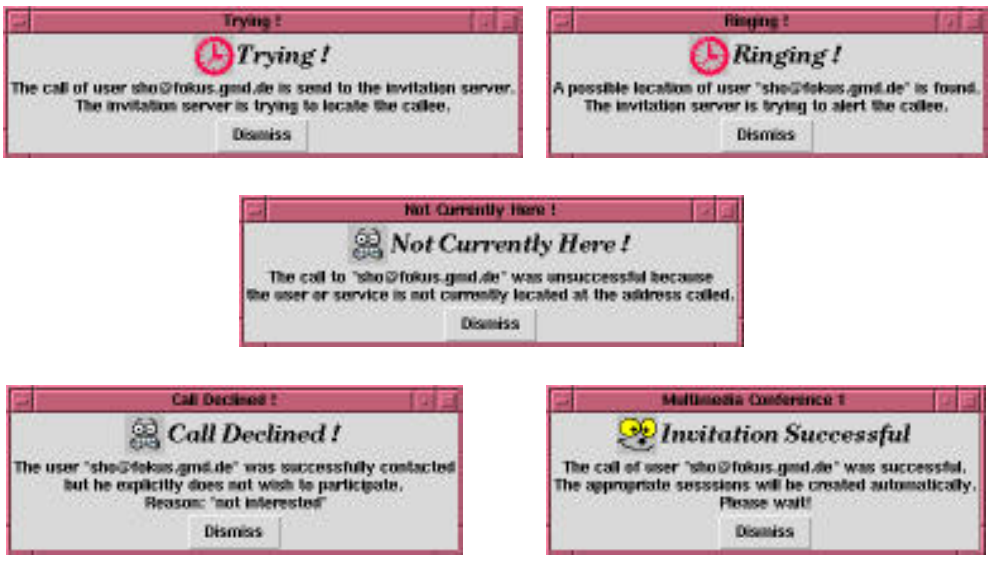


Figure B.4: Information Windows

Figure B.5: Information Windows

B.2.3.3 Receiving a Session Invitation

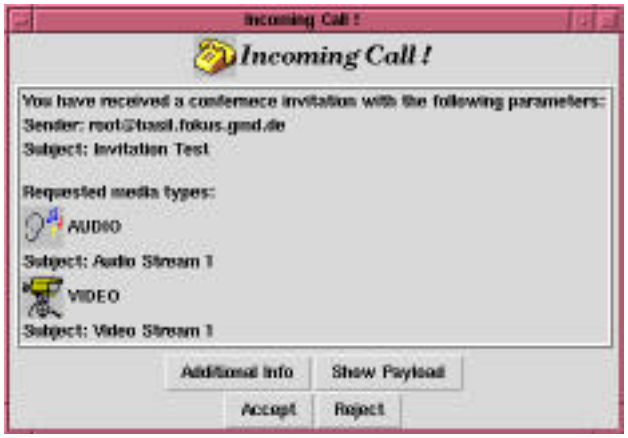


Figure B.6: *"Incoming Call"* Window

When receiving a session invitation, the callee is notified by the so called *"Incoming Call"* window which is shown in Figure B.6. The window deals with information about the invitation's sender and the session(s) requested by the call. Pressing the *"Accept"* button creates the appropriate session(s) automatically and sends a response message to the caller.

To reject the call, pressing the *"Reject"* button opens a sub-window which lets the callee specify a reason phrase why he declines the call. Pressing the *"Send"* button sends the reject message to the inviter.

