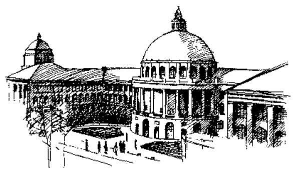
ETH Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

TIK Institut für Technische Informatik und Kommunikationsnetze
Computer Engineering and Networks Laboratory



Diploma Thesis

Convergence of Internet and Intelligent Networks:

Interaction of services using PINT

Bernhard Höneisen

Helsinki, 10th of September 1999

Professors	Albert Kündig (ETH) and Raimo Kantola (HUT)
Assistant	Urs Röthlisberger (ETH)
Instructor	Hannu Flinck (Nokia Research Center)

HELSINKI UNIVERSITY OF TECHNOLOGY
Laboratory of Telecommunications Technology

NOKIA
CONNECTING PEOPLE

Abstract

This thesis describes the advance in interaction of services between Internet and Intelligent Networks. One approach followed by the IETF Working Group PINT, which addresses connection arrangements through which Internet applications can request and enrich PSTN[1] telephony services, is the main focus of the thesis.

After an introduction about the main differences between Internet and Intelligent Networks and about interacting services (chapter 1), the protocols SIP (chapter 2) and SDP (chapter 3) are described, since the PINT protocol (chapter 4) is built on those. A prototype application using the PINT protocol is documented in chapter 5. The other parts cover results (chapter 6) and performance (chapter 7) of the thesis and at the end there is an outlook (chapter 8) about how the work can be continued.

The main result of the thesis is the prototype application, described in chapter 5. It includes also a section about the Parser which is used. Ideas for PINT related services can be found in section 4.1. Some proposals for improving the document about the PINT protocol [1] are listed in section 6.2.

[1]Public Switched Telephone Network

Conceptional Formulation

NOKIA DOCUMENTTYPE 1 (4)

Nokia Research Center
Hannu Flinck 10.05.1999

Diploma Thesis Spring term 99

Convergence of Internet and IN services

Bernhard Höneisen

1. DATES

Begin:	11th of May 1999
End:	10th of September 1999

2. RESPONSIBILITIES

- Tutoring

  H. Flinck, Nokia Research Center, Helsinki, Finland,
  +358-40-584 2977, hannu.flink@nokia.com

  P. Pöyhönen, Nokia Research Center, Helsinki, Finland
  +358-40-749 9159, petteri.poyhonen@nokia.com

  U. Röthlisberger, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
  +41-1-632 5447, ursr@tik.ee.ethz.ch

- Professors

  Prof. R. Kantola, Helsinki University of Technology (HUT), Espoo, Finland
  +358-9-451 2471, +358-40-540 3127, Raimo.Kantola@hut.fi

  Prof. A. Kündig, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland
  +41-1-632 7020, kuendig@tik.ee.ethz.ch

3. INTRODUCTION

Teleoperators have been introducing Internet services and networks into their service portfolio for some time already. The interaction with Internet based services and infrastructure with the conventional telephony systems and services such as PSTN, GSM and IN does not yet exist. However the convergence of telephony networks and Internet, that is motivated by new services and sharing management and operations costs, is likely to tighten the relationships of these two networks. This convergence seems to start from common services that then are reflected to the network infrastructure as gateways and new protocols. As an example of this development one can mention the work around IP telephony in ITU in terms of H.323 recommendation and recent activities in IETF in the IP telephony working group, the PSTN and Internet Interfaces (pint) working group and Multiparty Multimedia Session Control (mmusic) working group. In addition, Lucent has announced Internet Call Waiting, an IN software solution that will let online internet users receive a call-waiting message on their computer screens when a telephone call comes in on the same phone line used for surfing.

The convergence of Internet and telephony networks has so far addressed the wireline networks, mostly because the mobility support within Internet is still in its early states. Some development is however taken place in this area in form of an ETSI proposal called CAMEL that builds upon IN service control to manage mobile services. In order to share management costs and find common services it is natural starting point to consider how Internet services can be related to the IN supplementary services.

4. OBJECTIVE OF THE THESIS

The objective of the thesis is to explore the Convergence of Internet and IN Services and to provide a prototype demonstrating interworking capabilities. The emphasis is on the related Internet protocols and related services.

5. TECHNICAL APPROACH

At the high level work is divided into following phases:

1. Comparison of IN and Internet architectures and main features (weeks 19-20)
2. Identification of common/relevant interworking scenarios and services (week 21)
3. System level specification (weeks 22-23)
4. Building of the development and test environment (weeks 24-25)
5. Proof of concept (weeks 26-31)
6. Final report (weeks 32-36)

Since IN and Internet are both very wide subjects and research domains of their own and the interworking cases between them are several, a major concern is to scope the work into the one or two representative implementable cases. The emphasis of the work is to understand the interworking implications from Internet perspective rather than IN. Since the work is dealing with complex systems as such, it is anticipated that in the prototyping that will take place in the proof of concept phase, most of the IN-system features must be emulated.

5.1 Comparison of IN and Internet architectures and main features

The purpose of this phase is to familiarize into the main characteristics of both IN and Internet architectures. The nature of the work is literature study. (See appendix that lists some of the related material). The expected outcome of this phase is to have clearly articulated descriptions of the main differences of the mentioned systems.

NOKIA DOCUMENTTYPE 3 (4)

Nokia Research Center
Hannu Flinck 10.05.1999

5.2 Identification of common/relevant interworking scenarios and services

Both IETF and ETSI are working on the IN-Internet interworking issues what an assumption of some service set. The purpose of this work phase is to select a representative service and an interworking scenario that extends the ongoing work in either or both of these standardization bodies. The nature of the work is conceptual. The expected outcome of the work is a reference model that defines the relevant interworking entities and the protocols between them.

5.3 System level specification

The system level specification refines and details the above mentioned reference model. As a specification methodology UML and OMT are favorable. The outcome of this phase is to have a detail description of the availability of the software entities (some may be public domain), definition of the required changes into them, new software modules and definition which part to the system must be emulated.

5.4 Building of the development and test environment

Client(s), server(s) and relevant gateways will be configured and the software development environment will be set up.

5.5 Proof of concept

This is implementation phase. The object is to have a working prototype of the defined system and to provide feedback to the system specification.

5.6 Reporting

A detailed report that documents the prototyping effort and the conceptual work will be written.

6. RELATED LITERATURE

IETF Draft, PSTN-Internet (PINT) Working Group, The PINT Profile of SIP and SDP: a Protocol for IP Access to Telephone Call Services, March 1999

IETF Draft, PSTN-Internet (PINT) Working Group, A Proposal for Internet Call Waiting Service using SIP An Implementation Report, January 1999

IETF Draft, PSTN-Internet (PINT) Working Group, A proposal for the provisioning of PSTN initiated services running on the Internet, March 1999

IETF RCF, Network Working Group, Toward the PSTN/Internet Inter-Networking --Pre-PINT Implementations (RFC 2458), November 1998

IETF RCF, Network Working Group, SDP: Session Description Protocol (RFC 2327), April 1998

IETF RCF, Network Working Group, SIP: Session Initiation Protocol (RFC 2543), March 1999

IETF Draft "SS7-Internet Interworking – Architectural Framework <draft-greene-ss7-arch-frame-01.txt>

IETF Draft "SS7-Internet Gateway Archtecture" <draft-ong-ss7-interet-gateway-01.txt>

IETF Draft "H323 Signaling and SS7 ISUP gateway" <draft-ma-h323-isup-gateway-00.txt>

IETF Draft "Simple Gateway Control Protocol" Christian Huitema <deaft-huitema-sgcp-v1-02.txt>

The ITU Telecommunication Standardization, Recommendations Q.1200 - Q.1290

ETSI, GSM 03.78: " "Digital cellular telecommunications system (Phase 2+); Customised Applications for Mobile network Enhanced Logic (CAMEL) Phase 3; Stage 2", version X.1.0 Draft A Release 1999

ITU SWP4-4/11 Chair: Q.5/11 Rapporteur; Ray Forbes, (Marconi Comms, UK), Requirements for the Functional Architecture for IN support of IP-networks,Geneva; 1-19 March 1999

Contents

Chapter 1

Introduction

1.1 Comparing Internet and Intelligent Networks

Internet and Intelligent Networks (IN) have a mutual purpose: Providing connections and services in order to exchange information. But the properties and the way, how they fulfill this purpose differ. Table 1.1 gives an overview on the different properties.

The Internet is characterized by a distributed service architecture; there is no global service creation and provision framework. New services can be created locally by any user which can afford a server. This is one reason for the rapid growth of Internet services in the last couple of years. In IN the service creation is centralized. It provides a powerful service creation and provision platform. So the introduction of a new service takes only a few months nowadays. Due its success IN applications can be found also in mobile communications environment, e.g. CAMEL [2]

The access to standards and the network in order to create and run a service is not as easy as in the Internet. The access to the ITU standards is restricted, while the overwhelming part of Internet Standards is publicly available without licenses.

In Internet there are numerous possibilities for providing a comfortable user interface. Customization is flexible and easy for the user. In most cases the user is provided with a graphical user interface, which is often web-based (changes can be made directly in the browser window). In IN, usually the way for customization goes through the phone itself e.g. using the (limited) keyboard of the phone or by calling an operator, who has a direct access to the IN services. The first method (using keyboard of phone) might be quite long and painful. Both methods are usually charged, since they require a phone call.

The availability of the IN services is much better than in Internet. Almost everywhere, there is a phone, while Internet requires a computer and a connection through an Internet Service Provider (ISP).

Property	**Internet**	**Intelligent Networks**
Service creation	Distributed	Centralized
Availability of Standards	Usually publicly available	Restricted, licensed
User interface, customization of services	Often graphical and flexible	Complicated, long and painful procedures
Life cycle of services	Short	Relatively long
availability of services	Computer and Internet connection required	High, a phone is almost everywhere
Quality of service (QoS) in general	Only best effort	Provides guaranteed Quality of Services
Real time applications	Not (yet) possible in most cases	Fully supported
Quality of voice	Depends on the available bandwidth, the load in the network, etc.	High
Routing	Packet switched	Circuit switched
Delay behavior	Unpredictable, high and variating delays	Constant and short delays
Bandwidth needed	Little, use of physical connections dynamical	High, in order to offer the QoS
Reliability of the connection	Depends on the load in the network, etc.	Very high
Charging	Not well developed yet, mostly flat rate	Flexible, secure billing and charging system
Costs for using the services	Low, mostly flat rate	Can be expensive, service dependent
Estimated revenues	$ 6 billion	$ 600 billion

Table 1.1: Comparing Internet / Intelligent Networks

The quality of service (QoS) in the IN (mobile extensions not considered in this paragraph) is superior to the Internet. It provides guaranteed QoS, e.g. high speech quality, security, reliability. For most real-time applications, there is no problem running them through IN, since it provides short and constant delays, resulting from the circuit switched nature of the network. In Internet the connection is packet switched, which leads to unpredictable behavior (delays and delay variations). This can't be accepted for applications with real-time QoS requirements. (A new protocol for reserving bandwidth in Internet has been designed, but it is not yet widely in use. [3]) The circuit-switched approach provides on one side high QoS, but it requires more bandwidth: A resource (channel) is assigned to a user as long a the connection lasts, even in these moments, when no voice or data is transfered. In the packet-switched world, far less bandwidth is required, as the resources are assigned dynamically and shared. While used for voice transmission, the quality in Internet is dependent mainly on the available bandwidth and on the load in the network. The service is provided ''best effort", which means as good as possible. IN provides high speech quality.

For IN flexible, secure billing and charging systems are available, where as in Internet this is less developed. IN services are usually designed to be value added services. This means, that they can be quite expensive for the user; every service is charged separately when it is used. In Internet lots of services are free of charge, where only the connection to the Internet is to be paid. The access costs to the Internet are usually flat rate.

The total annual revenues in IN are estimated to $600 billion (two orders of magnitude higher than in Internet) [4].

In this brief comparison I only mentioned the most important differences between these two worlds, of course there are many more. For an Introduction to IN and Internet I refer to [5]. More about Internet can be found in [6]. Intelligent Networks are introduced in [7].

1.2 Hybrid Services

Teleoperators have been introducing Internet services and networks into their service portfolio for some time already. The recent growth in the number of Internet users—together with the strong foothold of the PSTN[1]—is creating a demand for a new class of services which can take advantage of both technologies—PSTN and Internet—simultaneously. The authors of [4] call them *hybrid services.* Examples of hybrid services are the PINT Milestone Services (see section 4.1.1), which could be used e.g. to connect the PSTN phones of a browsing user and the responsible agent in a call center, just by clicking a certain link in a web-page. Furthermore interconnecting one using of voice over PSTN with one using voice over IP or possibility to access email (or voice mail) either through Internet or PSTN.

Taken separately neither the PSTN nor the Internet are an ideal ground for developing future hybrid services, but if coupled together, they can complement each other quite effectively, choosing the best properties out of both. For example taking the high QoS of IN and the flexible customization of the Internet.

Hybrid services are expected to play a very important role in the nearer future. The users desire to integrate the ways they communicate, the service providers want to differentiate their offers from their competitors. On the mobile communication side, smart cellular phones such as the ones using WAP[2] [8] or the Nokia Communicator [9], which both include Internet features are also responsible for the rapid growth in the area of hybrid

[1]Public Switched Telephone Network
[2]Wireless Application Protocol

services.

The recent research activities that focus on hybrid services have emerged, especially on *inter-working*. The common approach taken by these activities is to model the PSTN (or Internet) as a stand-alone system whose services can be accessed through a gateway that acts like a PSTN (or Internet) terminal.

One approach is currently explored by an Internet Engineering Task Force (IETF) [10] named PINT[3] (chapter 4). The PINT Working Group addresses connection arrangements through which Internet applications can request and enrich PSTN telephony services. An example of such services is a Web-based Yellow Pages service with the ability to initiate PSTN calls between customers and suppliers. The PINT protocol is on its way to get an Internet Standard. It is an extension based on SIP[4] (chapter 2) and SDP[5] (chapter 3), which are studied by another IETF, the MMUSIC[6] [11] working group.

Another approach is using both networks for routing the information. There are different scenarios for this. A simple example is in normal telephone calls: The two parties can be in different networks (connecting PSTN and Internet telephone users) or the connection between two PSTN users can be switched partly over the Internet. Paavonen describes these scenarios in his Master's thesis [5].

In this report I concentrate on the PINT approach. Since PINT (chapter 4) is built on SIP (chapter 2) and SDP (chapter 3), a description of those is included in this report. In chapter 5 the prototype application, which is using the PINT protocol, is documented. The other parts cover results (chapter 6) and performance (chapter 7) of the thesis and at the end there is an outlook (chapter 8) about how the work can be continued.

[3]PSTN and Internet Interfaces
[4]Session Initiation Protocol
[5]Session Description Protocol
[6]Multiparty Multimedia Session Control

Chapter 2

Session Initiation Protocol (SIP)

The Session Initiation Protocol (SIP) [12][13] is an application-layer control (signaling) protocol for creating, modifying and terminating sessions with one or more participants. These sessions include Internet multimedia conferences, Internet telephone calls and multimedia distribution. Members in a session can communicate via multicast or via a mesh of unicast relations, or a combination of these. SIP invitations used to create sessions carry session descriptions which allow participants to agree on a set of compatible media types. SIP supports user mobility by proxying and redirecting requests to the user's current location. Users can register their current location. SIP is not tied to any particular conference control protocol. SIP is designed to be independent of the lower-layer transport protocol (UDP, TCP, AAL5, X.25, ...) and can be extended with additional capabilities. It is based on the well known Client-Server approach.

SIP is part of the IETF conference control architecture, which consists of:

Session Announcement Protocol (SAP) [14], which can be used e.g. for announcing multimedia sessions. It contains a session description and is distributed via a well known multicast address and port.

Real Time Streaming Protocol (RTSP) [3], an application-level protocol for control over the delivery of data with real-time properties. RTSP provides an extensible framework to enable controlled, on-demand delivery of real-time data, such as audio and video. Sources of data can include both live data feeds and stored clips. This protocol is intended to control multiple data delivery sessions, provide a means for choosing delivery channels such as UDP, multicast UDP and TCP, and provide a means for choosing delivery mechanisms based upon RTP (RFC 1889 [15]).

Session Description Protocol (SDP) [16] can be used as session description payload of SIP, SAP, ...
The SDP is described in chapter 3.

others, such as malloc, multicast, conference bus, ...
For more information see [11].

All these protocols are either already standards in Internet or on the way to.

SIP can also be used in conjunction with other call setup and signaling protocols. In that mode, an end system uses SIP exchanges to determine the appropriate end system address and protocol from a given address that is protocol-independent. For example, SIP could be used to determine that the party can be reached via H.323 [17], obtain the H.245 [18] gateway and user address and then use H.225.0 [19] to reestablish the call.

2.1 SIP and H.323

For signaling and control for Internet telephony, two standards have recently emerged. One is ITU-T Recommendation H.323, and the other one is the IETF SIP. These two protocols represent different approaches to the same problem: H.323 covers the more traditional circuit-switched approach to signaling based on the ISDN Q.931 protocol and earlier H-series recommendations, and SIP favors the more lightweight Internet approach based on HTTP.

In short the specialties of SIP compared to H.323 are:

- Lower complexity
- Low Call Setup Times
- Text Based Encoding
- Designed for IP Networks
- Rich extensibility and better scalability

The H.323 has its strength in designing PBX[1] for Multimedia Applications over a (private) LAN[2]. This is what it was originally designed for. Later the extensions for the use over the (worldwide) Internet were made. The constraint to be compatible to the original design caused a lower performance compared to SIP. H.323 is close to the traditional telephone network signaling protocols, which makes e.g. the interaction of IP telephony and ISDN easier compared to SIP.

A comparison between these two standards covering complexity, extensibility, scalability, and features can be found in [20]. For a more detailed discussion the reader is referred to [21].

[1]Private Branch Exchange
[2]Local Area Network

For the reader that is familiar with H.323 protocol family of ITU-T, here a is short comparison of the concerning protocols:

H.323	**SIP**
H.323	SIP & SDP
H.225.0 + RAS[3]	SIP
H.245	SDP, SMIL [22], ...
gatekeeper	proxy

2.2 Definitions for SIP

The RFC 2543 [13], which contains the SIP specifications, uses a number of terms to refer to the roles played by participants in SIP communications. The definitions of client, server and proxy are similar to those used by the Hypertext Transport Protocol (HTTP) (RFC 2068 [23]). The terms and generic syntax of URI and URL are defined in RFC 2396 [24]. The following definitions are from [13, section 1.3] as they have special significance for SIP:

Call: A call consists of all participants in a conference invited by a common source. A SIP call is identified by a globally unique call-ID (sec. 2.6.1, page 20). Thus, if a user is, for example, invited to the same multicast session by several people, each of these invitations will be a unique call. A point-to-point Internet telephony conversation maps into a single SIP call. In a multiparty conference unit (MCU) based call-in conference, each participant uses a separate call to invite himself to the MCU.

Call leg: A call leg is identified by the combination of Call-ID, To and From (see also sec. 2.6.1).

Client: An application program that sends SIP requests. Clients may or may not interact directly with a human user. User agents and proxies contain clients (and servers).

Conference: A multimedia session (see below), identified by a common session description. A conference can have zero or more members and includes the cases of a multicast conference, a full-mesh conference and a two-party "telephone call", as well as combinations of these. Any number of calls can be used to create a conference.

Downstream: Requests sent in the direction from the caller to the callee (i.e. user agent client to user agent server).

Final response: A response that terminates a SIP transaction, as opposed to a provisional response that does not. All 2xx, 3xx, 4xx, 5xx and 6xx responses are final. See also 2.4, section 2.5.2.

[3]Registration, Admission and Status (between Gatekeeper and Client)

Initiator, calling party, caller: The party initiating a conference invitation. Note that the calling party does not have to be the same as the one creating the conference.

Invitation: A request sent to a user (or service) requesting participation in a session. A successful SIP invitation consists of two transactions: an INVITE request followed by an ACK request.

Invitee, invited user, called party, callee: The person or service that the calling party is trying to invite to a conference.

Isomorphic request or response: Two requests or responses are defined to be isomorphic for the purposes of this document (and [13]), if they have the same values for the Call-ID, To, From and CSeq header fields. In addition, isomorphic requests have to have the same Request-URI.

Location service:/Location server: A location service is used by a SIP redirect or proxy server to obtain information about a callee's possible location(s). Location services are offered by location servers. Location servers may be co-located with a SIP server, but the manner in which a SIP server requests location services is beyond the scope of this document.

Parallel search: In a parallel search, a proxy issues several requests to possible user locations upon receiving an incoming request. Rather than issuing one request and then waiting for the final response before issuing the next request as in a sequential search, a parallel search issues requests without waiting for the result of previous requests.

Provisional response: A response used by the server to indicate progress, but that does not terminate a SIP transaction. 1xx responses are provisional, other responses are considered final. See also 2.4, section 2.5.2.

Proxy, proxy server: An intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced internally or by passing them on, possibly after translation, to other servers. A proxy interprets, and, if necessary, rewrites a request message before forwarding it. See also section 2.3.2.

Redirect server: A redirect server is a server that accepts a SIP request, maps the address into zero or more new addresses and returns these addresses to the client. Unlike a proxy server, it does not initiate its own SIP request. Unlike a user agent server, it does not accept calls. See also section 2.3.3.

Registrar: A registrar is a server that accepts REGISTER requests. A registrar is typically co-located with a proxy or redirect server and may offer location services.

Ringback: Ringback is the signaling tone produced by the calling client's application indicating that a called party is being alerted (ringing).

Server: A server is an application program that accepts requests in order to service requests and sends back responses to those requests. Servers are either proxy, redirect or user agent servers or registrars.

Session: From the SDP specification: "A multimedia session is a set of multimedia senders and receivers and the data streams flowing from senders to receivers. A multimedia conference is an example of a multimedia session." (RFC 2327 [16]) (A session as defined for SDP can comprise one or more RTP sessions.) As defined, a callee can be invited several times, by different calls, to the same session. If SDP is used, a session is defined by the concatenation of the user name, session id, network type, address type and address elements in the origin field.