B.2.3.4 The SIP Phonebook

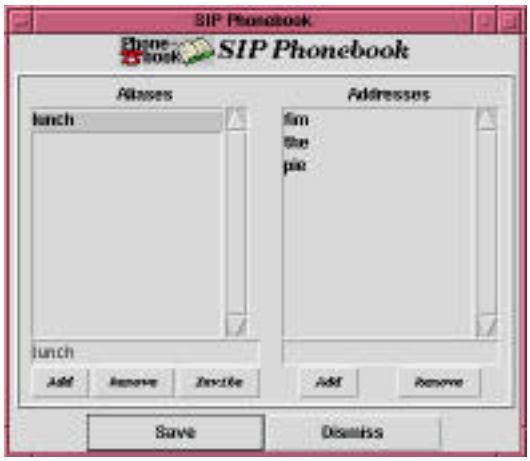


Figure B.7: SIP Phonebook

The SIP phonebook can be used to store a list of SIP addresses under a single alias. Pressing the *Phonebook* button in the standard or advanced invitation window opens a

simple phonebook editor shown in Figure B.7. The phonebook window consists of two parts. On the left side, a listbox deals with the aliases, whereas the right side lists the addresses stored for the selected alias.

Adding or Removing Aliases

To add a new alias, the name of the alias must be specified in the box under the left listbox. Please note, that no space characters are permitted within an alias. Pressing the *"Add"* button inserts the new alias. To remove an alias, select the appropriate list entry and press the *"Remove"* button.

Adding or Removing Addresses to an Alias

To add a new SIP address to an alias, first select the appropriate alias in the left listbox. Afterwards specify the SIP address (*"user@host"*) under the right listbox and press the *"Add"* button. To remove a SIP address, select the appropriate list entry and press the *"Remove"* button.

To store the settings, made in the SIP phonebook window, press the *"Save"* button.

B.2.3.5 Automatic Invitation Handler

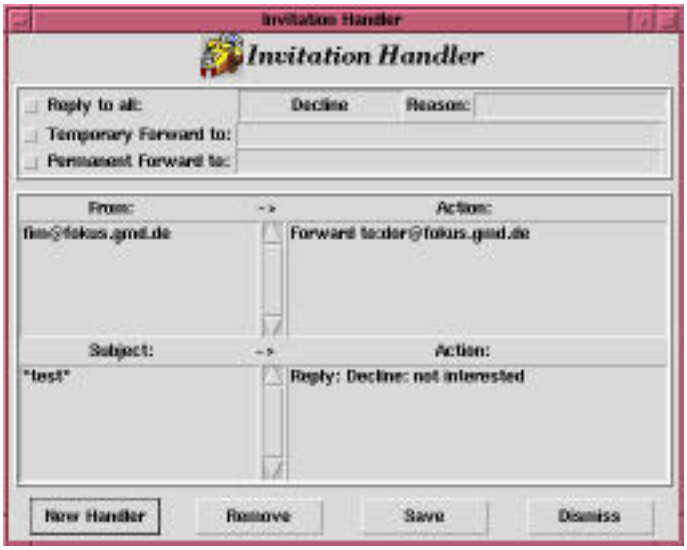


Figure B.8: Invitation Handler Window

Pressing the *"Invitation Handler"* button in the main isc window opens a simple editor (Figure B.8) which can be used to specify handlers which initiate automatic replies to incoming invitations.

Handler for all Incoming Invitations

To reply to all incoming calls independent of the inviter or subject of the invitation, select one of the checkbuttons in the upper part of the window. If one of the redirect buttons is selected one or several SIP address has to be specified, each separated by a single space character.

Handlers for Invitation with Respect to their Sender or Subject

To add a new handler which replies to incoming invitations with respect to their sender or subject, pressing the *"New Handler"* button opens the window shown in Figure B.9. First select one of the *"From"* or *"Subject"* checkbuttons to specify whether the handler should react to the call's sender or subject. Then fill in the appropriate item. To specify how to react to a call that matches the handler's criterion, select the *"Forward to"* or *"Reply"* checkbutton and fill in the appropriate item. Finally press the *"Add"* button.

Figure B.9: *"Add Invitation Handler"* Window

To remove a handler, select the handler in the invitation handler window and press the *"Remove"* button.

Appendix C

Source Code