(SIP) transaction: A SIP transaction occurs between a client and a server and comprises all messages from the first request sent from the client to the server up to a final (non-1xx) response sent from the server to the client. A transaction is identified by the CSeq number (section 2.6.1, page 20) within a single call leg. The ACK request has the same CSeq number as the corresponding INVITE request, but comprises a transaction of its own.

Upstream: Responses sent in the direction from the user agent server to the user agent client.

URL-encoded: A character string encoded according to Section 2.2 of RFC 1738 [25].

User agent client (UAC), calling user agent: A user agent client is a client application that initiates the SIP request.

User agent server (UAS), called user agent: A user agent server is a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user. The response accepts, rejects or redirects the request.

User agent (UA): An application which contains both a user agent client and user agent server.

An application program may be capable of acting both as a client and a server. For example, a typical multimedia conference control application would act as a user agent client to initiate calls or to invite others to conferences and as a user agent server to accept invitations. The properties of the different SIP server types are summarized in table 2.2.

property	redirect server	proxy server	user agent server	registrar
also acts as a SIP client	no	yes	no	no
returns 1xx status	yes	yes	yes	yes
returns 2xx status	no	yes	yes	yes
returns 3xx status	yes	yes	yes	yes
returns 4xx status	yes	yes	yes	yes
returns 5xx status	yes	yes	yes	yes
returns 6xx status	no	yes	yes	yes
inserts Via header	no	yes	no	no
accepts ACK	yes	yes	yes	no

Table 2.2: Properties of the different SIP server types

For description of the status-code classes see table 2.4 on page 19.

2.3 The Architecture of SIP

Table 2.3 lists the elements of a SIP signaling system.

Element	**Short Description**
UAC	User-agent client
UAS	User-agent server
Redirect server	Redirects requests
Proxy server	Can be in server, client or both of them
Registrar	Tracks user locations

Table 2.3: Elements of SIP

2.3.1 User Agent

A User Agent Client (UAC) initiates SIP requests, a User Agent Server (UAS) receives them on behalf of the user. (See also page 9.)

Figure 2.1 shows the basic message flow for the initiation of a SIP session (signaling). The following procedure is performed:

1. The UAC sends an INVITE request to the UAS of the called party
2. The UAS of the called party sends a response on behalf of his user to the UAC.

3. The UAC confirms the reception of the response sending an ACK to the UAS

After this signaling messages, the communication is established. The routing information (where and how to send the data) can be found in the body of the signaling messages (IP number, port, format, ...). See also chapter 3.

Either the UAC or the UAS can terminate the session issuing a BYE request, which the other party sends a response to.

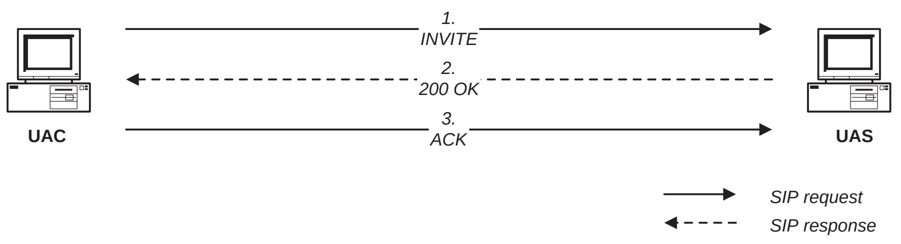


Figure 2.1: SIP basic case for session initiation

2.3.2 Proxy server

A proxy is an intermediary program that acts as both a server and a client for the purpose of making requests on behalf of other clients. The definition of a proxy is similar to the one used by the HTTP (RFC 2068 [23]).

There are two kinds of proxies:

near-end proxy: Used for outgoing calls and responsible e.g. for address lookup, policy, firewalls

far-end proxy: Used for incoming calls and responsible e.g. for callee firewall, call path hiding, address lookup

A proxy may fork requests, so parallel or sequential search are possible (see also page 8).

Figure 2.2 depicts the basic SIP operations using a proxy server (without forking).

The following signaling messages are exchanged:

1. The SIP proxy server receives the INVITE request initiated by the UAC.
2. The proxy server contacts the location server.
3. The proxy server obtains a more precise location of the desired UAS.

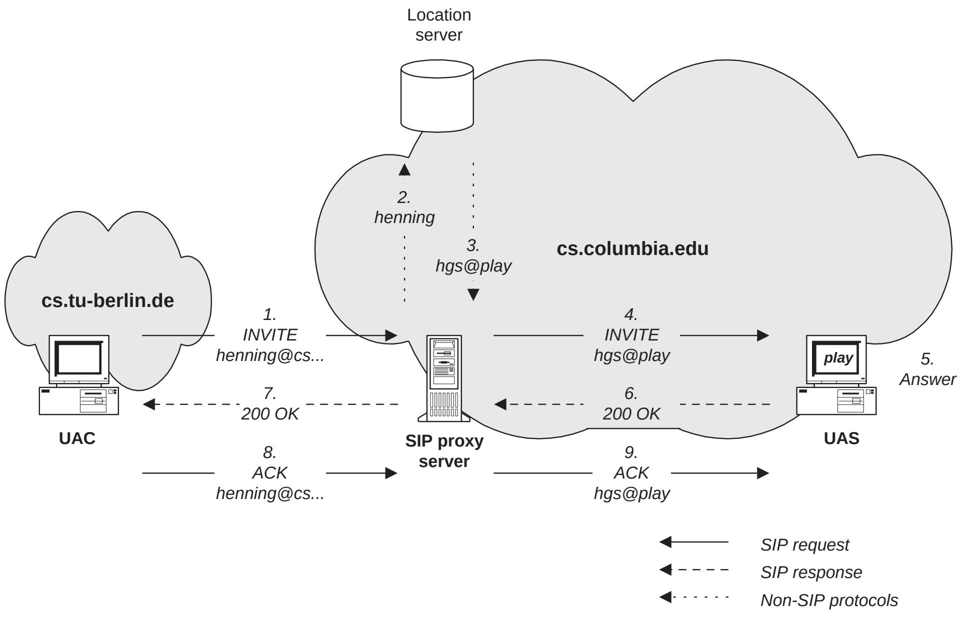


Figure 2.2: SIP proxy server mode

4. The proxy server then issues an INVITE request to the address returned by the location server.
5. The UAS alerts the called party (user) and gets a positive answer from the user.
6. The UAS returns a success indication to the proxy server.
7. The proxy server forwards the response to the UAC.
8. The confirming ACK request is sent by the UAC to the proxy server.
9. The proxy server forwards it to the UAS.

All requests and responses have the same Call-ID (sec. 2.6.1, page 20). [13, pages 15-16]

The used methods (INVITE, ACK) are described in section 2.4.2 and the responses in section 2.5.

2.3.3 Redirect server

The redirect server provides more precise information about the called party (see also page 8). Figure 2.3 shows the SIP signaling operations using a redirect server.

The following signaling messages are exchanged, while using a redirect server (the first three steps are similar as in the proxy server example):

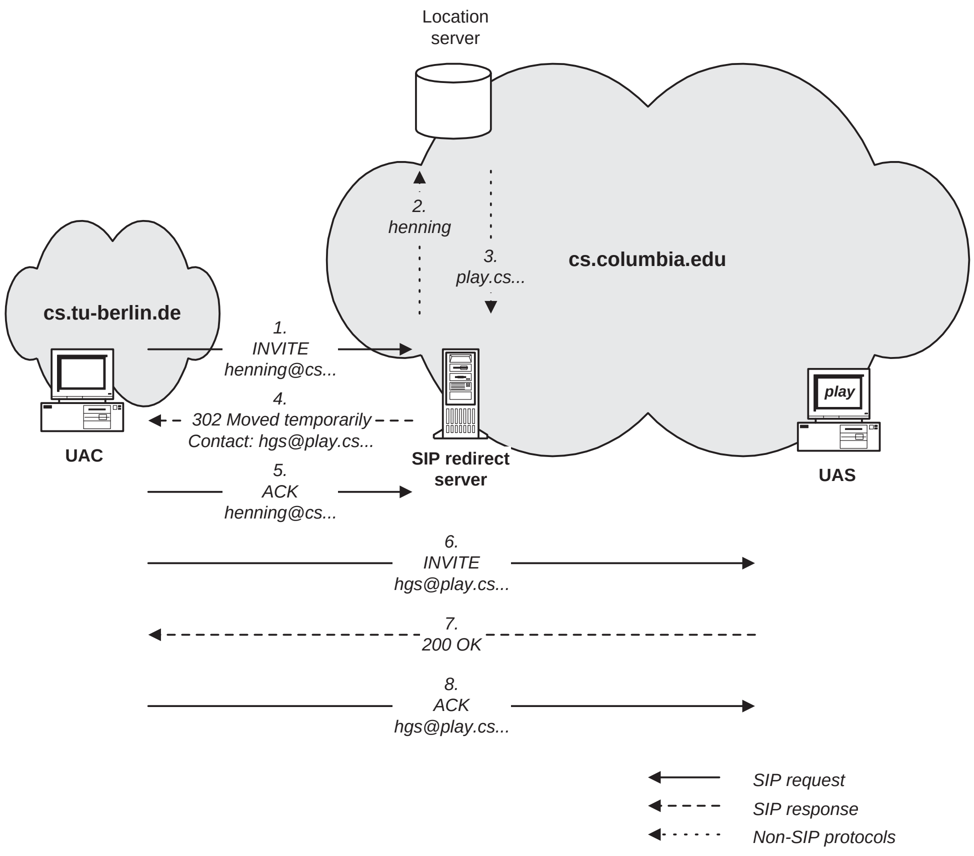


Figure 2.3: SIP redirect server mode

1. The SIP redirect server receives the INVITE request initiated by the UAC.
2. The redirect server contacts the location server.
3. The redirect server obtains a more precise location.
4. Instead of contacting the newly found address itself as in the proxy server example, the redirect server returns this address to the UAC.
5. The UAC acknowledges this with an ACK request.
6. The UAC issues a new INVITE request to the address returned by the redirect server.
7. The UAS sends a successful response for the invitation to the UAC.
8. The UAC completes the handshake with an ACK request to the UAS.

All requests and responses have the same Call-ID. The new INVITE is distinguished from the first one by a higher CSeq (see sec. 2.6.1, page 20)). [13, pages 16-17]

2.3.4 Registrar

A registrar is a server that accepts REGISTER requests (see sections 2.4.2 on page 17 and 2.6.5). A registrar is typically co-located with a proxy or redirect server and may offer location services.

2.4 SIP Request

A SIP Request consists of:

- Request-Line (see sec. 2.4.1)
- General headers (see sec. 2.6.1)
- Request headers (see sec. 2.6.2)
- Entity headers (see sec. 2.6.3)
- Message-body (see sec. 2.6.7)

Example

```
INVITE sip:watson@boston.bell-tel.com SIP/2.0
Via: SIP/2.0/UDP kton.bell-tel.com
From: A. Bell <sip:a.g.bell@bell-tel.com>
To: T. Watson <sip:watson@bell-tel.com>
Call-ID: 3298420296@kton.bell-tel.com
CSeq: 1 INVITE
Subject: Mr. Watson, come here.
Content-Type: application/sdp
Content-Length: 130

v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
s=Mr. Watson, come here.
c=IN IP4 kton.bell-tel.com
m=audio 3456 RTP/AVP 0 3 4 5
```

Remark: All the lines from 'v=0' on are part the message-body, containing the Session description, which is in this case SDP (chapter 3).

2.4.1 Request-Line

The request line contains the following elements:

- SIP method token (see sec. 2.4.2)
- Request-URI (see sec. 2.4.3)
- SIP protocol version (current version: SIP/2.0)

The Request Line ends with CRLF. The elements are separated by SP characters. No CR or LF are allowed except in the final CRLF sequence.

CRLF means line terminator, which is dependent of the system (UNIX, MS-DOS, etc.). It means either *CR*, *LF* or *CR LF*.

Example

```
INVITE sip:watson@boston.bell-tel.com SIP/2.0
```

2.4.2 SIP Methods

In SIP, the following six methods are defined:

Method	**Short description**
INVITE	Initiate call
ACK	Confirm final response
OPTIONS	Query for support of features by other side
BYE	Terminate call
CANCEL	Cancel searches and ”ringing”
REGISTER	Register with location service

INVITE

The INVITE method indicates that the user or service is being invited to participate in a session. The message body contains a description of the session to which the callee is being invited. For two-party calls, the caller indicates the type of media it is able to receive and possibly the media it is willing to send as well as their parameters such as network destination. A success response must indicate in its message body which media the callee wishes to receive and may indicate the media the callee is going to send. A higher CSeq number (see sec. 2.6.1, page 20) for an existing Call leg indicates a re-invite. This is the case e.g. if the session description has changed during the session.

ACK

The ACK request confirms that the client has received a final response to an INVITE request. (ACK is used only with INVITE requests.) 2xx responses (see sec. 2.5.2, table 2.4) are acknowledged by client user agents, all other final responses by the first proxy or client user agent to receive the response. The ACK request is forwarded as the corresponding INVITE request, based on its Request-URI.

OPTIONS

With the method OPTION, a server is being queried as to its capabilities. A server that believes it can contact the user, such as a user agent where the user is logged in and has been recently active, may respond to this request with a capability set. A called user agent may return a status reflecting how it would have responded to an invitation, e.g., 600 (Busy). Such a server should return an Allow header field (sec. 2.6.4) indicating the methods that it supports.

BYE

The user agent client uses BYE to indicate to the server that it wishes to release the call. A BYE request is forwarded like an INVITE request and may be issued by either caller or callee. A party to a call should issue a BYE request before releasing a call (”hanging up”). A party receiving a BYE request must cease transmitting media streams specifically directed at the party issuing the BYE request.

CANCEL

The CANCEL request cancels a pending request with the same Call-ID, To, From and CSeq (sequence number only) header field values, but does not affect a completed request. (A request is considered completed if the server has returned a final status response.) The Call-ID, To, the numeric part of CSeq and From headers in the CANCEL request are identical to those in the original request. This allows a CANCEL request to be matched with the request it cancels. However, to allow the client to distinguish responses to the CANCEL from those to the original request, the CSeq method component is set to CANCEL.

REGISTER

A client uses the REGISTER method to register the address listed in the To header field with a SIP server. A user agent may register with a local server on startup by sending a REGISTER request to the well-known ”all SIP servers” multicast address ”sip.mcast.net” (224.0.1.75).

Requests are processed in the order received. Clients should avoid sending a new registration (as opposed to a retransmission) until they have received the response from the server for the previous one. Section 2.6.5 describes the use of the header fields in the REGISTER method.

2.4.3 Request URI

The Request-URI is a SIP URL [13, section 2] or a general URI. It indicates the user or service which this request is being addressed to. Unlike the To field, the Request-URI may be re-written by proxies.

2.5 SIP Response

After receiving and interpreting a request message, the recipient responds with a SIP response message, which consists of:

- Status-Line (see sec. 2.5.1)
- General headers (see sec. 2.6.1)
- Response headers (see sec. 2.6.4)
- Entity headers (see sec. 2.6.3)
- CRLF message-body (optional) (see sec. 2.6.7)

Examples

- This example contains a temporary response (without message-body):

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP kton.bell-tel.com
From: A. Bell <sip:a.g.bell@bell-tel.com>
To: T. Watson <sip:watson@bell-tel.com> ;tag=37462311
Call-ID: 3298420296@kton.bell-tel.com
CSeq: 1 INVITE
Content-Length: 0
```

- and this one a final response (containing a message-body):

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP kton.bell-tel.com
From: A. Bell <sip:a.g.bell@bell-tel.com>
To: <sip:watson@bell-tel.com> ;tag=37462311
Call-ID: 3298420296@kton.bell-tel.com
CSeq: 1 INVITE
Contact: sip:watson@boston.bell-tel.com
Content-Type: application/sdp
Content-Length: 116

v=0
o=watson 4858949 4858949 IN IP4 192.1.2.3
s=I'm on my way
c=IN IP4 boston.bell-tel.com
m=audio 5004 RTP/AVP 0 3
```

Remark: Again all the lines from 'v=0' are part of the message-body, containing the Session description.

2.5.1 SIP Status-Line

The first line of a response message is the Status-Line, consisting of the following elements:

- SIP protocol version (current version: SIP/2.0)
- numeric Status-Code (see sec. 2.5.2),
- Textual phrase associated to Status Code (see also sec. 2.5.2)

The Status Line ends with CRLF. The elements are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

Example

```
SIP/2.0 302 Moved temporarily
```

2.5.2 SIP Status Codes and Reason Phrases

The Status-Code is a 3-digit integer result code that indicates the outcome of the attempt to understand and satisfy the request. The Reason-Phrase is intended to give a short textual description of the Status-Code. The Status-Code is intended for use by automata, whereas the Reason-Phrase is intended for the human user. The client is not required to examine or display the Reason-Phrase.

In table 2.4 the Status-Code classes are defined. (The first digit of the Status-Code defines the class of response.) For a full list and description of status codes the reader is referred to [13, section 7].

Status-Code	Class	Description
1xx	Informational	request received, continuing to process the request
2xx	Success	the action was successfully received, understood, and accepted
3xx	Redirection	further action needs to be taken in order to complete the request
4xx	Client Error	the request contains bad syntax or cannot be fulfilled at this server
5xx	Server Error	the server failed to fulfill an apparently valid request
6xx	Global Failure	the request cannot be fulfilled at any server

Table 2.4: SIP Status-Code Classes

2.6 SIP Headers

2.6.1 General Headers

General header fields apply to both request and response messages. The following are considered as general header fields:

To:/From: Requests and responses must contain a From and a To header field, indicating the initiator/desired recipient of the request.

Via: The Via field indicates the path taken by the request so far. This prevents request looping and ensures responses take the same path as the requests, which assists in firewall traversal and other unusual routing situations.

Call-ID: globally (time, space) unique call identifier. It uniquely identifies a particular invitation or all registrations of a particular client.

CSeq: Clients must add the CSeq (Command Sequence) header field to every request. A CSeq header field in a request contains the request method and a single decimal sequence number chosen by the requesting client, unique within a single value of Call-ID. A server must echo the CSeq value from the request in its response. CSeq values are monotonically increasing and contiguous. The ACK and CANCEL requests must contain the same CSeq value as the INVITE request that it refers to, while a BYE request canceling an invitation must have a higher CSeq number.

Accept: The Accept header field can be used to specify certain media types which are acceptable for the response. Accept headers can be used to indicate that the request is specifically limited to a small set of desired types. It is used only with the INVITE, OPTIONS and REGISTER request methods.

Accept-Encoding: The Accept-Encoding header field is similar to Accept, but restricts the content-codings that are acceptable in the response.

Accept-Language: The Accept-Language header field can be used to allow the client to indicate to the server in which language it would prefer to receive reason phrases, session descriptions or status responses carried as message bodies. A proxy may use this field to help select the destination for the call, for example, a human operator conversant in a language spoken by the caller.

Contact: The Contact header field can appear in INVITE, ACK, and REGISTER requests, and in 1xx, 2xx, 3xx, and 485 responses (sec. 2.5.2, table 2.4). In general, it provides a URL where the user can be reached for further communications.

Date: The Date header field reflects the time when the request or response is first sent. Thus, retransmissions have the same Date header field value as the original.

Encryption: The Encryption header field specifies that the content has been encrypted.

Expires: The Expires header field gives the date and time after which the message content expires. This header field is currently defined only for the REGISTER and INVITE methods.

Record-Route: The Record-Route header field is added to a request by any proxy that insists on being in the path of subsequent requests for the same call leg. It contains a globally reachable Request-URI that identifies the proxy server.

Timestamp: The timestamp header field describes when the client sent the request to the server. The value of the timestamp is of significance only to the client and it may use any timescale. The server must echo the exact same value. The timestamp is used by the client to compute the round-trip time to the server so that it can adjust the timeout value for retransmissions.

The Call-ID, To and From header fields are needed to identify a call leg. The distinction between call and call leg matters in calls with multiple responses to a forked request.

2.6.2 Request Headers

The request-header fields allow the client to pass additional information—about the request and the client itself—to the server.

The following are considered as request header fields:

Authorization: A user agent may authenticate itself with a server by including an Authorization header field with the request.

Hide: A client uses the Hide request header field to indicate that it wants the path comprised of the Via header fields to be hidden from subsequent proxies and user agents.

Max-Forwards: The Max-Forwards header field may be used with any SIP method to limit the number of proxies or gateways that can forward the request to the next downstream server. This can also be useful when the client is attempting to trace a request chain which appears to be failing or looping in mid-chain.

Organization: The Organization header field conveys the name of the organization to which the entity issuing the request or response belongs. The field may be used by client software to filter calls.

Priority: The Priority header field indicates the urgency (emergency, urgent, normal or non-urgent) of the request as perceived by the client.

Proxy-Authorization: The Proxy-Authorization header field allows the client to identify itself (or its user) to a proxy which requires authentication.

Proxy-Require: The Proxy-Require header field is used to indicate proxy-sensitive features that must be supported by the proxy.

Route: The Route header field determines the route to be taken by a request.

Require: The Require request-header field is used by clients to tell user agent servers about options that the client expects the server to support in order to properly process the request.

Response-Key: The Response-Key request-header field can be used by a client to request the key that the called user agent should use to encrypt the response with. If the client insists that the server return an encrypted response, it includes a *Require: org.ietf.sip.encrypt-response* header field in its request.

Subject: This is intended to provide a summary, or to indicate the nature, of the call, allowing call filtering without having to parse the session description.

User-Agent: The User-Agent header field contains information about the client user agent originating the request.

2.6.3 Entity Headers

The entity-header fields define meta-information about the message-body or, if no body is present, about the resource identified by the request. The term 'entity header' is an HTTP 1.1 term where the response body can contain a transformed version of the message body. The original message body is referred to as the 'entity'. We retain the same terminology for header fields but usually refer to the 'message body' rather then the entity as the two are the same in SIP.

The following are considered as entity header fields:

Content-Encoding: The Content-Encoding entity-header field is used as a modifier to the "media-type". When present, its value indicates what additional content codings have been applied to the entity-body, and thus what decoding mechanisms must be applied in order to obtain the media-type referenced by the Content-Type header field. Content-Encoding is primarily used to allow a body to be compressed without losing the identity of its underlying media type.

Content-Length: The Content-Length entity-header field indicates the size of the message-body, in decimal number of octets, sent to the recipient.

Content-Type: The Content-Type entity-header field indicates the media type of the message-body sent to the recipient.

2.6.4 Response Headers

The "response-header" fields allow the server to pass additional information about the response which cannot be placed in the Status-Line. These header fields give information about the server and about further access to the resource identified by the Request-URI.

The following are considered as response header fields:

Allow: The Allow header field lists the set of methods supported by the resource identified by the Request-URI. The purpose of this field is strictly to inform the recipient of valid methods associated with the resource.

Proxy-Authenticate: The Proxy-Authenticate header field must be included as part of a 407 (Proxy Authentication Required) response. The field value consists of a challenge that indicates the authentication scheme and parameters applicable to the proxy for this Request-URI.

Retry-After: The Retry-After header field can be used with a 503 (Service Unavailable) response to indicate how long the service is expected to be unavailable to the requesting client and with a 404 (Not Found), 600 (Busy), or 603 (Decline) response to indicate when the called party anticipates being available again. The value of this field can be either an SIP-date or an integer number of seconds (in decimal) after the time of the response.

Server: The Server header field contains information about the software used by the user agent server to handle the request.

Unsupported: The Unsupported header field lists the features not supported by the server.

Warning: The Warning header field is used to carry additional information about the status of a response. Warning headers are sent with responses.

WWW-Authenticate: The WWW-Authenticate header field must be included in 401 (Unauthorized) response messages. The field value consists of at least one challenge that indicates the authentication scheme(s) and parameters applicable to the Request-URI.

2.6.5 Headers used with REGISTER

In this section the REGISTER header fields are described, since the REGISTER method is somehow special.

The "address-of-record" is defined here as the SIP address that the registry knows the registrand, typically of the form "user@domain" rather than "user@host". In third-party registration, the entity issuing the request is different from the entity being registered.

To: The To header field contains the address-of-record whose registration is to be created or updated.

From: The From header field contains the address-of-record of the person responsible for the registration. For first-party registration, it is identical to the To header field value.

Request-URI: The Request-URI names the destination of the registration request, i.e. the domain of the registrar. The user name must be empty. Generally, the domains in the Request-URI and the To header field have the same value; however, it is possible to register as a ”visitor”, while maintaining one’s name. For example, a traveler sip:alice@acme.com (To) might register under the Request-URI sip:atlanta.hiayh.org, with the former as the To header field and the latter as the Request-URI. The REGISTER request is no longer forwarded once it has reached the server whose authoritative domain is the one listed in the Request-URI.

Call-ID: All registrations from a client should use the same Call-ID header value, at least within the same reboot cycle.

CSeq: Registrations with the same Call-ID must have increasing CSeq header values. However, the server does not reject out-of-order requests.

Contact: The request may contain a Contact header field; future non-REGISTER requests for the URI given in the To header field should be directed to the address(es) given in the Contact header.

Here an example, using the REGISTER method for a third-party registration. The secretary jon.diligent registers his boss, T. Watson:

```
REGISTER sip:bell-tel.com SIP/2.0
Via: SIP/2.0/UDP pluto.bell-tel.com
From: sip:jon.diligent@bell-tel.com
To: sip:watson@bell-tel.com
Call-ID: 17320@pluto.bell-tel.com
CSeq: 1 REGISTER
Contact: sip:tawatson@example.com
```

More examples can be found in [13, section 16].

2.6.6 Compact Form of SIP

In certain environment it is important, to keep the packets small (e.g. limitations in MTU[4]). For this reason the most used SIP fields have an abbreviation. A list of those can

[4]Maximum Transfer Unit

be found in table 2.5.

Short field name	Long field name
c	Content-Type
e	Content-Encoding
f	From
i	Call-ID
m	Contact (from "moved")
l	Content-Length
s	Subject
t	To
v	Via

Table 2.5: Compact form for SIP header field names

2.6.7 SIP Message Body

The message-body of a SIP message contains as payload usually (but not necessarily) a session description. In most cases it will be SDP—the Session Description Protocol [16], since SIP and SDP belong to the same protocol family. In the following chapter 3 the SDP protocol is described.

Chapter 3

Session Description Protocol (SDP)

SDP [16] is also part of the IETF conference control architecture (see chapter 2, page 5). It is intended for describing multimedia sessions for the purposes of session announcement, session invitation, and other forms of multimedia session initiation. It can be used as payload of SIP messages. But also other protocols such as SAP [14] use SDP as their session description. If used as the payload of SAP, it is typically used to announce multicast sessions.

3.1 SDP content

3.1.1 Session Information

The SDP contains information about:

- Session name and purpose
- Time(s) the session is active (see sec. 3.1.3 and table 3.2)
- The media comprising the session and how (addresses, ports, formats and so on) to receive those media (see sec. 3.1.2 and table 3.3)
- Information about the bandwidth to be used by the conference, as resources necessary to participate in a session may be limited
- Contact information for the person responsible for the session

3.1.2 Media Information

The Media information in SDP covers the range of:

- The type of media (video, audio, ...)
- The transport protocol (RTP/UDP/IP, H.320, ...)
- The format of the media (H.261 video, MPEG video, ...)
- IP address for media (unicast or multicast)
- Transport Port for media or contact address
- (Media specific) bandwidth information

The semantics of this address and port depend on the media and transport protocol defined.

3.1.3 Time Information

Sessions may either be bounded or unbounded in time. Whether or not they are bounded, they may be only active at specific times. Thus SDP can convey Timing information:

- An arbitrary list of start and stop times bounding the session
- For each bound, repeat times such as "every Wednesday at 4 a.m. for one hour"
- Time zone adjustments

3.2 SDP fields

In this section the meaning of the field names of SDP are described.

SDP fields look like <type>=<value> and are separated by CRLF.

Example

```
v=0
o=bell 53655765 2353687637 IN IP4 128.3.4.5
s=Mr. Watson, come here.
c=IN IP4 kton.bell-tel.com
m=audio 3456 RTP/AVP 0 3 4 5
```

A description consists of a session-level section followed by zero or more media-level sections. The session-level part starts with a 'v=' line and continues to the first media-level section. The media description starts with an ‘m=’ line and continues to the next media description or end of the whole session description.

Some fields can appear in the session-level as well as in media level (e.g. bandwidth information). In general, session-level values are the default for all media unless overridden by an equivalent media-level value.

When SDP is conveyed by SAP, only one session description is allowed per packet. When SDP is conveyed by other means, many SDP session descriptions may be concatenated together (the ‘v=’ line indicating the start of a session description terminates the previous description). Some lines in each description are mandatory and some optional but all must appear in exactly the order given in table 3.1. Optional items are marked with a ‘∗’.

Session description fields	∗ ≡ optional
v=	Protocol version
o=	Owner/creator and session identifier
s=	Session name
i= ∗	Session information
u= ∗	URI of description
e= ∗	Email address
p= ∗	Phone number
c= ∗	Connection information - not required if included in all media
b= ∗	Bandwidth information
One or more *time descriptions* (see table 3.2)	
z= ∗	time zone adjustments
k= ∗	encryption key
a= ∗	zero or more session attribute lines
Zero or more *media descriptions* (see table 3.3)	

Table 3.1: SDP fields

Time description fields	∗ ≡ optional
t=	time the session is active
r= ∗	zero or more repeat times

Table 3.2: SDP time description fields

Protocol Version: The ’v=’ field gives the version of the Session Description Protocol. Currently ’v=0’.

Origin: The ’o=’ field gives the originator of the session (username and address of the user’s host) plus a session id and session version number.

Media description fields		$* \equiv$ optional
m=	media name and transport address	
i= *	media title	
c= *	connection information - optional if included at session-level	
b= *	bandwidth information	
k= *	encryption key	
a= *	zero or more media attribute lines	

Table 3.3: SDP media description fields

Session Name: The 's=' field is the session name. There must be one and only one 's=' field per session description.

Session and Media Information: The 'i=' field is information about the session. There may be at most one session-level 'i=' field per session description, and at most one 'i=' field per media.

A single 'i=' field can also be used for each media definition. In media definitions, 'i=' fields are primarily intended for labeling media streams. As such, they are most likely to be useful when a single session has more than one distinct media stream of the same media type. An example would be two different whiteboards, one for slides and one for feedback and questions.

URI: The 'u=' field contains a Universal Resource Identifier as used by WWW clients. The URI should be a pointer to additional information about the conference. This field is optional, and no more than one URI field is allowed per session description.

Email Address and Phone Number: The 'e=' and 'p=' fields specify the contact information for the person responsible for the conference. This is not necessarily the same person that created the conference announcement. Either an email field or a phone field must be specified. More than one email or phone field can be given for a session description.

Phone numbers should be given in the conventional international format—preceded by a '+' and the international country code. There must be a space or a hyphen ('-') between the country code and the rest of the phone number. (e.g. p=+358-50-369 9596 or p=+1 617 253 6011)

Connection Data: The 'c=' field contains connection data.

A session announcement must contain one 'c=' field in each media description (see below) or a 'c=' field at the session-level. It may contain a session-level 'c=' field and one additional 'c=' field per media description, in which case the per-media values override the session-level settings for the relevant media.

The first sub-field is the *network type*, which is a text string giving the type of network. Initially "IN" is defined to have the meaning "Internet".

The second sub-field is the *address type*. This allows SDP to be used for sessions that are not IP based. Currently only IP4 is defined.

The third sub-field is the *connection address*. Optional extra subfields may be added after the connection address depending on the value of the 'address type' field.

Bandwidth: The 'b=' field contain information about the proposed bandwidth to be used by the session or media, and is optional.

Times, Repeat Times and Time Zones: The 't=' fields contains start and stop time, the 'r=' fields the repeat times for a session and the 'z=' (time zone) field allows the sender to specify a list of adjustment times and offsets from the base time (e.g. daylight saving time). This allows to make the necessary corrections to the different time zones and the daylight saving time.

Encryption Keys: The 'k=' fields contains the encryption method and possible information about the encryption key. This can be a URI, the key itself or and encoded key. It is also possible to prompt the user for a key.

Attributes: Attributes ('a=' fields) are the primary means for extending SDP. Attributes may be defined to be used as "session-level" attributes, "media-level" attributes, or both. Attribute interpretation depends on the media tool being invoked.

Media Announcements: A session description ('m=' field) may contain a number of media descriptions. Each media description starts with an 'm=' field, and is terminated by either the next 'm=' field or by the end of the session description. A media field also has several sub-fields:

- The first sub-field is the *media type*. Currently defined media are "audio", "video", "application", "data" and "control", though this list may be extended as new communication modalities emerge.
- The second sub-field is the *transport port* to which the media stream will be sent. The meaning of the transport port depends on the network being used as specified in the relevant 'c=' field and on the transport protocol defined in the third sub-field.

- The third sub-field is the *transport protocol.* The transport protocol values are dependent on the address-type field in the 'c=' fields. Thus a 'c=' field of IP4 defines that the transport protocol runs over IP4.

  The following transport protocols are preliminarily defined, but may be extended through registration of new protocols with IANA[1]:

  RTP/AVP: The IETF's Realtime Transport Protocol using the Audio/Video profile carried over UDP. For more details on RTP audio and video formats, see RFC 1890 [26].

  udp: User Datagram Protocol

- The fourth and subsequent sub-fields are *media formats.* For audio and video, these will normally be a media payload type as defined in the RTP Audio/Video Profile.

  When a list of payload formats is given, this implies that all of these formats may be used in the session, but the first of these formats is the default format for the session.

  For media whose transport protocol is not RTP or UDP the format field is protocol specific. Such formats should be defined in an additional specification document.

[1]IANA (Internet Assigned Numbers Authority) is the organization under the Internet Architecture Board (IAB) of the Internet Society that, under a contract from the U.S. government, has overseen the allocation of IP addresses to Internet service providers (ISPs). IANA also has had responsibility for the registry for any "unique parameters and protocol values" for Internet operation. These include port numbers, character sets, and MIME media access types.

Partly because the Internet is now a global network, the U.S. government has withdrawn its oversight of the Internet, previously contracted out to IANA, and lent its support to a newly-formed organization with global, non-government representation, the Internet Corporation for Assigned Names and Numbers (ICANN).

Chapter 4

PINT

PINT (PSTN/Internet Interfaces) [27] is a Working Group of the Internet Engineering Task Force [10]. It addresses connection arrangements through which Internet applications can request and enrich PSTN[1] telephony services. An example of such services is a Web-based Yellow Pages service with the ability to initiate PSTN calls between customers and suppliers.

This working group has six main objectives:

- Study architecture and protocols needed to support services in which a user of the Internet requests initiation of a telephone (i.e. PSTN-carried) call to a PSTN terminal (i.e. telephone, fax machine). Specific services to be considered initially are Click-to-Dial, Click-to-Fax, Click-to-Fax-Back, and Web access to voice content delivered over the PSTN.

- Produce an informational RFC[2] that describes current practices for supporting the services in question.

- Based on the existing practice and agreed on improvements, develop a standards track RFC that specifies a SSTP[3] between Internet applications or servers and PSTN Intelligent Network Service Nodes (or any other node that implement the Service Control Function).

- Consider security issues relating to providing functions of this type. In particular understand any threats posed by this technology and resolve them, and any other security issues in the proposed standard.

[1]Public Switched Telephone Network

[2]Request For Comments

[3]SSTP (Service Support Transfer Protocol) is an application-specific transport protocol operating over TCP.

- Based on the existing practice and agreed on improvements, develop a standards track RFC for a relevant MIB[4] (SSTP MIB) to support the service management protocol between Internet applications and the PSTN Service Management System. The SSTP MIB is to conform to SNMP[5] standards.
- Consider extensions of the above architecture and protocols to support a wider range of PSTN IN[6] based services.

The abbreviation PINT is used for the IETF working group and its related work, as well as for one facet of its services, which is described in the next section.

4.1 Services related to PINT Working Group

In principle two facets of interaction services are studied:

1. PINT services, which are *initiated in Internet* and *carried out in IN*
2. TNIP (reverse spelling of PINT) services, *initiated in IN* and *carried out in Internet*

Also a combination of these to facet is possible. First ideas about TNIP and combined services are published in [28].

All facets of PINT services can use different ways for input and output on telephone side. Two examples for input are:

- *DTMF[7] signals,* normally used for transmitting the dialed phone number to the exchange. Nowadays most telephones which are in use, support them. They are also used for transmitting information during the phone call. Several services are using them: Phone-banking, checking voicemail, etc.
  The advantage is the easiness to proceed this signals and the availability (almost every telephone uses them anyway). The limited charset (0–9,$*$,#) might be considered as disadvantageous.
- *Voice-to-text conversion software,* which is an application of IN.

[4]A MIB (Management Information Base) is a formal description of a set of network objects that can be managed using the SNMP (Simple Network Management Protocol). Product developers can create and register new MIB extensions at IANA (Internet Assigned Numbers Authority). More information about MIB and SNMP can be found in RFC 1155 ff.

[5]Simple Network Management Protocol

[6]Intelligent Network

[7]Dual Tone Multi Frequency

As output on telephony side, there is usually voice, recorded or synthesized speech. On way to synthesize speech is to use "text-to-speech-over-the-phone", a application of IN which is taking text as an input and reading it out to the phone. Further output methods are fax, pager or SMS[8]

4.1.1 Examples for PINT services

Here I mean *only* these kind of services, which are initiated in Internet and carried out in the telephone network.

In [1] three PINT Milestone Services are defined. These are the services, which will be supported by the first version of the PINT protocol:

R2C Request to Call (Click-to-Dial): A request is sent from an IP host that causes a phone call to be made, connecting party A to some remote party B.

R2F Request to Fax (Click-to-Fax): A request is sent from an IP host that causes a fax to be sent to fax machine B. The request must contain a pointer to the fax data (that could reside in the IP network or in the Telephone Network), *or* fax data itself.

R2HC Request to Hear Content (Click-to-Hear-Content): A request is sent from an IP host that causes a phone call to be made to user A, and for some sort of content to be spoken out. The request must either contain a URL pointing to the content, *or* include the content itself.

Call center services

A company providing Internet shopping webpages could set up it in such a way, that a user, which is browsing its pages and finds a product, he would like to get more information about, can connect his telephone to an agent in the costumer care center, just by clicking a (PINT) link.

Yellow pages

Web-based Yellow Pages service with the ability to initiate PSTN calls between customers and suppliers, just by clicking a link in the Browser. [27]

[8]Short Message Service (SMS) is a feature widely used in GSM; basically to transmit short text messages (up to 160 characters) to a mobile phone.

Email notification

Services using the telephone network for email purpose: After an email arrives, different kinds of calls can be performed:

- A automatic phone call with a artificial voice telling that an email has arrived, the user can then make a dialup session for reading it.
- A automatic phone call, where an artificial voice reads out the content of the arrived email through a text to voice unit on request.
- The email is automatically faxed to a predefined number. A combination of phone and fax call also makes sense; phone for alerting, fax for sending the content.
- The email system makes a data call to the user and uploads the email automatically to the user's home computer. Also here a combination of phone and data call makes sense.

A combination with email filters can reduce unwanted phone calls. A useful feature would also be the possibility to disable the phone calls during certain times, e.g. nights.

System administrator services

This is useful for computer systems (e.g. Unix), where it is disadvantageous to have downtimes. For such computer systems a surveillance software could detect the problem as they occur and perform:

- A phone call reporting about the problem by synthesized voice. The person in charge of the systems could react by remote access the computer systems or go onto site in order to solve it.
- A phone call so that the person in charge of the systems could perform certain actions directly in the same phone call (see also paragraph 'System administrator services' on page 38 in section 4.1.2).
- A fax is sent, containing the problem description. Also here a combination of phone and fax call makes sense. A phone call to alert the user, the fax call to get the detailed information about.

Access to voicemail

The access to voicemail through Internet belongs also to this category. This service allows the user, to play the voicemails on his computer clicking a link in the browser. So the user has more freedom in handling voicemail messages; he can either access it with the conventional method by phone or access it through Internet. Especially if there are many messages, it is much easier to handle them though Internet.

4.1.2 Examples for TNIP services

These are the services, which are initiated by telephone and executed in Internet. One advantage is, that Internet services can be accessed wherever there is a operational phone. As disadvantageous might be considered, that the handling will not be that easy and fast as if using a computer.

Internet call waiting service

In [29] an Internet call waiting service is described. A user, while surfing on the Internet and using the same phone line as for normal phone calls, gets a graphical alert on his display whenever there is an incoming call on his phone line. He can either reject, accept or divert it to voicemail.

Check email through telephone

Where there is no Internet connection available—but normal telephone—a user could inquiry his email INBOX. For this purpose he dials a certain PSTN number and follows the instruction of the synthetic voice. After this the From- and Subject-lines of the messages are read out to the telephone through a text to voice unit. He can choose which mails he wants to hear the body (content) from. The input of the user operations can be done using DTMF or just by speaking, using a voice recognition unit.

Sending email by telephone

An email could be sent by telephone using a speech to text unit or the DTMF keyboard using a predefined code, which represents the chars by numbers. '*' or '#' of the telephone keyboard could be used as char separators. An example for using ASCII code: A sequence '72*101*108*108*111*32*74*111*101*33#' would be interpreted as 'Hello Joe!'. For instructions a synthetic voice and confirmation of the input a text-to-speech unit could be used.

Surfing with only the telephone

The user could get (text-based) webpages through phone. The requested webpages would be transmitted thought a text to voice unit. One example is if someone wants to check the latest news or sports results. Some URLs could be predefined, to provide a faster access to the desired webpages. The input works similar as in the last paragraph.

System administrator services

This is useful for computer systems (e.g. Unix), where it is disadvantageous to have long down-times. The system administrator could check through his telephone, whether the systems are running normally. Some shell commands or predefined operations (e.g. reboot, checking fullness of disks, for hardware problems or aliveness of other hosts) could be executed. The input works similar as in the last two paragraphs.

4.1.3 Examples for combined services

If PINT and TNIP services are combined, the set of services can be extended remarkably. The (original) initiation could be either in IN or in Internet. Some examples are:

- The email services described in the sections 4.1.1 and 4.1.2 would allow to send selected emails to a fax, pager or by SMS in 160 char pieces to a mobile phone.
- The previously mentioned system administrator services could be extended, to get e.g. the response of a shell command immediately by fax. This allows to get selectively more information about a certain problem.
- Also the web-surfing through telephone gets more comfortable, if the pages can be requested by fax.
- Another example of a service using both of these facets might be a number portability service. A user could use a telephone to specify the telephone number at their current location (perhaps using Calling Line Identity CLI) this is sent over the Internet (using the protocol which would come out of any future work) to a repository. Another user could then attempt to telephone the first user. This call is intercepted and the number called checked against the current known location by a request (using the protocol or profile which would come out of any future work) to ascertain the number registered by the first user. If the number is different, a PINT request could be issued back to the PSTN to connect the call to the new number. [28]

In the next sections I concentrate on the PINT protocol, which is on its way to be a proposed Internet standard.

4.2 Definitions for PINT

The PINT milestone services are already defined in section 4.1.1. In the following some more definitions with special significance in PINT out of [1]:

Requestor: An Internet host from which a request for service originates.

PINT Service: A services invoked within a phone system in response to a request received from a PINT client.

PINT Client: An Internet host that sends requests for invocation of a PINT Service, as described in [1].

PINT Gateway: An Internet host that accepts requests for PINT Services and dispatches them onwards towards a telephone network.

Executive System: A system that interfaces to a telephone network that executes a PINT service, and to a PINT Server. It is not directly associated with the Internet, and is represented by the PINT Server.

Requesting User: The initiator of a request for service. This role may be distinct from that of the "party" to any telephone network call that results from the request.

(Service Call) Party: A person who is involved in a telephone network call that results from the execution of a PINT service request, or a telephone network-based resource that is involved (such as an automatic Fax Sender or a Text-to-Speech Unit).

4.3 The Architecture of PINT

A PINT system is a SIP system (see chapter 2), so that PINT clients and servers are SIP clients and servers. SIP is used to carry the request over the IP network to the correct PINT server in a secure and reliable manner, and SDP (see chapter 3) is used to describe the telephone network session that has to be invoked (or whose status has to be returned).

A PINT system consists of the following basic elements, which are described in section 4.2:

- PINT Client (SIP User Agent Client)
- PINT Gateway (SIP User Agent Server)
- Executive System

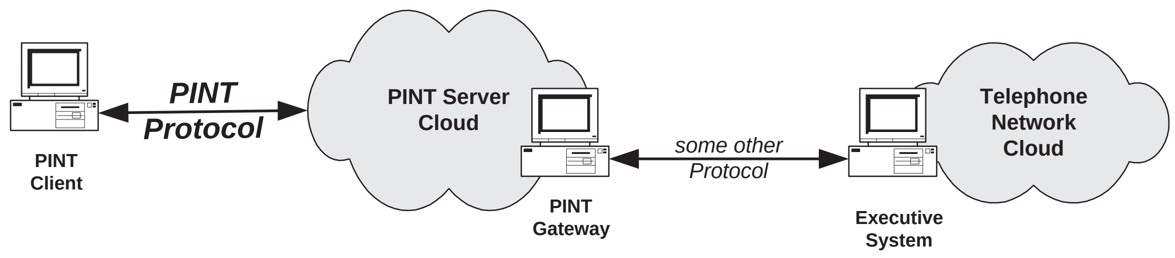


Figure 4.1: PINT interaction

Besides those there can also be SIP proxy, redirect and location servers as they are used in SIP (see sections 2.3.2 and 2.3.3).

Figure 4.1 shows a simplified case of interactions between the elements. [1, page 7]

The system of PINT servers is represented as a cloud to emphasize that a single PINT request might pass through a series of location servers, proxy servers, and redirect servers, before finally reaching the correct PINT gateway that can actually process the request by passing it to the Telephone Network Cloud.

The PINT gateway might have a true telephone network interface, or it might be connected via some other protocol or API[9] to an "Executive System" that is capable of invoking services within the telephone cloud.

The Executive System that lies beyond the PINT gateway is outside the scope of PINT. [1, page 8]

For the IN part of PINT the ITU-T is responsible. The topics concerning PINT are taken care by Study Group 11 (ITU-T SG 11), which is specifying requirements for an functional architecture that supports IN and the Internet inter-working. The work relates mainly to the PINT concept of IETF. The architecture model is an extension to the IN CS-2 functional model. It is intended to be included in the CS-4 documentation. The most important new component will be a service control gateway function, which transmits service requests and responses to them between the two networks. [5]

In general, the PINT activities on IN side are in their early states.

An ITU issue, which is affecting the PINT, is *Signaling support of services over IP-based networks (SoI).* For the IP Experts meeting in Geneva (31.8.-9.9.1999) five new questions were created and proposed, of which two concern PINT. The questions can be found in [30] For further details, please consult the reports of the mentioned meeting.

Figure 4.2 gives an overview about the PINT reference model from the IETF's point of

[9]An API (application program interface) is the specific method prescribed by a computer operating system or by another application program by which a programmer writing an application program can make requests of the operating system or another application.

view. It shows the basic architecture for the interfaces between Internet and IN. For a better understanding, also some IN internal interfaces are depicted.

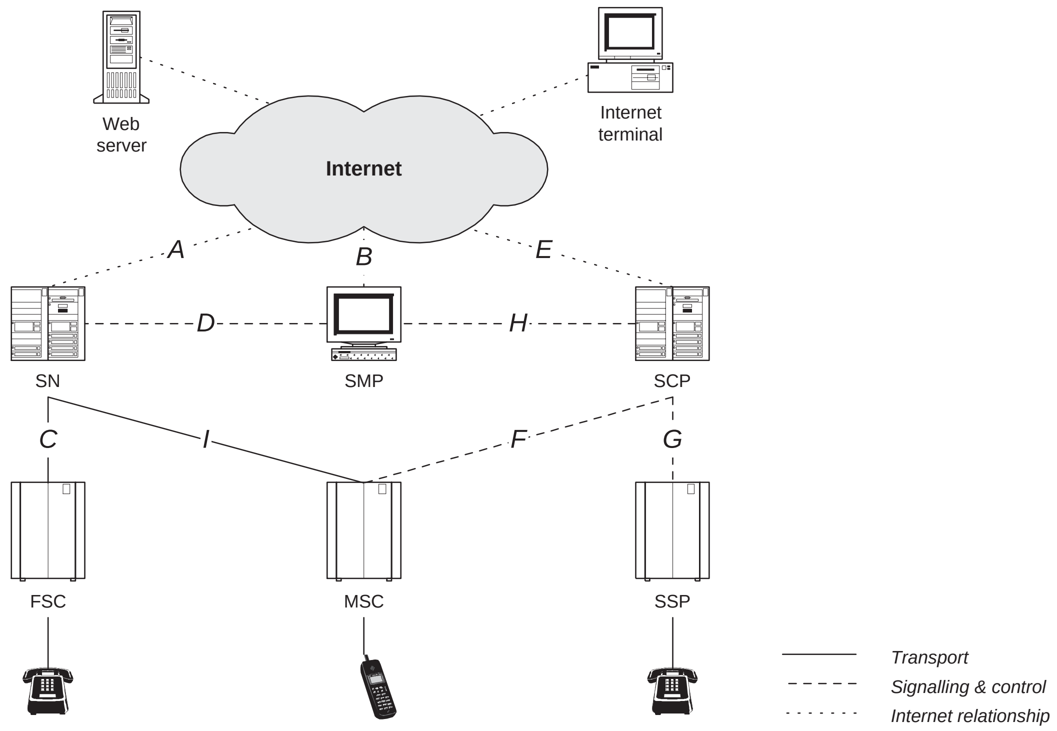


Figure 4.2: PINT reference model (IETF)

The reference model in figure 4.2 defines the following interfaces:

A The interface for delivering Internet requests for service to the Service Node (SN).

B The interface over which service management requests are carried to the Service Management Point (SMP).

C The interface for conveying call control requests from the SN to the Fixed Switching Center (FSC).

D The interface over which the SMP manages the SN.

E The interface for delivering Internet requests for service to the Service Control Point (SCP).

F The interface over which the SCP sends service call control requests to the Mobile Switching Center (MSC).

G The interface for transferring service control requests to the Service Switching Point (SSP).

H The interface over which the SMP manages the SCP.

I The interface for sending service call control requests from the SN to the MSC.

Interfaces A, B and E are going to be specified as Internet protocols by IETF, whereas the rest of the interfaces belong to the scope of ITU-T.

Also other architectures exist; these are the implementations made before the IETF WG PINT started. Most of them are described in the IETF document RFC 2458 [31].

Further information about the PINT reference model can be found in [5, page 39 ff.] as well as in [31, section 4].

Since the interconnecting parts to IN are under construction in their early states, I am not describing that topic any further here. In the following I concentrate on the Internet side of PINT.

4.4 Communication PINT Client – PINT Gateway

As mentioned above the communication in PINT relays on SIP (chapter 2). Figure 4.3 depicts an example for the PINT Milestone Service Request to Call (R2C). In this simple example there are neither proxy nor redirect servers involved.

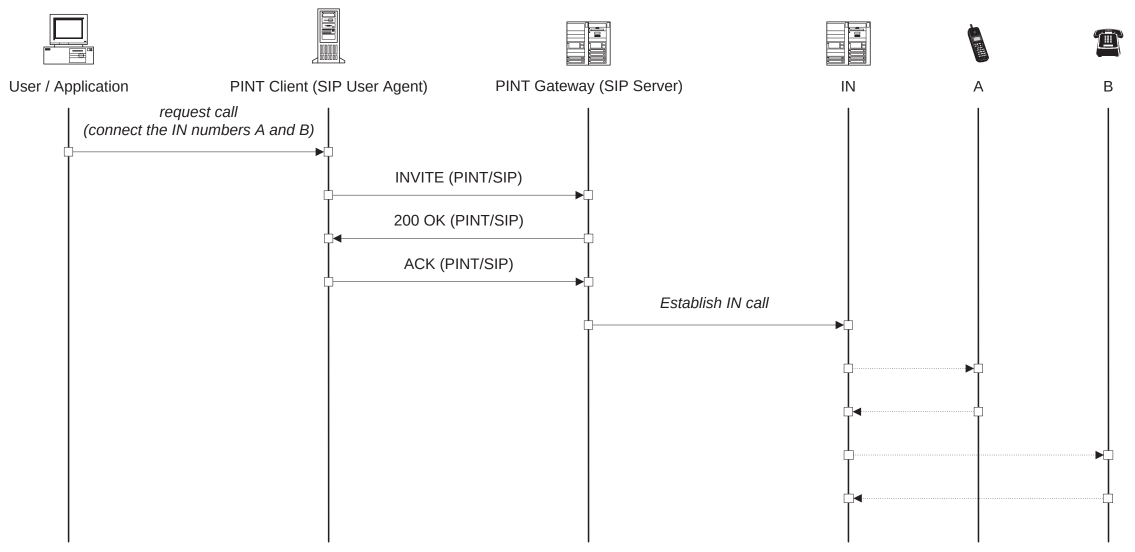


Figure 4.3: Establishing a call with PINT

The User/Application and the User Agent might be running on the same host, or even in the same program. The communication between the User Agent and the PINT Gateway uses the PINT protocol. The protocols between the PINT Gateway and the IN units are still to be defined by the IETF PINT (figure 4.2, interfaces A, B and E). The protocols in the IN part belong to the scope of ITU-T and are not discussed further here.

In figure 4.3 the exchanged signaling messages are the following:

1. A user or application wishes to connect to two parties A and B, and it requests this from its User Agent.
2. The User Agent creates the corresponding PINT message (PINT/SIP INVITE) and sends it to a PINT Gateway.
3. After the PINT Gateway has made sure that it can handle the call, it sends a final response (PINT/SIP 200 OK) (sec. 2.5.2) to the User Agent.
4. The User Agent acknowledges the reception of the final response (PINT/SIP ACK) to the PINT Gateway.
5. The PINT Gateway connects to the IN units and requests the connection of A and B.

To terminate the session either the User Agent or the PINT Gateway sends a BYE request, which has to be confirmed by the receiver.

In figure 4.3 the term IN is used more general, meaning the Executive System as well as the different IN units, involved for establishing an IN call.

In this simple example the User Agent doesn't know about the state of the call. To make this possible, the PINT protocol defines the SUBSCRIBE and NOTIFY methods. This feature is described in section 4.5.1.

4.5 PINT Extensions to SIP and SDP

For the specific needs in the telephone network, a couple of enhancements and additions to SIP and SDP are defined for PINT. They are summarized in the following sections. Sections 4.5.1–4.5.6 describe the extensions to SIP and sections 4.5.7–4.5.9 extensions to SDP.

4.5.1 SUBSCRIBE and NOTIFY methods

To get some information about the status of a call, PINT defines two new methods, additional to the methods defined in SIP (sec. 2.4.2):

SUBSCRIBE: A SUBSCRIBE request indicates that a user wishes to receive information about the status of a session.

NOTIFY: During the subscription period, the Gateway may, from time to time, send a spontaneous NOTIFY request to the entity specified in the SUBSCRIBE request. Normally this will happen as a result of any change in the status of the service session for which the Requestor has subscribed.

A simple use is, to inform the user about the state of the call, whether it has been successful. Figure 4.4 shows a simple example, where SUBSCRIBE and NOTIFY are used. The messages belonging to this feature are marked in bold face.

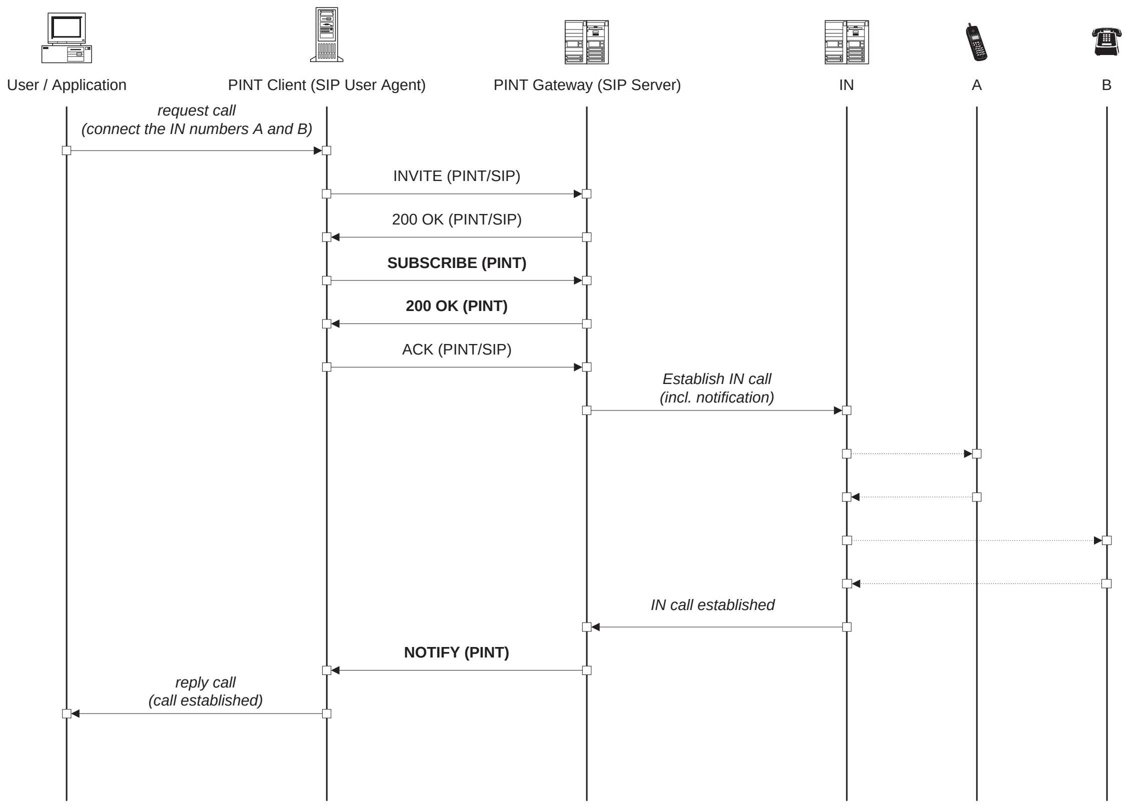

User / Application
PINT Client (SIP User Agent)
PINT Gateway (SIP Server)
IN
A
B
request call (connect the IN numbers A and B)
INVITE (PINT/SIP)
200 OK (PINT/SIP)
SUBSCRIBE (PINT)
200 OK (PINT)
ACK (PINT/SIP)
Establish IN call (incl. notification)
IN call established
NOTIFY (PINT)
reply call (call established)

Figure 4.4: Establishing a call with PINT

In figure 4.4 the exchanged messages are the following:

1. A user or application wishes to connect to two parties A and B, and it requests this from its User Agent.
2. The User Agent creates the corresponding PINT message (PINT/SIP INVITE) and sends it to a PINT Gateway.
3. After the PINT Gateway has made sure that it can handle the call, it sends a final response (PINT/SIP 200 OK) (sec. 2.5.2) to the User Agent.

4. After a response of the PINT Gateway the User Agent subscribes in order to obtain more information about the state of the call (PINT SUBSCRIBE).
5. The PINT Gateway sends a (PINT 200 OK) to the User Agent, after it has made sure, that it can handle the subscription.
6. The User Agent acknowledges the reception of the final response (for the INVITE) to the PINT Gateway (PINT/SIP ACK).
7. The PINT Gateway connects to the IN units and requests the connection of the two parties A and B.
8. After connection the two parties, the IN informs the PINT Gateway, when the connection is established.
9. The PINT Gateway informs the User Agent (PINT NOTIFY).
10. The User Agent informs its User/Application, that the call was successfully established.

In figure 4.4 the term IN is used more general, meaning the Executive System as well as the different IN units, involved for establishing an IN call (as in figure 4.3).

A further example where SUBSCRIBE and NOTIFY can be used is, to obtain information about the status of a fax transmission. As a result of a successful subscription, NOTIFY messages containing e.g. “3 pages of 5 sent” are issued.

[1, section 3.5.3]

The topic about SUBSCRIBE and NOTIFY is still in process and derived from the discussions in the PINT mailing list [32], major changes can be expected in the next version of the PINT protocol draft [1] (published on [27]).

4.5.2 Multipart MIME payloads

This allows to send data along with SIP requests, in more than one part. This could be used e.g. for sending messages, where the message length is limited. See also section 4.5.8. [1, section 3.5.1]

4.5.3 Mandatory support for *Warning* headers

A PINT server must support the SIP *Warning* header so that it can signal lack of support for individual PINT features. As an example, suppose the PINT request is to send a jpeg[10]

[10]Joint Photographic Experts Group (format for pictures allowing compression; widely used in Internet

picture to a fax machine, but the server cannot retrieve and/or translate jpeg pictures from the Internet into fax transmissions. [1, section 3.5.2]

4.5.4 *Require* headers

PINT clients use the *Require* header to signal to the PINT server that a certain PINT extension of SIP is required. Currently two strings for Require header are defined:

org.ietf.sip.subscribe means that the PINT server can fulfill SUBSCRIBE requests as described in section 4.5.1 and [1, section 3.5.3]

org.ietf.sdp.require means that the PINT server (or the SDP parser associated to it) understands the "require" attribute defined in [1, section 3.4.4]. (See also section 4.5.9, page 49.)

See also [1, section 3.5.4].

4.5.5 Format for PINT URLS within a PINT request

Normally the hostnames and domain names that appear in the PINT URLs are the internal affair of each individual PINT system. A client uses the appropriate SDP payload to indicate the particular service it wishes to invoke; it is not necessary to use a particular URL to identify the service.

A PINT URL is used in two different ways within PINT requests:

- within the Request-URI (sec. 2.4.3)
- within the To and From header fields (sec. 2.6.1)

Use within the Request-URI requires clarification in order to ensure smooth inter-working with the Telephone Network serviced by the PINT infrastructure. The following conventions for URL are offered for use in PINT requests:

1. The user portion of a SIP URL [13, section 2] indicates the service to be requested. At present the following services are defined:

R2C	for Request-to-Call
R2F	for Request-to-Fax
R2HC	for Request-to-Hear-Content

   Example: INVITE sip:**R2C**@pint.pintservice.com SIP/2.0

2. The host portion of a sip URL contains the domain name of the PINT service provider.

   Example: INVITE sip:13@**pint.telco.com** SIP/2.0

3. A new URL-parameter is defined to be *tsp*[11]. This can be used to indicate the actual TSP to be used to fulfill the PINT request.

   Example: INVITE sip:R2HC@pint.mycom.com;**tsp=pbx23.mycom.com** SIP/2.0

More information about this can be found in [1, section 3.5.5]

4.5.6 Telephone Network Parameters within PINT URLs

Any legal SIP URL [13, section 2] can appear as a PINT URL 4.5.5 within the Request-URI (sec. 2.4.3) or To header (sec. 2.6.1) of a PINT request. But if the address is a telephone address, it may be necessary to include more information in order correctly to identify the remote telephone terminal or service. PINT clients may include these attribute tags within PINT URLs if they are necessary or a useful complement to the telephone number within the SIP URL. These attribute tags must be included as URL parameters as defined in [13] (i.e. in the semi-colon separated manner). [1, section 3.5.6]

4.5.7 New network and address types

PINT uses a new network type "TN" and address types "RFC2543" and "X- . . . ". Network and address types are part of the SDP connection field "c=", introduced on page 31.

The TN ("Telephone Network") network type is used to indicate that the terminal is connected to a telephone network.

The address types allowed for network type TN are "RFC2543" and private address types, which MUST begin with an "X-".

[1, section 3.4.1]

4.5.8 New media types, transport protocols and format types

If PINT uses new media types "text", "image", and "application", and with the Network Type "TN" (sec. 4.5.7) new protocol transport keywords "voice"[12], "fax" and "pager"

[11] for "telephone service provider"

[12] the authors of [1] didn't use consequently "voice" throughout the document. In the ABNF [33] section of [1, Appendix A], there is "phone" defined, (I assume) instead of "voice". I reported this to the authors of the document.

and the associated format types and attribute tags. Media types, transport protocols and format types are part of the SDP media "m=" field introduced on page 31. [1, section 3.4.2]

Furthermore new format specific attributes for included content data are used. As an alternative to pointing to the data e.g. via a URI, it is possible to include the content data within the SIP request itself. This is done by using multipart MIME for the SIP payload. The first MIME part contains the SDP description of the telephone network session to be executed. The other MIME parts contain the content data to be transported. [1, section 3.4.2.4]

4.5.9 New Attribute Tags

In PINT several new attribute tags are defined in order to pass information to the telephone network. It may be desired to include within the PINT request service parameters that can be understood only by some entity in the "Telephone Network Cloud". SDP attribute parameters are used for this purpose. They may appear within a particular media description or outside of a media description.

These attributes may also appear as parameters within PINT URLs [1, section 3.5.6] as part of a SIP request.

This is necessary so that telephone terminals that require the attributes to be defined can appear within the To header field (sec. 2.6.1) of a PINT request as well as within PINT session descriptions.

Phone-context attribute: An attribute is specified to enable "remote local dialing". This is the service that allows a PINT client to reach a number from far outside the area or network that can usually reach the number. It is useful when the sending or receiving address is only dialable within some local context, which may be remote to the origin of the PINT client.

For example, if Alice wanted to report a problem with her telephone, she might then dial a "network wide" customer care number; within the Swisscom network in Switzerland, this is "175". Note that in this case she doesn't dial any trunk prefix—this is the whole dialable number. If dialed from another operator's network, it will not connect to Swisscom's Engineering Enquiries service; and dialing "+41 175" will not normally succeed. Such numbers are called Network-Specific Service Numbers.

[1, section 3.4.3.1]

Presentation Restriction attribute: Although it has no affect on the transport of the service request through the IP Network, there may be a requirement to allow originators of a PINT service request to indicate whether or not they wish the "B party" in the resulting service call to be presented with the "A party's" calling telephone number. It is a legal requirement in some jurisdictions that a caller be able to select whether or not their correspondent can find out the calling telephone number (using Automatic Number Indication or Caller Display or Calling Line Identity Presentation equipment). Thus an attribute may be needed to indicate the originator's preference. [1, section 3.4.3.2]

ITU-T CalledPartyAddress attributes parameters: These attributes correspond to fields that appear within the ITU-T Q.763 "CalledPartyAddress" field (see [34, section 3.9]). PINT clients use these attributes in order to specify further parameters relating to Terminal Addresses, in the case when the address indicates a "local-phone-number". In the case that the PINT request contains a reference to a PSTN terminal, the parameters may be required to correctly identify that remote terminal. [1, section 3.4.3.3]

Require: A new attribute tag "require" is used by a client to indicate that some attribute is required to be supported in the server. According to the SDP specification, a PINT server is allowed simply to ignore attribute parameters that it does not understand. In order to force a server to fail a request if it does not understand one of the PINT attributes, "require" attribute are used.

The "require" attribute may appear anywhere in the session description, and any number of times, but it must appear before the use of the attribute marked as required.

Since the "require" attribute is itself an attribute, the SIP specification allows a server that does not understand the require attribute to ignore it. In order to ensure that the PINT server will comply with the "require" attribute, a PINT client should include a Require header with the tag "ietf.org.sdp.require" (sec. 4.5.4).

[1, section 3.4.4]

4.6 Parameter Mapping to PINT Extensions

In [1, section 6.5–6.6] a possible way for the parameter mapping to the PINT Extensions is described. This means the way, in which the parameters, needed to specify a PSTN service request fully, might be carried within a "PINT extended" message. There are other choices, and these are not precluded.

The Service Identifier can be sent as the userinfo element (R2C, R2F or R2HC) of the Request-URI, as described in section 4.5.5 on page 46. Such a PINT URL would look like the following:

```
INVITE <serviceID>@<pint-server>.<domain>  SIP/2.0
```

The A Party for the R2C service can be held in the To header field. In this case the To header value will be different from the Request-URI. In the services where the A party is not specified, the To field is free to repeat the value held in the Request-URI. This is the case for R2F and R2HC services.

The B party is needed in all PINT milestone services, and can be held in the enclosed SDP sub-part, as the third sub-field (connection address) of the “c=” field (see also page 31 and section 4.5.7).

The call format parameter can be held as the third subfield of the SDP “m=” field, which maps to the “transport protocol” element (“voice”, “fax” or “pager”) as described on page 31 and in section 4.5.8.

The source format specifier can be held in the first subfield (media type) of the SDP “m=” field. PINT defines “audio”, “text”, “image” or “application”. The media format sub-type, which appears as the fourth and subsequent sub-fields of the SDP “m=” field, is required for all services. In some cases e.g. R2C is has no meaning and a “-” is inserted instead[13]. Other possible media formats are defined in RFC 2046 [35]. More about source format parameter in PINT can be found in section 4.5.8.

The source itself is identified by an “a=fmtp:” field value, where needed. It is used to give more information how the media format(s) at the end of the “m=” field can be accessed. It might be an URI or a reference to an IN resource (opaque reference).

In summary, the parameters needed by the different services, are carried in fields as shown in table 4.1:

Service Parameter	**SIP or SDP**	**PINT/SIP header or SDP field used**	**Example value**		
			R2C	**R2F**	**R2HC**
Service-ID	SIP	Request-URI (userinfo)	R2C	R2F	R2HC
B-Party	SIP	To header field	`sip:123@p.com`	`sip:1-730-1234@p.com` (not used)	`sip:R2HC@pint.abc.net` (not used)
A-Party	SDP	3rd sub-field of “c=” field	TN RFC2543 4567	TN RFCxxx +441213553	TN RFCxxx +441213554
Call Format	SDP	3rd sub-field of “m=” field	voice	fax	voice
Source Format	SDP	1st sub-field of “m=” field	audio	image	text
		4th and subsequent sub-fields of “m=” field	- (not used)	jpeg	html
Source	SDP	“a=fmtp:” field qualifying preceding “m=” field	(not used)	a=fmtp:jpeg opr:1234 or: a=fmtp:jpeg <uri-ref>	a=fmtp:html <uri-ref>

Table 4.1: Parameter mapping in PINT

[13]In the SDP specification it is mandatory to have at least one media format

4.7 Examples

RFC 2327 [16] demands that the SDP fields “s=”, “t=” and either “e=” or “p=” are mandatory (see sec. 3.2). In the current version of the PINT protocol [1] the examples do not contain them, since they don’t make much sense in the PINT context. I reported this conflict to the PINT mailing list [32]. Until now it has not been defined, how to treat this problem. That is why those are also omitted in the following examples.

Furthermore in the examples of [1] there is no CSeq header field, although SIP demands this. I reported also this mistake to the PINT mailing list.

4.7.1 R2C

```
INVITE sip:R2C@pint.nokia.com  SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
From: sip:anon-1827631872@chinet.net
To: sip:+1-201-456-7890@iron.org;user=phone
Call-ID: 19971205T234505.56.78@chinet.net
CSeq: 1 INVITE
Subject: Information about Mobile Phone
Content-type: application/sdp
Content-Length: 106

v=0
o=- 53655765 2353687637 IN IP4 128.3.4.5
i=Model 7110
c=TN  RFC2543  +1-201-406-4090
m=audio 1  voice -
```

4.7.2 R2F

```
INVITE sip:R2F@pint.nokia.com  SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
From: sip:john.jones.3@chinet.net
To: sip:R2F@pint.nokia.com
Call-ID: 19971205T234505.66.79@chinet.net
CSeq: 1 INVITE
Content-type: application/sdp
Content-Length: 133

v=0
o=- 53655768  2353687637 IN IP4 128.3.4.5
c= TN  RFC2543  1-201-406-4091
m=image 1 fax jpeg
a=fmtp:jpeg uri:http://www.nokia.com/Products/MobilePhones/7110.jpeg
```

4.7.3 R2HC

```
INVITE sip:R2HC@pint.nokia.com  SIP/2.0
Via: SIP/2.0/UDP 169.130.12.5
From: sip:john.jones.3@chinet.net
To: sip:R2HC@pint.nokia.com
Call-ID: 19971205T234505.66.99@chinet.net
CSeq: 1 INVITE
Content-type: application/sdp
Content-Length: 147

v=0
o=- 53655768  2353687637 IN IP4 128.3.4.5
c= TN  RFC2543  1-201-406-4088
m=text 1 voice plain
a=fmtp:plain uri:http://www.nokia.com/Products/MobilePhones/7110.txt
```

Chapter 5

Application using the PINT protocol

This chapter describes the practical part of the thesis. It contains a prototype application, which is using the PINT protocol. In section 5.5 the corresponding Parser application is described.

5.1 Description

The implementation is close to figure 4.3, but not exactly the same. The main difference between this application and figure 4.3 is, that the IN Emulator is informing the PINT Gateway about the state of the connection, which then informs the User Agent. Different is also that after receiving an INVITE request the PINT Gateway asks the IN Emulator, whether the two numbers (A and B party) can be connected. This scenario is depicted in figure 5.1. It shows the case, where the the call can be established. The termination is done by the IN Emulator (which emulates a normal "hang up").

How to install and run the application is described in appendix B and C.

5.2 FSM

The Application is built with four Finite State Machines (FSM). The inputs are "messages", which are coming from another FSM or from the keyboard. The outputs are messages to other FSM or the display of the User.

The syntax used in the FSM is: *INPUT / OUTPUT*

OUTPUT consists of zero or more output messages. Figure 5.2 depicts the interactions between the state machines.

Table 5.1 shows, which meaning the processes have.

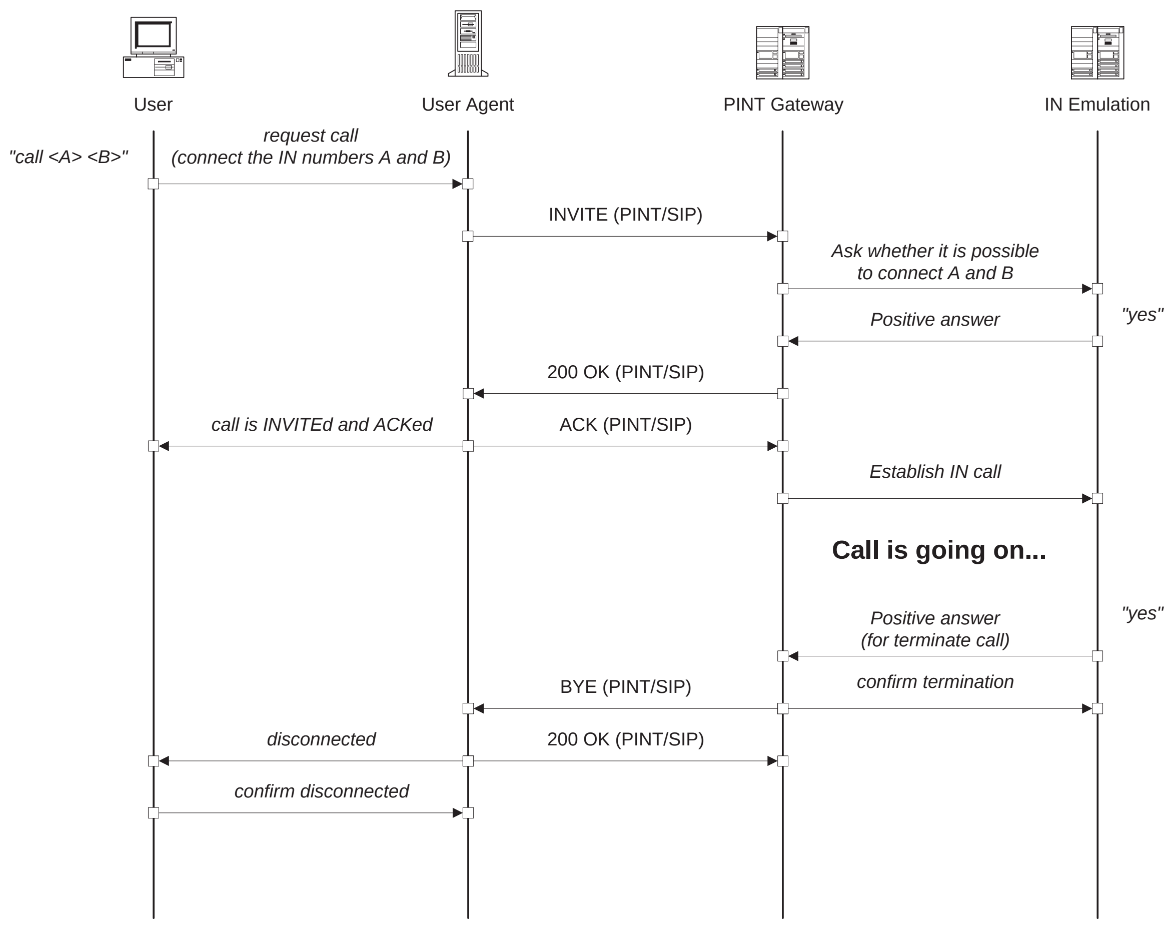


Figure 5.1: Timing diagram of the application

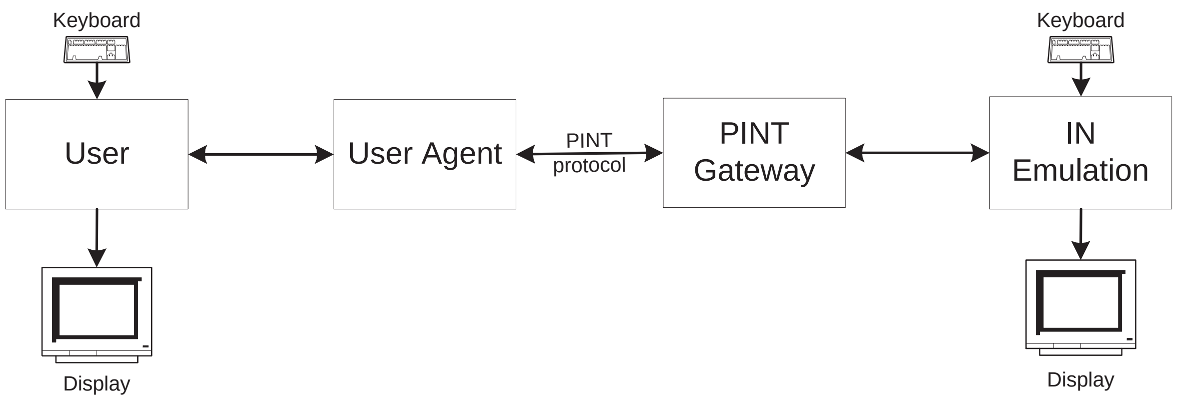


Figure 5.2: Interaction of Processes

Process name	**Short description**
User	Interface between (human) user and the User Agent
User Agent	Communicates with the PINT Gateway using the PINT protocol, responsible for PINT communication, so that the User does not need to care about PINT protocol
PINT Gateway	Communicates with the User Agent using the PINT protocol. Invokes the IN calls trough an Executive System, which is somehow connected to the IN
IN Emulation	This is to emulate the telephone network cloud

Table 5.1: Processes in the application

5.2.1 User

In the following *User* (with capital letter 'U') stands for the User process, and *user* (with small letter 'u') stands for a human user which using keyboard and display in this application.

The User process takes input lines from the keyboard and requests the order of the user from its User Agent. It also gets information from the User Agent and prints it to the display of the user. This process is meant to simulate an application, which requests services through a User Agent. In this prototype application its functionality is just to request a call from its User Agent on behalf of the user.

Figure 5.3 shows the FSM of the User process. Its exchanged messages have the following meaning:

- **Input Messages**

  STDIN: call The (human) user wants to connect two numbers A and B.

  UA: callInvAndAck The User Agent informs the User, that the connection is INVITEd and ACKed.

  UA: disconnected The User Agent informs the User, that the call has been disconnected.

  STDIN: disconnect The user wants to disconnect the call.

  UA: replyDisconnect This is the confirmation to the User Agent for a "requestDisconnect"

- **Output Messages**

  UA: requestCall Request of the PINT service R2C from the User Agent.

  UA: requestDisconnect Request the termination of the call from the User Agent.

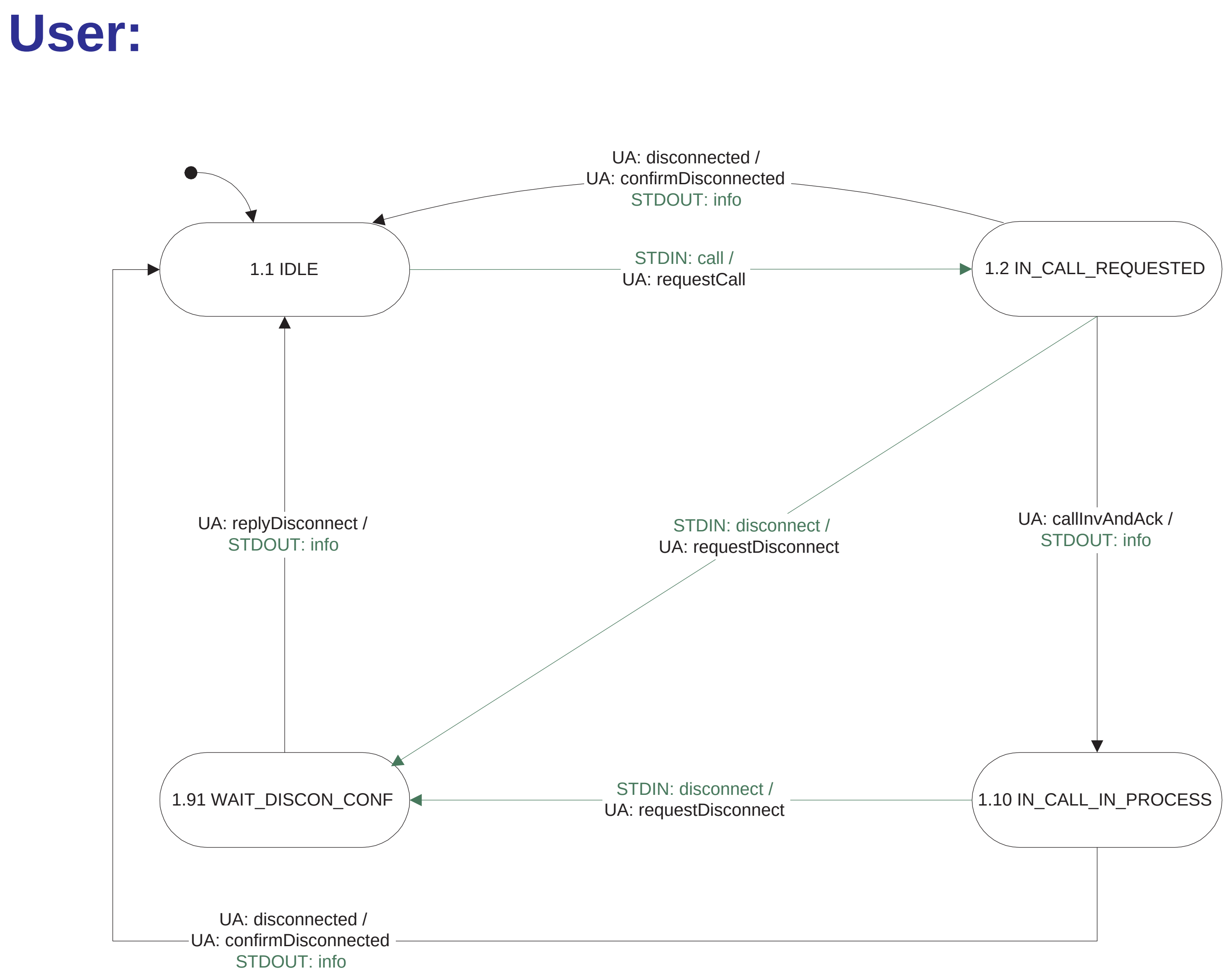


Figure 5.3: FSM for User process

UA: confirmDisconnected This is the confirmation to the User Agent for a "disconnected".

STDOUT: info means, that some information for the (human) user is printed to the display.

5.2.2 User Agent

The process User Agent knows, how to communicate with the PINT gateway using the PINT protocol. The User Agent provides the initiation of the PINT services on behalf of the User. User and User Agent might be running on the same machine, or even be part of the same application.

Figure 5.4 shows the FSM of the User Agent process. Its exchanged messages have the following meaning:

- **Input Messages**

  User: requestCall The PINT service R2C is requested by the User.

  User: requestDisconnect The User asks to terminate the call.

  User: confirmDisconnected This is the confirmation from the User for a "disconnected".

  PG: replyInvite The PINT Gateway sends a positive response for the previous INVITE request (using the PINT protocol).

  PG: requestBye The PINT Gateway sends a BYE request (using the PINT protocol).

  Remark: In this application prototype, also a negative response for an INVITE is indicated by a BYE, which is a simplification of the PINT protocol and *not* standard.

  PG: replyBye The PINT Gateway sends a positive response for a previous BYE request (using the PINT protocol).

- **Output Messages**

  User: callInvAndAck The User gets informed, that the connection is INVITEd and ACKed.

  User: disconnected The User gets informed, that the connection was shut down.

  User: replyDisconnect This is the confirmation to the User for the "requestDisconnect"

  PG: requestInvite Send an INVITE request to the PINT Gateway (using the PINT protocol).

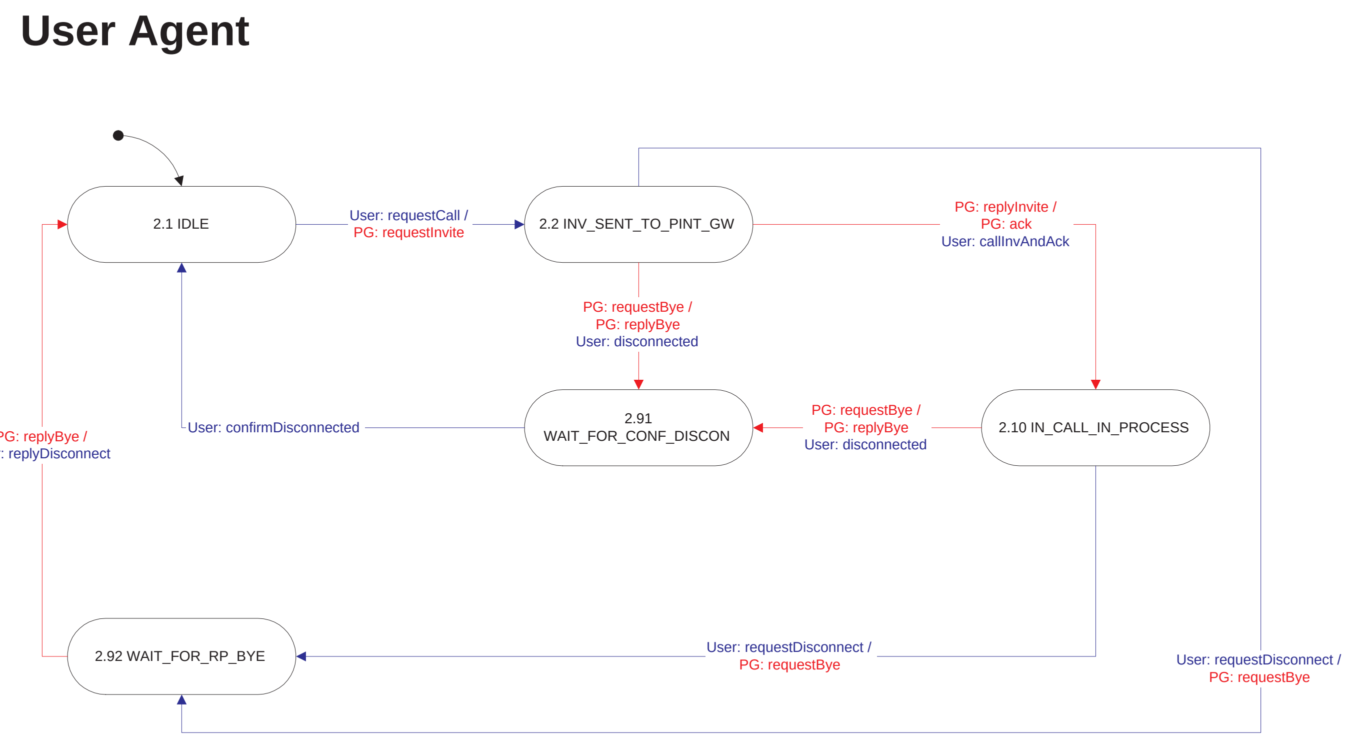


Figure 5.4: FSM for User Agent Process

PG: ack Send an ACK to the PINT Gateway (using the PINT protocol), in order to confirm the reception of the positive response.

PG: requestBye Send a BYE request to the PINT Gateway (using the PINT protocol).

PG: replyBye Send a positive response for a BYE request to the PINT Gateway (using the PINT protocol).

5.2.3 PINT Gateway

The PINT Gateway process accepts requests from User Agents. These requests are transferred by the PINT protocol. The PINT Gateway knows, how to communicate with the Executive system, which is interfaced to the telephone network, in order to execute PINT services.

Figure 5.5 shows the FSM of the PINT Gateway process. Its exchanged messages have the following meaning:

- **Input Messages**

  UA: requestInvite INVITE request from the User Agent (using the PINT protocol).

  UA: ack ACK from the User Agent (using the PINT protocol).

  UA: requestBye BYE request from the User Agent (using the PINT protocol).

  UA: replyBye Positive response for a BYE request from the User Agent (using the PINT protocol).

  INE: posReplyPrepareCall The IN Emulation informs that the call can be established.

  INE: negReplyPrepareCall The IN Emulation informs that the call can *not* be established.

  INE: requestBye The IN Emulation terminated the call.

  INE: replyBye The IN Emulation confirms the termination of the call.

- **Output Messages**

  UA: replyInvite Send a positive response for the INVITE request to the User Agent (using the PINT protocol).

  UA: requestBye Send a BYE request to the User Agent (using the PINT protocol).

  Remark: In this application prototype, also a negative response for an INVITE is indicated by a BYE, which is a simplification of the PINT protocol and *not* standard.

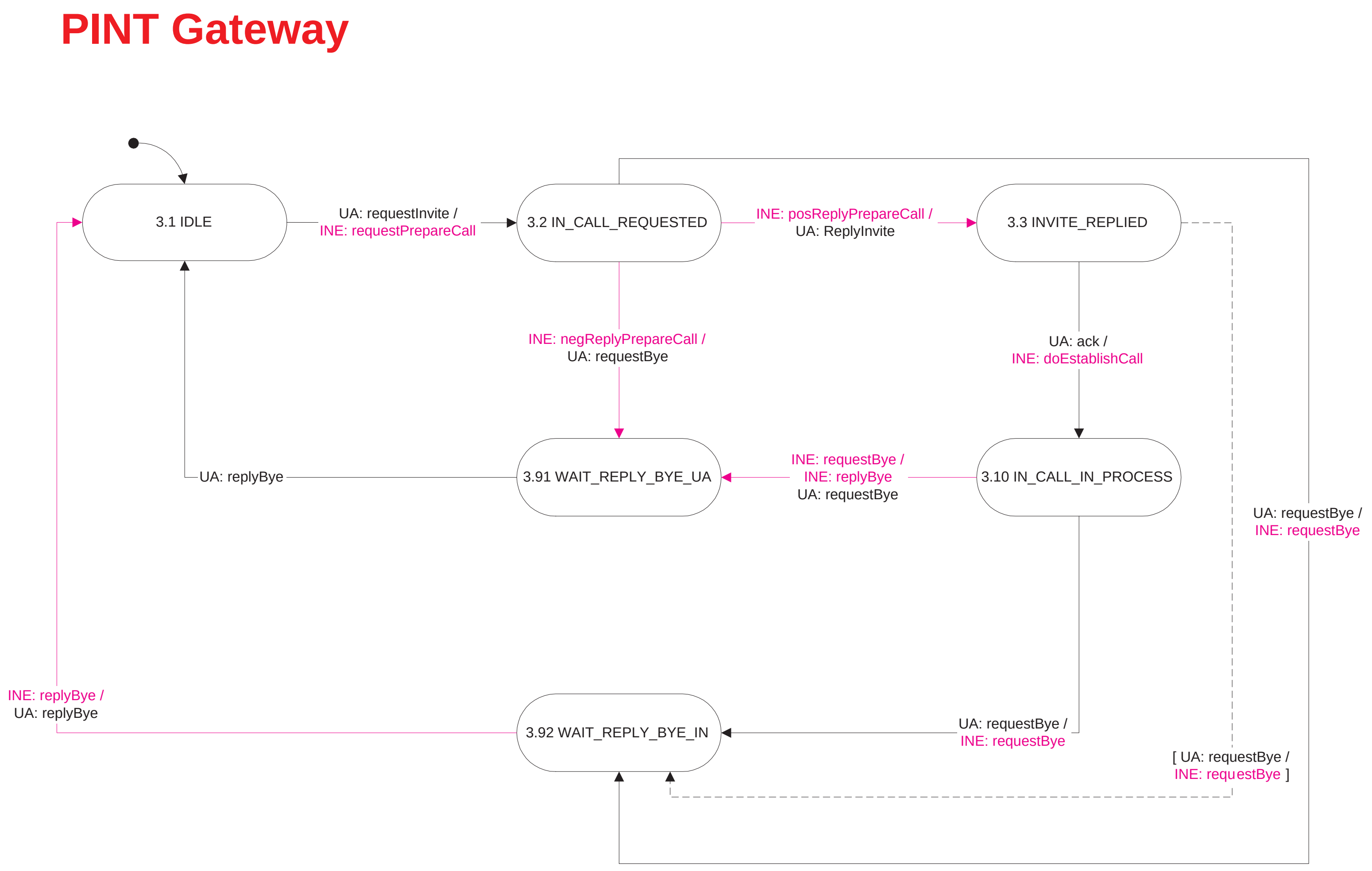


Figure 5.5: FSM for PINT Gateway Process

UA: replyBye Send a positive response for a previous BYE request to the User Agent (using the PINT protocol).

INE: requestPrepareCall The IN Emulation is asked to check out, whether the call can be established or not.

INE: doEstablishCall The IN Emulation is asked to establish the call.

INE: requestBye The IN Emulation is asked to terminate to call.

INE: replyBye Confirmation to the IN Emulation, that the "requestBye" was received.

5.2.4 IN Emulation

Since the protocols to connect Internet and IN do not exist yet, the whole Telephone Network Cloud (figure 4.1) including the Executive System is emulated by this process. In this prototype application it is taking requests form the PINT Gateway and printing messages to the screen to which the "(human) emulation operator" can react to by keyboard. Currently only "yes" and "no" are used as input from the keyboard.

Figure 5.6 shows the FSM of the IN Emulator process. Its exchanged messages have the following meaning:

- **Input Messages**

  PG: requestPrepareCall The PINT Gateway asks to check out, whether the call can be established or not.

  PG: doEstablishCall The PINT Gateway asks to establish the call.

  PG: requestBye The PINT Gateway asks to terminate to call.

  PG: replyBye The PINT Gateway confirms, that "requestBye" was received.

- **Output Messages**

  The output messages of the IN Emulator have the following meaning:

  PG: posReplyPrepareCall Inform the PINT Gateway that the call can be established.

  PG: negReplyPrepareCall Inform the PINT Gateway that the call can *not* be established.

  PG: requestBye Inform the PINT Gateway that the call is terminated.

  PG: replyBye Confirms the termination of the call to the PINT Gateway.

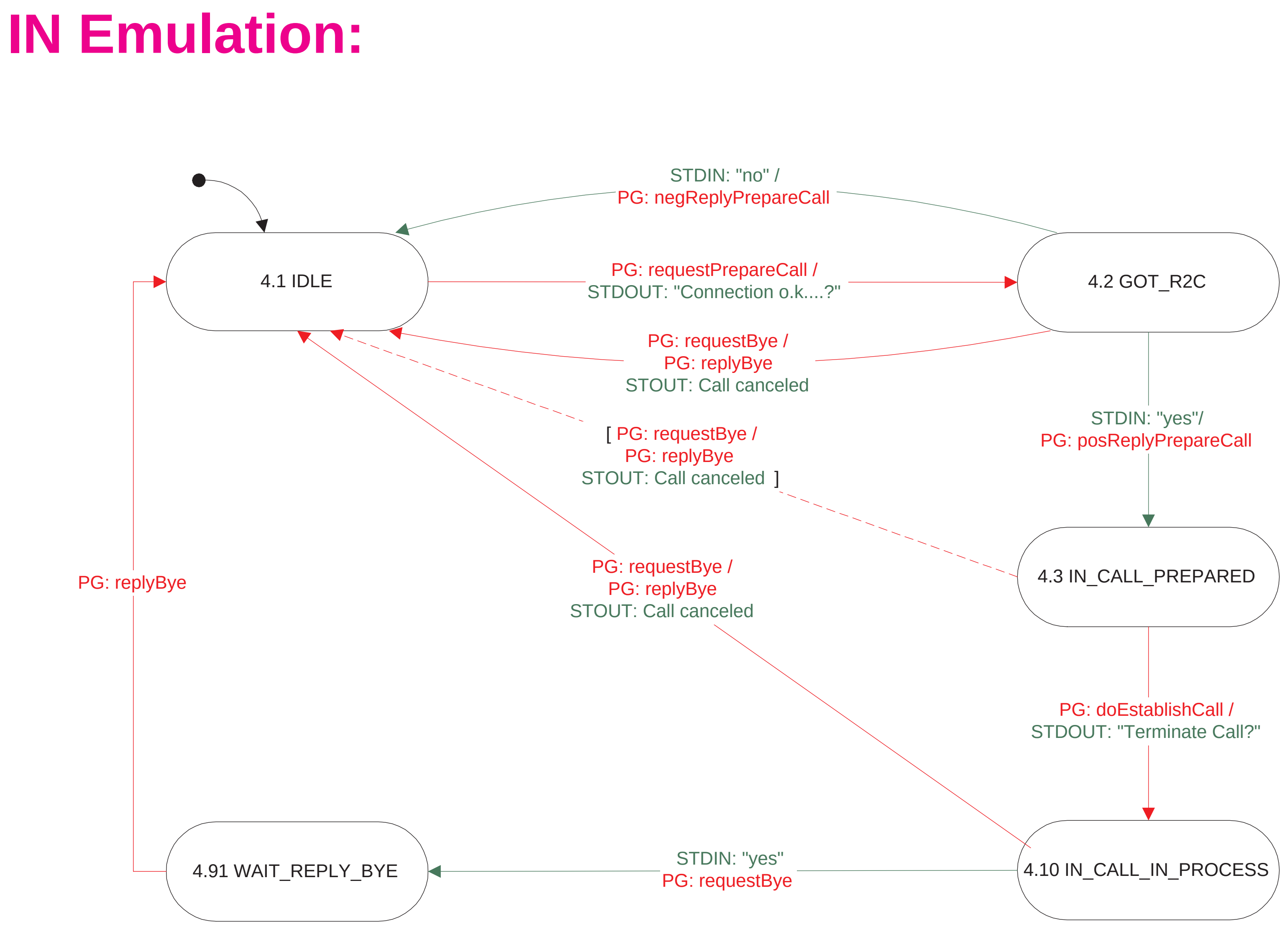


Figure 5.6: FSM for IN Emulation Process

5.3 Java Overview

For the implementation I choose Java (JDK 1.2.1.) Java is platform independent and allows Exception Throwing, which makes the debugging much easier, than for example in C. It is also quite strict, so that most of the programming mistakes are indicated by the compiler. For more information about Java, please consult the Java related literature e.g. [36] or the Java Homepage [37].

5.3.1 Static Overview

In Figure 5.7 there is a static overview of the Java classes, which I have implemented. It shows the static dependencies between the classes. The Parser classes are described in section 5.5.3.

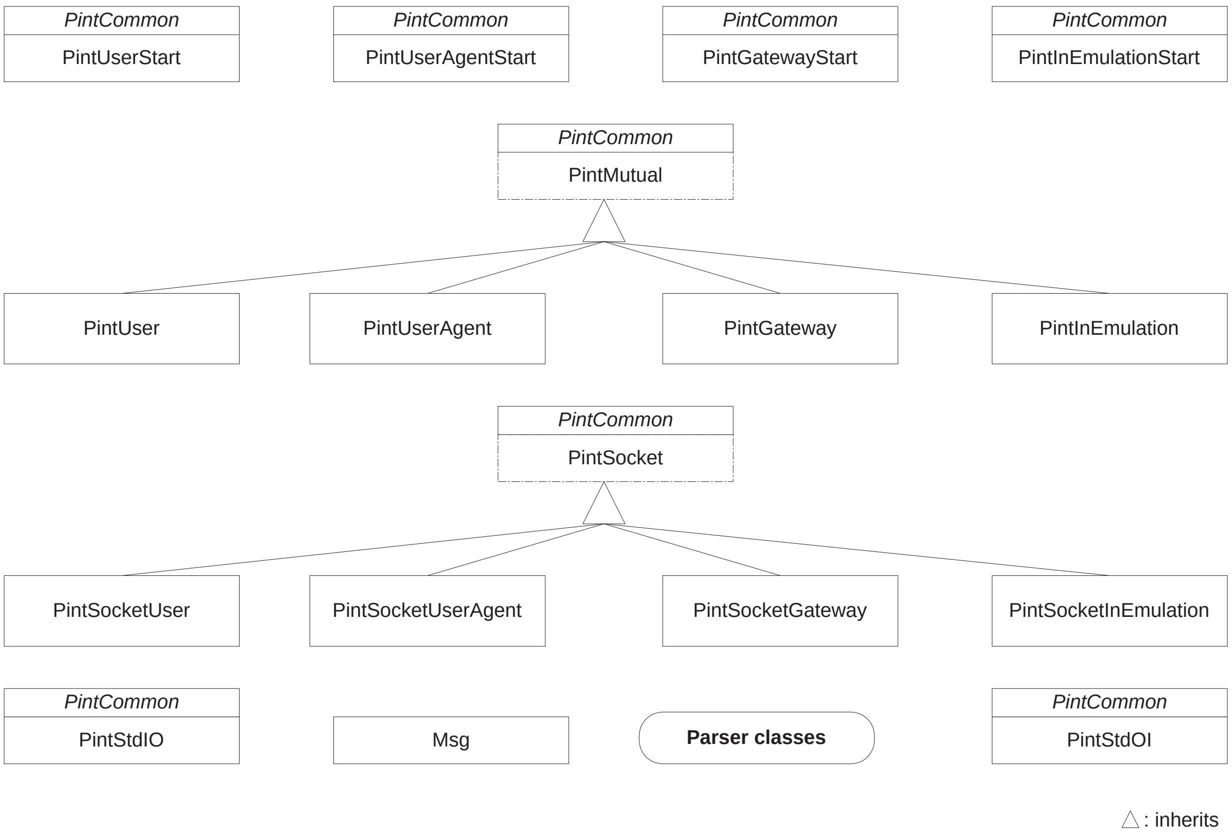


Figure 5.7: Static Overview of the Java Classes

Java provides some mechanism to make interfaces, which are implemented by other classes. *PintCommon* is such an interface. The classes *PintMutual* and *PintSocket* are abstract classes and thus not used directly in the application. They are extended by other classes.

this is another mechanism Java provides. In order to group the message and information about the sender in the same object, I wrote the class *Msg*.

5.3.2 Dynamic Overview

In Figure 5.8 the dynamic overview of the classes is depicted.

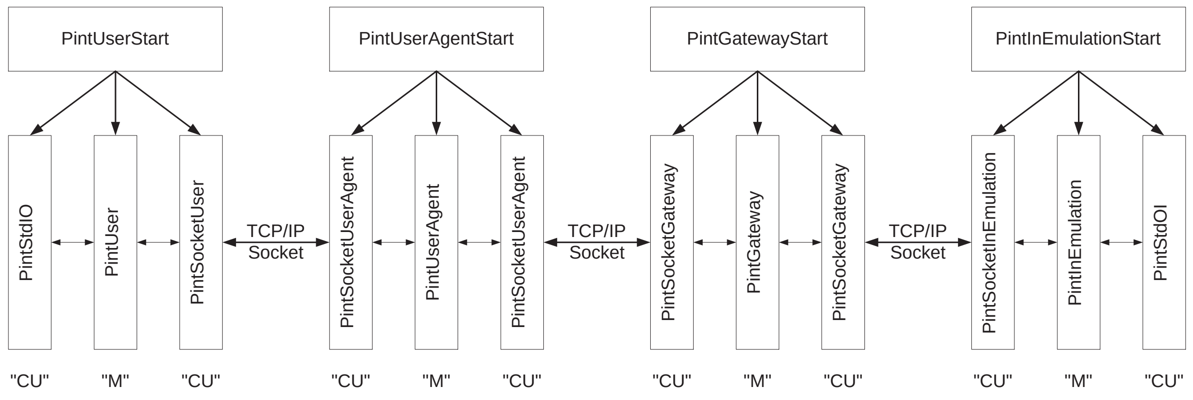


Figure 5.8: Dynamic Overview of the Java Classes

The four processes start three threads each. The middle thread contains the main program (“M”), the FSM and the corresponding preprocessing units. The left and the right threads of a process are the communication units (“CU”) to the other processes. The threads *PintStdIO* and *PINTStdOI* are for interaction with a human user, writing to standard output (display) and reading from standard input (keyboard). The threads containing the string ”Socket” in the class name are connected through a TCP/IP socket to the other processes, on the left side there are the server sockets, on the right side the client sockets. When a complete message arrives at a “CU”, it invokes the “M”, which processes the message and as a result the “M” performs the change of the FSM state and usually invokes one or both “CU”, in order to send the outgoing message(s).

5.4 Description of the Java Classes

This section describes the Java classes and their methods. In the whole sections the abbreviation “M” stands for Main thread and “CU” for Communication Unit thread, as just introduced in section 5.3.2.

5.4.1 PintCommon

PintCommon is an interface class, containing global definitions:

- Definition of the integers for the occurring senders (the four processes and the keyboard)
- Default values for the hostnames of each process
- Default values for the ports of each process
- Definition of the integers for SERVER and CLIENT

This interface is implemented by almost all classes, sometimes through an abstract class. Only *Msg* and the classes that belong to the parser do not use this interface.

5.4.2 Msg

The *class Msg* is an auxiliary class in order to provide a structure for grouping the message and information about the sender.

5.4.3 PintMutual

The *abstract class PintMutual extends Thread implements PintCommon* is the base object for all the "M" threads.

PintMutual
cache sessMem

put()
abstract proceedInput()
fillInCommand()
fillInAB()
run()

cache

The *Vector cache* is a FIFO[1]-Buffer, used to provide a causal ordering[2], which means that the messages are processed in the same order as they arrive. The *cache* is also a buffer for all arriving messages.

[1]First In First Out
[2]also known as happened-before ordering

sessMem

The *Hashtable sessMem* is used, to save the variables belonging to the session. Some information has to be saved, for example because the session description is only sent with an INVITE request, but later still used e.g. after the ACK.

put()

The *void put(Msg msg)* method is invoked by either of the"CU". It adds the received message to *cache* and interrupts (wakes up) the sleeping *run()* method, which is responsible that the incoming message is processed further.

abstract proceedInput()

The *abstract void proceedInput(int sender, String str) throws Exception* method is described further in the classes, which implement it. It has to be in this class as predefinition, because the compiler wouldn't accept its absence.

fillInCommand()

This method is only used, if the received message was in PINT protocol.

The *void fillInCommand(Hashtable ht) throws Exception* method is only used, if the received message was in PINT protocol. This method does a preprocessing for the received messages. It is invoked by "M", after parsing the PINT message. As input it gets the result of the parser 5.5 in the Hashtable. In this method a key "Command" is added to the Hashtable. This key is evaluated later in the *feedFsm()* method. The content of the key "Command" is equal to keywords which appear as IN- and OUTPUTs of the FSM described in section 5.2. To generate the "Command" it evaluates the method in the Request-Line (sec. 2.4.1). In case it is a response[3], it evaluates, the Status-Code (sec. 2.5.2) in the Status-Line (sec. 2.5.1) and the method component in the CSeq header field (page 20).

fillInAB()

The *void fillInAB(Hashtable ht) throws Exception* method is only used, if the received message was in PINT protocol. This method is invoke by the *fillInCommand()* method for extracting the the A and B parties of a requested phone call out of the SDP fields (see sec. 4.6) and update the *sessMem.*

[3]If the parser recognizes a response it fills the method field of the Hashtable with the string "RESPONSE".

run()

The *public void run()* method is the entry point of a thread, invoked by the *start* method. It checks whether there are message waiting in *cache* to be processed. If there are waiting messages, it takes the next one and invokes the *proceedInput()* method. If there are no messages waiting, is goes to a sleep() state, until an interrupt() in the *put()* method wakes it up. This is made, to avoid endless running (consuming processor time), while checking all time the *cache* for new entries.

5.4.4 PintUser

The *public class PintUser extends PintMutual* method is the "M" of the User process. It is a thread, which is initialized and started by the *main()* method of the *PintUserStart* class. It defines integers for the different states. Those correlate with the values used in figure 5.3.

PintUser
state stdIO ua

init()
PintUser()
proceedInput()
feedFsm()

state

The *static int state* represents the current state of the FSM (see figure 5.3).

stdIO

The *PintStdIO stdIO* is used to access the methods in the "CU" class, which takes input from the keyboard and writes the output to the display.

ua

The *PintSocketUser ua* is used to access the methods in the other "CU" class, which connects as client to the corresponding "CU" of the User Agent process by TCP/IP socket.

init()

The *void init(PintStdIO stdIO, PintSocketUser ua)* method is used in the initialization phase. It is invoked by the *main()* method of the *PintUserStart* class after the *main()* knows the reference to the two "CU". This method initializes the "CU" references in this thread and creates the *Vector cache.*

PintUser()

PintUser() is the constructor of this class and is empty at the moment.

proceedInput()

The *void proceedInput(int sender, String str) throws Exception* method is invoked by the *run()* method after a new message arrived. Depending on the sender of the message it calls the corresponding parser and invokes the *feedFsm()* method to perform the FSM.

feedFsm()

The *protected void feedFsm(int sender, Hashtable recv) throws Exception* method performs the assigned FSM. Depending on the content of "Command" in the *Hashtable recv* and the sender, it generates the corresponding zero, one or two output messages and changes to the new state. The sending of the output messages goes through the corresponding "CU". The FSM is described in section 5.2.1.

5.4.5 PintUserAgent

The *public class PintUserAgent extends PintMutual* is the "M" of the UserAgent process. It is a thread, which is initialized and started by the *main()* method of the *PintUserAgentStart* class. It defines integers for the different states. Those correlate with the values used in figure 5.4.

PintUserAgent
state user gw

PintUserAgent()
init()
proceedInput()
gwRequestInvite()
gwAck()
gwRequestBye()
gwReplyBye()
feedFsm()

state

The *static int state* represents the current state of the FSM (see figure 5.4).

user

The *PintSocketUserAgent user* is used to access the methods in the “CU” class, which is a TCP/IP server socket, getting connected by the corresponding “CU” of the User process.

gw

The *PintSocketUserAgent gw* is used to access the methods in the other “CU” class, which connects as client to the corresponding “CU” of the PINT Gateway process by TCP/IP socket.

PintUserAgent()

PintUserAgent() is the constructor of this class and is empty at the moment.

init()

The *void init(PintSocketUserAgent user, PintSocketUserAgent gw)* method is used in the initialization phase. It is invoked by the *main()* method of *PintUserAgentStart* class after the *main()* knows the reference to the two “CU”. This method initializes the “CU” references in this thread and creates the *Vector cache.*

proceedInput()

The *void proceedInput(int sender, String str) throws Exception* method is invoked by the *run()* method after a new message arrived. Depending on the sender of the message it calls the corresponding parser. If the message is sent by PINT protocol, it invokes the *fillInCommand()* method for further preprocessing of the message. Then it invokes the *feedFsm()* method to perform the FSM.

feedFsm()

The *protected void feedFsm(int sender, Hashtable recv) throws Exception* method performs the assigned FSM. Depending on the content of “Command” in the *Hashtable recv* and the sender, it generates the corresponding zero, one or two output messages and changes to the new state. The sending of the output messages goes through the corresponding “CU”. The FSM are described in section 5.2.2.

If the output message is a PINT message, one of the below methods is used:

- *protected void gwRequestInvite() throws Exception*
- *protected void gwAck() throws Exception*
- *protected void gwRequestBye() throws Exception*
- *protected void gwReplyBye() throws Exception*

The “gw” at the beginning of the method name indicates, that the message is sent to the Pint Gateway process. Some of these methods need the A or B party, which are looked up in *sessMem*. To mark the boundary between the messages, in this prototype application, at the end of each PINT message there is a line inserted, containing a string “EOF” and CRLF.

5.4.6 PintGateway

The *public class PintGateway extends PintMutual* is the “M” of the Pint Gateway process. It is a thread, which is initialized and started by the *main()* method of the *PintGatewayStart* class. It defines integers for the different states. Those correlate with the values used in figure 5.5

PintGateway
state
ua
ine
PintGateway()
init()
proceedInput()
uaReplyInvite()
uaRequestBye()
uaReplyBye()
feedFsm()

state

The *static int state* represents the current state of the FSM (see figure 5.5).

ua

The *PintSocketGateway ua* is used to access the methods in the “CU” class, which is a TCP/IP server socket, getting connected by the corresponding “CU” of the User Agent process.

ine

The *PintSocketGateway ine* is used to access the methods in the other “CU” class, which connects as client to the corresponding “CU” of the IN Emulation process by TCP/IP socket.

PintGateway()

PintGateway() is the constructor of this class and is empty at the moment.

init()

The *void init(PintSocketGateway ua, PintSocketGateway ine)* method is used in the initialization phase. It is invoked by the *main()* method of the *PintGatewayStart* class after the *main()* knows the reference to the two “CU”. This method initializes the “CU” references in this thread and creates the *Vector cache.*

proceedInput()

The *void proceedInput(int sender, String str) throws Exception* method is invoked by the *run()* method after a new message arrived. If the message is sent by PINT protocol, it invokes the PINT parser and after this the *fillInCommand()* method for further preprocessing of the message. If the messages comes from the IN Emulation, a simple string comparison is used as parser. Afterwards it invokes the *feedFsm()* method to perform the FSM.

feedFsm()

The *protected void feedFsm(int sender, Hashtable recv) throws Exception* method performs the assigned FSM. Depending on the content of "Command" in the *Hashtable recv* and the sender, it generates the corresponding zero, one or two output messages and changes to the new state. The sending of the output messages goes through the corresponding "CU". The FSM are described in section 5.2.3.

If the output message is a PINT message, one of the below methods is used:

- *protected void uaReplyInvite() throws Exception*
- *protected void uaRequestBye() throws Exception*
- *protected void uaReplyBye() throws Exception*

The "ua" at the beginning of the method name indicates, that the message is sent to the User Agent process. Some of these methods need the A or B party, which are looked up in *sessMem*. To mark the boundary between the messages, in this prototype application, at the end of each PINT message there is a line inserted, containing a string "EOF" and CRLF.

5.4.7 PintInEmulation

The *public class PintInEmulation extends PintMutual* is the "M" of the IN Emulation process. It is a thread, which is initialized and started by the *main()* method of the *PintInEmulationStart* class. It defines integers for the different states. Those correlate with the values used in figure 5.6.

PintInEmulation
state stdOI pgw

init()
PintInEmulation()
proceedInput()
feedFsm()

state

The *static int state* represents the current state of the FSM (see figure 5.6).

stdOI

The *PintStdOI stdOI* is used to access the methods in the "CU" class, which writes the output to the display and takes input from the keyboard.

pgw

The *PintSocketInEmulation pgw* is used to access the methods in the other "CU" class, which is a TCP/IP server socket, getting connected by the corresponding "CU" of the Pint Gateway process.

init()

The *void init(PintStdOI stdOI, PintInEmulation)* method is used in the initialization phase. It is invoked by the *main()* method of the *PintInEmulationStart* class after the *main()* knows the reference to the two "CU". This method initializes the "CU" references in this thread and creates the *Vector cache.*

PintInEmulation()

PintINEmulation() is the constructor of this class and is empty at the moment.

proceedInput()

The *void proceedInput(int sender, String str) throws Exception* method is invoked by the *run()* method after a new message arrived. If the message comes from the Pint Gateway, the corresponding parser is invoked and and if needed, the *sessMem* us updated. If the message comes from the keyboard, a simple string comparison is used as parser. Afterwards it invokes the *feedFsm()* method to perform the FSM.

feedFsm()

The *protected void feedFsm(int sender, Hashtable recv) throws Exception* method performs the assigned FSM. Depending on the content of “Command” in the *Hashtable recv* and the sender, it generates the corresponding zero, one or two output messages and changes to the new state. The sending of the output messages goes through the corresponding “CU”. The FSM are described in section 5.2.4.

5.4.8 PintStdIO

The *public class PintStdIO extends Thread implements PintCommon* is one of the “CU” of the User process. It is a thread, which is initialized and started by the *main()* method of the *PintUserStart* class.

PintStdIO
in out pu

PintStdIO
send()
receive()
run()

in

The *BufferedReader in* is used to read from the keyboard.

out

The *PrintWriter out* is used to write to the display.

pine

The *PintInEmulation pu* is used to access the methods in the “M” class.

PintStdIO()

The *public PintStdIO(PintUser pu) throws Exception* method is the constructor of this class. It initializes *in*, *out* and *pu*.

send()

The *public void send(String str) throws Exception* method is invoked by the "M" thread to print a message to the display.

receive()

The *public String receive() throws Exception* method reads full lines from the keyboard and returns them to the calling function (in this case *run()*).

run()

The *public void run()* method is waiting for messages of the *receive()* method and fills the *Msg msg* with the sender information (here keyboard) and the message itself and calls the *put* method in the "M" thread.

5.4.9 PintStdOI

The *public class PintStdOI extends Thread implements PintCommon* is one of the "CU" of the IN Emulation process. It is a thread, which is initialized and started by the *main()* method of the *PintInEmulationStart* class.

PintStdOI
in
out
pine
PintStdOI
send()
receive()
run()

in and out

See section 5.4.8.

pine

The *PintInEmulation pine* is used to access the methods in the "M" class.

PintStdOI()

The *public PintStdOI(PintInEmulation pine) throws Exception* method is the constructor of this class. It initializes *in*, *out* and *pine*.

send(), receive() and run()

See section 5.4.8.

5.4.10 PintSocket

The *abstract class PintSocket extends Thread implements PintCommon* is the base object for all the "CU" threads, which are connected through TCP/IP sockets.

PintSocket
s in out kindOfSocket hostname port otherEnd

init()
send()
receive()
abstract put()
run()

s

The *Socket s* is the socket, which is configured by the *init()* method.

in

The *BufferedReader in* is used to read from the socket.

out

The *PrintWriter out* is used to write into the socket.

kindOfSocket

The *int kindOfSocket* is a parameter, which is initialized by the constructor of the class. (The constructor is defined in the extending classes.) It contains an integer representation for either SERVER or CLIENT, defined in *PintCommon* (sec. 5.4.1), first paragraph.

hostname

The *String hostname* is a parameter, which is initialized in the constructor of the class. It contains the hostname of the other end to which the socket gets connected to.

port

The *int port* is a parameter, which is initialized in the constructor of the class. It contains the port number of the other end to which the socket gets connected to.

otherEnd

The *int otherEnd* is a parameter, which is initialized by the constructor of the class. It contains an integer representation for the process name it gets connected to. The integer representation is defined in *PintCommon* (sec. 5.4.1), last paragraph.

init()

The *void init(Socket s)* method is used to initialize a socket. It is invoked by the *run()* method in this class.

send()

The *public void send(String str) throws Exception* method is used to write into the socket.

receive()

The *public void receive() throws Exception* method is used to receive from the socket. If the *port* number is 5060 (default SIP port) it expects a PINT message, which has more than one line and its end is marked by the string "EOF" and CRLF (in this prototype application). After getting this "EOF", it returns the whole message, without the "EOF" line. In the other cases (port != 5060), it reads one line and returns it.

abstract put()

The *abstract void put(Msg msg)* method is described further in the classes, which implement it. It has to be in this class as predefinition, because the compiler wouldn't accept its absence.

run()

The *public void run()* method is the entry point of a thread, invoked by the *start* method.

- If CLIENT socket is specified in *kindOfSocket*, this method creates a stream socket and connects it to the specified *port* at the specified *hostname*
- If a SERVER socket specified, the method creates a server socket on the specified port. Then it listens for a connection to be made to this socket by the corresponding CLIENT and accepts it.

After this it invokes the *init()* method and in an endless loop, waits for messages by calling the *receive()* method. After receiving a message, it adds the information about the sender and forwards both in one (as *Msg* class) invoking *put.*

5.4.11 PintSocketXxx

In *class public class PintSocketXxx extends PintSocket* the abbreviation Xxx is used. It stands for User, UserAgent, Gateway and InEmulation. All theses classes are basically the same. pxx is an abbreviation for pu, pua, pgw and pine.

PintSocketXxx
pxx

PintSocketXxx()
put()

Those contain a reference *pxx* to the "M" class and a constructor *public PintSocketXxx(PintXxx pxx, int kindOfSocket, int otherEnd, String hostname, int port) throws Exception*, which is taking its arguments and makes them known to the whole class.

Furthermore it contains a *void put(Msg msg)* method, which gets invoked by the *run()* method and does nothing else, than forward the message to the corresponding "M" class.

5.4.12 PintXxxStart

In *public class PintXxxStart implements PintCommon*, the same abbreviation as in section 5.4.11 is used. All theses (Start) classes are basically the same.

PintXxxStart

PintXxxStart()
main()

Those contain a constructor *public PintXxxStart()*, which is empty at the moment.

Furthermore there is a *public static void main(String args[])* method, which is the entry point of the class. If there are arguments (hostnames to connect), it overrides the default hostnames defined in *PintCommon* (sec. 5.4.1). Then it creates the "M" thread, and with its reference to the "M" thread, it creates the two "CU" threads. To inform to "M" thread, how the "CU" threads can be accessed, it invokes the *init()* method of the "M" thread. At the end it starts all three (one "M" and two "CU") threads.

5.5 Lexer and Parser

The [38] the tools lex & yacc are described. Those are designed for the production of lexical analyzers and parsers in C Code. Since I have implemented my application in Java, I used their Java equivalents:

- JFlex [39] (instead of Lex)
- CUP [40] (instead of yacc)

The following introduction is from [38], but it applies also to JFlex and CUP (not only for lex and yacc).

5.5.1 Introduction

Lex and yacc are tools designed for writers of compilers and interpreters. But they are also useful for many applications, that will interest the noncompiler writer. Any application that looks for patterns in its input, or has an input or command language is a good candidate for lex and yacc. Furthermore, they allow rapid application prototyping, easy modifications, and simple maintenance of programs. In general, lex and yacc help to write programs that transform structured input. In such programs two tasks that occur over and over are dividing the input into meaningful units, and then discovering the relationship among the units. For a text searching program, the units would probably be lines of text, with a distinction between lines that contain a match of the target string and lines that don't. (For the reader that is familiar with the Unix command *grep*: This is what grep does.) For a C program, the units are variable names, constants, strings, operators, punctuation, etc. This division into units (which are usually called *tokens*) is known as *lexical analysis*, or *lexing* for short. Lex helps by taking a set of descriptions of possible tokens and producing a C routine (in case of JFlex a Java class), called *lexical analyzer* (or *Lexer* for short) which identify those tokens. The set of descriptions given to lex are called a *lex specification.*

The token description that lex uses are known as *regular expressions.* Lex turns this regular expressions into a form that the Lexer can use to scan the input text extremely fast, independent of the number of expressions that it is trying to match. A lex Lexer is almost always faster than a Lexer that you might write in C (or Java) by hand.

As the input is divided into tokens, a program often needs to establish the relationship among the tokens. A C compiler needs to find the expressions, statements, declarations, blocks, and procedures in the program. This task is known as *parsing* and the list of rules that define the relationships that the program understand is a *grammar.* Yacc takes a concise description of a grammar and produces a C routine (in case of CUP a Java class), that can parse that grammar, a *Parser.* The yacc Parser automatically detects whenever a sequence of input tokens matches one of the rules in the grammar and also detects a syntax error whenever its input doesn't match any of the rules.

A yacc Parser is generally not as fast as a Parser you could write by hand, but the ease in writing and modifying the Parser is invariably worth any speed loss. The amount of time a program spends in a Parser is rarely enough to be an issue anyway.

More information about lex & yacc, which covers also a lot of general issues on Lexers and Parser can be found in [38]. The Java equivalent JFlex is documented on [39] and CUP on [40].

5.5.2 Parser used in Application

In the application described at the beginning of this chapter, the CUP Parser (together with its underlaying JFlex Lexer) is used for parsing the PINT protocol. The Parser checks, whether the protocol is correct following the specified grammar and extracts the information from the PINT protocol (see section 4.6), in order to put it into a Java object Hashtable, that can be easily maintained by the any Java code. I wrote also other small Parsers, to parse the (one line) messages, which are exchanged between the processes.

When parsing a SIP message, the Lexer divides the incoming string into tokens. In principle I used two kind of tokens:

- The names of the header fields
- Zero or more tokens generated out of the content of each header field

When parsing a SDP message, the tokens are:

- The types of the SDP field
- Zero or more tokens generated out of the corresponding content

The Parser takes these tokens and tries to match the rules of the grammar. When a rule is matched, an action might be performed. Usually the action consists of adding some values to the Hashtable, which the Parser gets as argument. Tokens can be defined as Java objects String, Integer, etc. I used only the String object. Some actions can generate new tokens out of the tokens, that just have been matched. These new token can be part of a new rule (hierarchical specification). It is even possible, to assign a value to this new token. I generate sometimes a new String out of the tokens, that just have been matched. I assign this String to the new token.

5.5.3 Java Classes in Parser

CUP generates three Java classes. In the example of SIP, those are:

- SipParser
- CUP$SipParser$actions
- sym

JFlex produces one output class. In the example of SIP: SipLexer

Figure 5.9 shows the interaction of theses classes:

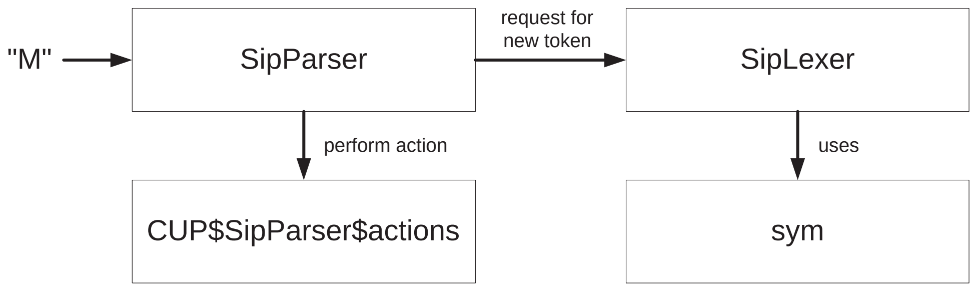


Figure 5.9: Interaction of Parser related Java classes

After receiving a SIP (PINT) message, the "M" invokes the *SipParser*. The Parser requests tokens from its Lexer *SipLexer*. After the Parser can match a rule, an action is performed, calling *CUP$SipParser$actions*. The class *sym* is used by the Lexer, to ensure using the same conventions for the symbols as the Parser does.

I varied from the normal use of JFlex and CUP. This is described in the following sections.

5.5.4 Tracing with JFlex

In order to ease the debugging of the Lexer and Parser I do not return the tokens directly. I catch them in order to print the name of the token to the standard output (display). If a lexical state changes, I print also a message to the standard output.

Since the Lexer (Java class) operates with integers (and not with the meaningful names given to the tokens in the JFlex specification), the meaningful names have to be decoded from the integers. For reaching this, I wrote a PERL script, which reads the files “sym.java” and “SipLexer.java” (the output files of JFlex), where the assignments can be extracted from. The output of the PERL script are two following simple text based files:

- “states.txt” contains the integer and the corresponding meaningful name of the lexical state in format:

  integer CRLF state CRLF.

- “sym.txt” contains the assignment of the tokens in the same format:

  integer CRLF tokenname CRLF.

Unfortunately the PERL script currently only works with Unix, since the Unix Command “grep” is used in the script.

A java class “LookUp.java” is written for reading these two files and saving the relevant values in a Hashtable. In “sip.flex” file the following code is included in order to get this tracing information:

```
LookUp luSym = new LookUp("sym.txt") ;
LookUp luSt  = new LookUp("states.txt") ;

private Symbol symbol(int type) {
   System.out.println("-> Type: " + luSym.lookUp(type)) ;
   return new Symbol(type) ;
}

private Symbol symbol(int type, Object value) {
   System.out.println("-> Type: " +  luSym.lookUp(type) + "; Value: " + value) ;
   return new Symbol(type, value) ;
}

private void chState(int newState) {
   yybegin(newState) ;
   System.out.println("## New State: " + luSt.lookUp(newState)) ;
}
%}
```

Whenever there is a change of the lexical state, the *chState(newState)* method is called, instead of changing the state directly with *yybegin(newState)*. This allows to execute some Java code when the state is changed. In this example print the new state to standard output.

Instead of returning a token directly using *return new Symbol(type, value)*, the same command with a small "s" in symbol is used: *return new* s*ymbol(type, value)*. This allows to execute some Java code when a symbol is returned. In this case print the returned symbol to standard output.

I configured the CUP Parser so that whenever it cannot match the tokens by any rule, it prints "Syntax error" to the standard error. This is using error recovery as described in the CUP User's Manual, which can be found on [40].

These three outputs together allow an exact tracing of the Parser and Lexer.

Remark: The change of the state (and the related printing to the standard output) is done before the current token is returned. This might be confusing, when the above described tracing is used.

This feature is not limited to this application. It can also be used for other Lexer/Parser specifications, which are designed with JFlex/CUP.

5.5.5 Actions in CUP

Furthermore I have added some personal code to CUP, in order to have the choice to either add a value to an existing Hashtable key or to create a new Hashtable key. In the later case it reports an error, if the key is already existing and the existing value is overridden by the new one. The added lines are the following:

```
action code {:

private void newEntry(String key, String entry) {
   String oldEntry = null;
   oldEntry = (String)parser.ht.put(key, entry);
   if (oldEntry != null) {
      System.err.println("Warning: Entry for \'" + key + "\' exists already.");
      System.err.println("Old entry \'" + oldEntry + "\' ignored.");
   }
}

private void addEntry(String key, String entry) {
   String oldEntry = null;
   oldEntry = (String)parser.ht.put(key, entry);
   if (oldEntry != null) {
      parser.ht.put(key, oldEntry + "\n" + entry);
   }
}

:};
```

Chapter 6

Results

This chapter is divided in three parts. Section 6.1 contains the direct results, which can be found in this report. Section 6.2 contains the input, I could contribute for the standardization process of the PINT protocol. Section 6.3 summarizes the educational benefit, which I personally can take out of this work.

6.1 Direct Results

One result the introductory part of this report (chapters 1–4) allows a newcomer, to get an overview about SIP, SDP and PINT within a reasonable amount of time. This I reached focusing on the essential parts, skipping the details and deeper descriptions. They also contain some figures, which help a lot for the understanding of the topic. As a newcomer, the RFCs and Internet Drafts are not always easy to read, because they usually cover a whole subject with all its special cases and exceptions, which might be confusing the reader.

In section 4.1 I describe some services, which can be implemented using PINT, TNIP or both of them together. Some of these services exist already, using a non-PINT protocol (e.g. checking voicemail through web) or mentioned in the referenced documents (e.g. Internet Call Waiting Service), but there are also some new ideas, which came into my mind and are worth considering more closely.

On practical side, I can present a working prototype application, which uses the PINT protocol. It can be used as a base for further development in PINT. Especially the Lexer and the Parser, written in the Java tools "JFlex" and "Java CUP" can be used as a basic building block of any PINT application. The design of the Parser is made so, that it also parses normal SIP and SDP packets. Thus it can also be used for the development of SIP applications. The application is described in chapter 5. In section 5.5 it contains a description about the Parser and the specialties I've added to it.

As the outcome of phase 3 following the conceptional formulation of this thesis, I found two sources for software, which are worth on considering more closely:

- There is a company called *Dynamicsoft* (`http://www.dynamicsoft.com`), which advertises *jsip*, Java tools for SIP. I contacted this company, in order to get some tools for the implementation part. But the reaction time of their sales department was rather slow, so that I didn't use them in my implementation.
- The Columbia University (`http://www.cs.columbia.edu/~hgs`) is also developing SIP software. A research group on Helsinki University of Technology (HUT), Laboratory of Telecommunications Technology is using a SIP server, which was developed at the Columbia University.

A list of the companies and institutions, which are working with SIP, can be found on URL: `http://www.cs.columbia.edu/~hgs/sip/implementations.html`. There are also two public SIP server available, which can be used for testing purpose. More about this can be found on URL: `http://www.cs.columbia.edu/~hgs/sip/servers.html`

6.2 Input for IETF PINT

This section lists the most important inputs, I provided the developers of the PINT protocol [1].

The most important question I put to the PINT mailing list [32] concerns an ambiguous BYE request, if SUBSCRIBE is used with the same Call-ID as the previous INVITE was sent with. In [1] it is not stated, to what a BYE request refers to in this case. It might either terminate the monitoring session started with SUBSCRIBE or terminate the call and let the monitoring session running or shut down both at the same time. After my posting, loads of followups were going through the mailing list. A decision has not yet been reached, but it is likely, that it will be published in the next version of [1].

In [1, Appendix A], which lists the ABNF [33] rules of the PINT protocol, there is a rule, which allows as transport protocol (sec. 4.5.8) a keyword "phone", instead of "voice" which is used all over the document. I reported this mismatch to the authors of [1].

In all the examples of [1], there was the CSeq header field missing, which is mandatory in SIP (section 2.6.1, page 20).

Another conflict, which I reported to the mailing-list, concern the SDP "s:" (subject) and "t:" (time) fields. As described in section 3.2, these fields are mandatory in SDP. Also the person responsible for a session—either "e=" (email) or "p=" (phone) field in SDP—has to be part of a session description, as described on page 30. The examples of [1] do not contain these fields. As long as the SDP parser is tolerant, this doesn't matter. But a strict parser might reject a packet, were these fields are missing. A final decision concerning this hasn't been made until now. A proposal to solve this problem is expected to be presented in the next version of [1].

6.3 Educational Benefit

The whole diploma thesis extended my personal experience and knowledge remarkably. I learned a lot about protocols used in Internet. The domains of SIP, SDP and PINT were totally new for me at the time I started the thesis.

One interesting part was, to follow and to take part actively in a standardization process. I could contribute a couple of improvements to the Internet Draft [1] and maybe prevent developers from problems, resulting from mistakes in the emerging standard.

In the software area, I got in touch the first time with an Object Orientated programming language—Java. Also the use of Lexers and Parsers was something, I learned during this time.

Since this thesis has been carried out completely in English, I improved also my language skills. Besides this I had the opportunity to extend my reporting skills with the tool LaTeX.

Chapter 7

Performance of the thesis

7.1 General Problems

Studying PINT required a clear picture of the protocols it is built on—namely SIP and SDP. The SIP taken by itself is quite extensive, so that it would easily fill a thesis alone.

Compared to the original plans, I had to reduce the functionality of the software due lack of time. It was just impossible to learn so many new tools (Java, Lexer and Parser) and efficiently use them within the time scope foreseen for the thesis. Also the implementation of the SIP Parser took much more time then expected. Mostly this was caused, since the ABNF grammar for SIP was spread around the whole document (RFC 2543 [13]). After I finally succeeded to make the Parser running, a complete html-linked SIP grammar was published on URL: `http://www.cs.columbia.edu/~hgs/sip/SIPgrammar.html`. In this html-version of the SIP grammar, there were also a couple of deficiencies corrected, which had been found in [13]. The standardization process of the PINT protocol is still in its early state (Internet Draft). The concept as well as the relating Internet Drafts are not yet far developed and contain mistakes. This had the consequence, that the understanding was sometimes quite difficult. Furthermore the Interface to IN—the Executive System of PINT—currently exists only in layouts.

7.2 Compared to the Conceptional Formulation

Compared to the Conceptional Formulation, included at the beginning of this report, I progressed as follows:

1. **Comparison IN/Internet:** I have been reading through several related literature and documented the results in the introductory chapters of this report.

2. **Interworking scenarios:** The most typical interworking scenarios are described in chapter 4 of this report.

3. **System level specification:** The outcome of this chapter is documented in section 6.1 (available software) and in section 5.2 (design of the application).

4. **Building of development/test environment:** I decided to use Java for the development and write the client and server application on my own. There was no hardware installation required, since I could use the existing environment in NRC. The applications were developed mainly on Unix platforms.

5. **Proof of concept:** The application is described in chapter 5.

6. **Reporting:** This report covers all the topics, studied during this thesis

Overall seen, I could fulfill the requirements of the thesis pretty well. Only the functionality of the software application is limited from that, what I originally planed to implement. But since software developments always take more time, than expected, I don't consider this a real drawback in my thesis.

Chapter 8

Outlook

The application described in chapter 5 can be extended to more functionality and handling of several calls at the same time. Also a UDP version of the PINT message exchange should be considered.

The quality of the Lexer can also be improved, using more the lexical states.

The parser can be modified, so that it uses the same naming conventions as in the just published SIP grammar (`http://www.cs.columbia.edu/~hgs/sip/SIPgrammar.html`). It will be easier to maintain it, if newer versions of SIP (or PINT) are released. It should also be considered, to separate the SIP and the SDP Parser, so the no naming conflicts occur (some tokens have the same name in SIP and SDP grammar). Furthermore it will be easier to handle any payload of SIP; this applies also for encrypted payload. In the current version the production rules are kept easy. In these cases where more than one session description occurs in the same SDP, the output in the Hashtable is unusable, since the second session description overrides the first one. The same problem occurs, if more than one media type is defined. This problem could be solved, e.g. by using other Java objects for the output of the parser.

After researching the PINT area, I came to the conclusion, that it is worth on taking effort in developing PINT applications. A supplier, which can't provide this feature has a remarkable drawback compared to its competitors that offer this. The benefit for the users is quite high, since it makes the use of the web combined with telephone services much easier. Thus the teleoperators are almost forced to include this feature in their future service packets.

It should be considered to implement some of the services described in section 4.1.

It is expected that new kinds of PINT services will emerge in the nearer future. For example new PINT building blocks with applications to Conference Calling. Such a proposal is described in [41].

Chapter 9

Final Words

The work for this diploma thesis was carried out at Nokia Research Center Helsinki in the Communication Systems and Networks laboratory, supervised by Professor Raimo Kantola, Helsinki University of Technology (HUT) and Professor Albert Kündig, Swiss Federal Institute of Technology Zurich (ETH).

I would like to express my gratitude to my instructor Hannu Flinck, for his guidance during my work for the thesis. I would like to thank also the other workers at Nokia Research Center for their competent assistance.

Furthermore I would like to thank Professor Raimo Kantola, especially for providing me with a contact to Nokia and assisting me in administrative matters.

Last but not least I would like to thank Professor Albert Kündig and Dr. Urs Röthlisberger for making it possible to do my thesis abroad in a company as well as for the provided "remote assistance".

Helsinki, 10th of September 1999 Bernhard Höneisen

Appendix A

Abbreviations

ABNF	Augmented Backus-Naur Form
API	Application Program Interface
ASCII	American Standard Code for Information Interchange
CAMEL	Customized Applications for Mobile network Enhances Logic
CR	US-ASCII CR, carriage return character (%d13)
CRLF	Carriage Return AND/OR Linefeed (%d13, %d10 or %d13 %d10)
CS	Capability Set (IN)
CSeq	Command Sequence
DTMF	Dual Tone Multi Frequency
FIFO	First In First Out
FSC	Fixed Switching Center
GSM	Global System for Mobile communication
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
HUT	Helsinki University of Technology
IANA	Internet Assigned Numbers Authority
ICANN	Internet Corporation for Assigned Names and Numbers (former IANA)
IETF	Internet Engineering Task Force
IN	Intelligent Networks
IP	Internet Protocol
ISDN	Integrated Services Digital Network
ISP	Internet Service Provider
ITU-T	International Telecommunications Union - Telecommunication Standardization Sector
JDK	Java Development Kit
JPEG	Joint Photographic Experts Group
LAN	Local Area Network
LF	US-ASCII LF, line feed character (%d10)
MIB	Management Information Base
MIME	Multi-Purpose Internet Mail Extensions

MMUSIC	Multiparty Multimedia Session Control (IETF)
MSC	Mobile Switching Center
MTU	Maximum Transfer Unit
NRC	Nokia Research Center
PBX	Private Branch Exchange
PINT	PSTN/Internet Interfaces (IETF)
PSTN	Public Switched Telephone Network
QoS	Quality of Service
R2C	Request to Call (Click-to-Dial)
R2F	Request to Fax (Click-to-Fax)
R2HC	Request to Hear Content (Click-to-Hear-Content)
RAS	Registration, Admission and Status (H.323)
RFC	Request For Comments
SAP	Session Announcement Protocol
SCP	Service Control Point (IN)
SDP	Session Description Protocol
SIP	Session Initiation Protocol
SMIL	Synchronized Multimedia Integration Language
SMP	Service Management Point (IN)
SMS	Short Message Service (GSM)
SMS	Service Management System (IN)
SN	Service Node (IN)
SNMP	Simple Network Management Protocol
SP	US-ASCII SP, space character (%d32)
SSP	Service Switching Point (IN)
SSTP	Service Support Transfer Protocol
SoI	Signaling support of services over IP-based networks (ITU-T)
TCP	Transmission Control Protocol
TNIP	PINT service, initiated in IN and executed in IP (reverse spelling to PINT)
TSP	Telephone Service Provider
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
WAP	Wireless Application Protocol

Appendix B

Installation and Code Generation

Here the files on the appended floppy disk are described. The disk contains all relevant files of the application described in chapter 5. Also some hints, how to generate code with CUP and JFlex can be found in this chapter.

B.1 Software versions

The following versions were used:

- JDK 1.2.1 (Java)
- JFlex 1.2.1
- CUP 0.10i

If you want to run the prototype application, make sure that you have the same (or a higher) Java version installed on your system.

B.2 Installation

I recommended to copy to directory "pint" (including subdirectories) to a local directory. The CLASSPATH must be set to this directory (that one which contains "pint"). I might be necessary to reinstall JFlex and/or CUP. In this case make sure, that you use the same versions (see above).

B.3 Files on the disk

On the disk there are the following files:

/:

This is the root directory.

```
JFlex/*       # Files, which were installed for JFlex
java_cup/*    # Files, which were installed for Cup
pint/         # Files, which were implemented by me, see below
```

/pint/:

This directory contains all files, which I implemented. They are described in chapter 5. The Parser and Lexer files are the in subdirectories “SIPparser” and “parser”.

```
SIPparser/
parser/
Msg.class
Msg.java
PintCommon.class
PintCommon.java
PintGateway.class
PintGateway.java
PintGatewayStart.class
PintGatewayStart.java
PintInEmulation.class
PintInEmulation.java
PintInEmulationStart.class
PintInEmulationStart.java
PintMutual.class
PintMutual.java
PintSocket.class
PintSocket.java
PintSocketGateway.class
PintSocketGateway.java
PintSocketInEmulation.class
PintSocketInEmulation.java
PintSocketUser.class
PintSocketUser.java
```

```
PintSocketUserAgent.class
PintSocketUserAgent.java
PintStdIO.class
PintStdIO.java
PintStdOI.class
PintStdOI.java
PintUser.class
PintUser.java
PintUserAgent.class
PintUserAgent.java
PintUserAgentStart.class
PintUserAgentStart.java
PintUserStart.class
PintUserStart.java
```

/pint/SIPparser/:

This subdirectory contains the files, belonging to the SIP Parser:

```
CUP$SipParser$actions.class # generated after compiling ‘SipParser.java’
LookUp.class
LookUp.java      # can be used for tracing the SIP Parser/Lexer
SipLexer.class
SipLexer.java    # output of JFlex with input sip.flex
SipParser.class
SipParser.java   # output of CUP with input sip.cup
StartIt.class
StartIt.java     # can be used for tracing the SIP Parser/Lexer
makeit           # can be used for generating all in once: CUP, JFlex and com-
                 # piling of StartIt, which compiles also the Parser and Lexer
sip.cup          # contains the SIP grammar and production rules
sip.flex         # contains the lexical specification
startit          # can be used to start a tracing for the SIP Parser/Lexer
states.txt       # output of tablize.perl
sym.class
sym.java         # output of CUP with input sip.cup
sym.txt          # output of tablize.perl
tablize.perl     # takes ‘sym.java’ and ‘SipLexer.java’ and generates
                 # the output files ‘sym.txt’ and ‘states.txt’
                 # only needed for tracing the SIP Parser/Lexer
```

/pint/parser/:

This subdirectory contains the files, belonging to the other parsers used it the application. The explanations are basically the same as in this SIPparser subdirectory (just above).

```
CUP$PgwIneParser$actions.class
CUP$UaUserParser$actions.class
CUP$UserUaParser$actions.class
PgwIneLexer.class
PgwIneLexer.flex
PgwIneLexer.java
PgwIneParser.class
PgwIneParser.cup
PgwIneParser.java
PgwIneStartIt.class
PgwIneStartIt.java
StartIt.class
UaUserLexer.class
UaUserLexer.flex
UaUserLexer.java
UaUserParser.class
UaUserParser.cup
UaUserParser.java
UaUserStartIt.class
UaUserStartIt.java
UserUaLexer.class
UserUaLexer.flex
UserUaLexer.java
UserUaParser.class
UserUaParser.cup
UserUaParser.java
UserUaStartIt.class
UserUaStartIt.java
makeitPgwIne
makeitUaUser
makeitUserUa
startitPgwIne
startitUaUser
startitUserUa
sym.class
sym.java
```

B.4 Generation of Parser/Lexer

The following commands have to be used in order, when using CUP and JFlex together:

```
java java_cup.Main -package pint.parser -parser UserUaParser < UserUaParser.cup
jflex UserUaLexer.flex
```

This example uses a grammar specified in *UserUaParser.cup* and as output it generates the files *UserUaParser.java* and *sym.java.* Then it takes lexical specification in the file*UserUaLexer.flex* and generates the output file *UserUaLexer.java*

After this, all the generated Java files have to be compiled.

During the implementation of the Parser, I faced some problems with JFlex and CUP. In the latest versions those are fixed due my feedback. The whole implementation is made with the older versions, were these problems still last.

B.4.1 Out of memory in JFlex

Since the SIP grammar is quite complex, the Lexer required more memory than allocated normally by the Java Virtual Machine, which resulted to an *Out of memory* Exception. This problem could be solved assigning a higher *-Xmx* value to the Java Virtual Machine when starting JFlex, as done in */JFlex/bin/jflex*:

```
java -Xmx256m JFlex.Main $@
```

B.4.2 reserved words in CUP

Since "parser" used to be a reserved word in CUP, it was not allowed to use it in the package name. But it was possible to specify the package in the parameter *-package* when starting CUP as in the following example:

```
java java_cup.Main -package pint.parser -parser UserUaParser < UserUaParser.cup
```

For further details on JFlex and CUP I refer to [39] [40].

B.5 Tracing the SIP Parser

As described in section 5.5.4, I inserted a mechanism into the Lexical Specification in order to trace the SIP parser.

To use this feature, uncomment the code (listed in section 5.5.4) in *sip.flex.* After generating the Parser and Lexer files (see B.4), run the PERL script *tablize.perl*[1] and to start the tracing, use:

```
java pint.SIPparser.StartIt <name_of_the_file_containing_a_SIP_SDP_protocol>
```

[1]Only working in Unix!
It might be necessary to modify the $filepath variable in this PERL script. The $filepath variable must contain the same directory, where the SIP Parser is located.

Appendix C

Running the test Application

After the installation (see Appendix B), the prototype application can be started. The four processes might be running on the same or on different hosts. Make sure, that you start the processes in the following order:

- java PintInEmulationStart <host_of_PintGateway>
- java PintGatewayStart <host_of_PintUserAgent> <host_of_PintInEmulation>
- java PintUserAgentStart <host_of_PintUser> <host_of_PintGateway>
- java PintUserStart <host_of_PintUserAgent>

Here a short description how to use the prototype application:

After starting all four processes, the first three ones should indicate, that the socket was accepted. Then you can write into the window of the PintUser process:

call <phone number A> <phone number B>

In the IN Emulation window there should now appear:

Allow connection of number <phone number A> with number <phone number B> ? (yes/no)

If you write *yes* into the IN Emulation window, the connection will be established. This is indicated by:

Phone Call established...

At the same time the question:

Terminate call? (yes/no)

is asked. After writing *yes* into the IN Emulation window the phone call is terminated.

Any time it is possible to terminate the call from the User process window writing:

disconnect

Besides this some tracing messages are printed to the windows. Those are useful, if you want to follow the FSMs in chapter 5.2. There you can also find more details about the application.

Bibliography

[1] http://www.ietf.org/internet-drafts/draft-ietf-pint-protocol-01.txt
The PINT Service Protocol: Extensions to SIP and SDP for IP Access to Telephone Call Services .

[2] http://www.etsi.org/SMG/SMG1/CAMEL.htm
Costumized Applications for Mobile network Enhances Logic (CAMEL).

[3] ftp://ftp.isi.edu/in-notes/rfc2326.txt
RTSP: Real Time Streaming Protocol (RFC 2326).

[4] C. Gbaguidi, J.-P. Hubaux, C. Pacifici, and A.N. Tantawi. An architecture for the integration of internet and telecommunication services. *IEEE Second Conference on Open Architectures and Network Programming Proceedings 1999*, pages 9–21, 26.-27. March 1999.

[5] M. Paavonen. Extending intelligent networks to the Internet. Master's thesis, Helsinki University of Technology (HUT), faculty of Electrical and Communications Engineering, 1999.

[6] Andrew S. Tanenbaum. *Computer networks.* Prentice Hall, 1996.

[7] Thomas Magedanz and Radu Popescu-Zeletin. *Intelligent Networks.* Coriolis Group (Sd), 1996. ISBN: 1850322937.

[8] http://www.forum.nokia.com/developers/wap/wap.html
WAP: Wireless Application Protocol.

[9] http://www.nokia.com/phones/9110/index.html
Phones: Nokia 9110 Communicator.

[10] http://www.ietf.org/
IETF: The Internet Engineering Task Force .

[11] http://www.ietf.org/html.charters/mmusic-charter.html
Multiparty Multimedia Session Control (mmusic).

[12] `http://www.cs.columbia.edu/~hgs/sip/`
SIP: Session Initiation Protocol .

[13] `ftp://ftp.isi.edu/in-notes/rfc2543.txt`
SIP: Session Initiation Protocol (RFC 2543).

[14] `http://www.ietf.org/internet-drafts/draft-ietf-mmusic-sap-v2-02.txt`
SAP: Session Announcement Protocol .

[15] `ftp://ftp.isi.edu/in-notes/rfc1889.txt`
RTP: A Transport Protocol for Real-Time Applications (RFC 1889).

[16] `ftp://ftp.isi.edu/in-notes/rfc2327.txt`
SDP: Session Description Protocol (RFC 2327).

[17] International Telecommunication Union. *Visual telephone systems and equipment for local area networks which provide a non-guaranteed quality of service, Recommendation H.323.* Telecommunication Standardization Sector of ITU, Geneba, Switzerland, May 1996.

[18] International Telecommunication Union. *Control protocol for multimedia communication, Recommendation H.245.* Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Feb. 1998.

[19] International Telecommunication Union. *Media stream packetization and synchronization on non-guaranteed quality of service LANs, Recommendation H.225.0.* Telecommunication Standardization Sector of ITU, Geneva, Switzerland, Nov. 1996.

[20] H. Schulzrinne and J. Rosenberg. A Comparision of SIP and H.323 for Internet Telephony. 1998. `http://www.cs.columbia.edu/~hgs/sip/papers.html#sip_h323`.

[21] N. Beijar. *Signaling Protocols for Internet Telephony, Architectures based on H.323 and SIP.* Helsinki University of Technology, Laboratory of Telecommunications Technology, 1998.

[22] `http://www.w3.org/AudioVideo/#SMIL`
SMIL: Synchronized Multimedia Integration Language .

[23] `ftp://ftp.isi.edu/in-notes/rfc2068.txt`
HTTP/1.1: Hypertext Transfer Protocol (RFC 2068).

[24] `ftp://ftp.isi.edu/in-notes/rfc2396.txt`
URI: Uniform Resource Identifiers (RFC 2396).

[25] `ftp://ftp.isi.edu/in-notes/rfc1738.txt`
URL: Uniform Resource Locators (RFC 1738).

[26] `ftp://ftp.isi.edu/in-notes/rfc1890.txt`
RTP Profile for Audio and Video Conferences with Minimal Control (RFC 1890).

[27] `http://www.ietf.org/html.charters/pint-charter.html`
PSTN/Internet Interfaces (pint).

[28] `http://www.ietf.org/internet-drafts/draft-ietf-pint-saint-00.txt`
A proposal for the provisioning of PSTN initiated services running on the Internet.

[29] `http://www.ietf.org/internet-drafts/draft-brusilovsky-icw-00.txt`
A Proposal for Internet Call Waiting Service using SIP.

[30] G. Ratta. Proposal for new Questions SoI. In *delayed Contributions, ITU-T SG 13 IP Experts meeting, Geneva, 31.8. - 9.9.1999.*

[31] `ftp://ftp.isi.edu/in-notes/rfc2458.txt`
Pre-PINT Implementations (RFC 2458).

[32] `http://www.bell-labs.com/mailing-lists/pint/`
Archive of the PINT mailing list .

[33] `ftp://ftp.isi.edu/in-notes/rfc2234.txt`
ABNF: Augmented BNF for Syntax Specifications (RFC 2234).

[34] *Q.763 - Formats and Codes for the ISDN User Part of SS No7.* ITU-T Study Group 11, August 1994.

[35] `ftp://ftp.isi.edu/in-notes/rfc2046.txt`
MIME: Multipurpose Internet Mail Extensions, Part Two: Media Types (RFC 2046).

[36] et al. M. Morrison. *Java 1.1, Unleashed.* Sams.net Publishing, 1997.

[37] `http://java.sun.com`
The Source for Java(TM) Technology .

[38] John R. Levine, Tony Mason, and Doug Brown. *lex & yacc.* O'Reilly & Associates, Inc., USA, 1995. ISBN: 1-56592-000-7.

[39] `http://www.jflex.de/`
JFlex - The Fast Scanner Generator for Java .

[40] `http://www.cs.princeton.edu/~appel/modern/java/CUP/`
CUP Parser Generator for Java .

[41] `http://www.ietf.org/internet-drafts/draft-pint-conf-00.txt`
A proposal for new PINT building blocks with applications to Conference Calling .