Implementation of Caller Preferences in Session Initiation Protocol (SIP)

Master thesis performed in Information Networks Division at Linköping Institute of Technology by

Marcin Dzieweczyński

LiTH-ISY-EX-3580-2004

March 9th, 2004
Implementation of Caller Preferences in Session Initiation Protocol (SIP)

Master thesis performed in
Information Networks Division at Linköping Institute of Technology
by

Marcin Dzieweczyński

LiTH-ISY-EX-3580-2004

Supervisor: George Liu
Examiner: Robert Forchheimer

Linköping March 9th, 2004
Implementation of Caller Preferences in Session Initiation Protocol (SIP)

Marcin Dzieweczynski

Session Initiation Protocol (SIP) arises as a new standard of establishing and releasing connections for a vast variety of multimedia applications. The protocol may be used for voice calls, video calls, video conferencing, gaming and many more. The 3GPP (3rd Generation Partnership Project) suggests SIP as the signalling solution for 3rd generation telephony. Thereby, this purely IP-centric protocol appears as a promising alternative to older signalling systems such as H.323, SS7 or analog signals in PSTN. In contrast to them, SIP does not focus on communication with PSTN network. It is more similar to HTTP than to any of the mentioned protocols. The main standardisation body behind Session Initiation Protocol is The Internet Engineering Task Force (IETF). The most recent paper published on SIP is RFC 3261 [5]. Moreover, there are working groups within IETF that publish suggestions and extensions to the main standard. One of those extensions is “Caller Preferences for the Session Initiation Protocol (SIP)” [1]. This document describes a set of new rules that allow a caller to express preferences about request handling in servers. They give ability to select which Uniform Resource Identifiers (URI) a request gets routed to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining three new request header fields, Accept-Contact, Reject-Contact, and Request-Disposition, which specify the caller preferences. [1]. The aim of this project is to extend the existing software with caller preferences and evaluate the new functionality.

Keyword
SIP, Session Initiation Protocol, Softswitch, Caller Preferences, VoIP
Upphovsrätt

Detta dokument hålls tillgängligt på Internet – eller dess framtida ersättare – under en längre tid från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för ickekommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida [http://www.ep.liu.se/]

Copyright

The publishers will keep this document online on the Internet - or its possible replacement - for a considerable time from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for your own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its WWW home page: [http://www.ep.liu.se/]

© [Publiceringsår/Year of publication, Författarens för- och efternamn/Author]
Abstract

Session Initiation Protocol (SIP) arises as a new standard of establishing and releasing connections for vast variety of multimedia applications. The protocol may be used for voice calls, video calls, video conferencing, gaming and many more.

The 3GPP (3rd Generation Partnership Project) suggests SIP as the signalling solution for 3rd generation telephony. Thereby, this purely IP-centric protocol appears as a promising alternative to older signalling systems such as H.323, SS7 or analog signals in PSTN. In contrast to them, SIP does not focus on communication with PSTN network. It is more similar to HTTP than to any of the mentioned protocols.

The main standardisation body behind Session Initiation Protocol is The Internet Engineering Task Force (IETF). The most recent paper published on SIP is RFC 3261 [5]. Moreover, there are working groups within IETF that publish suggestions and extensions to the main standard. One of those extensions is “Caller Preferences for the Session Initiation Protocol (SIP)” [1].

This document describes a set of new rules that allow a caller to express preferences about request handling in servers. They give ability to select which Uniform Resource Identifiers (URI) a request gets routed to, and to specify certain request handling directives in proxies and redirect servers. It does so by defining three new request header fields, Accept-Contact, Reject-Contact, and Request-Disposition, which specify the caller preferences. [1].

The aim of this project is to extend the existing software with caller preferences and evaluate the new functionality.
Acknowledgements

This thesis would not be created without support of many people. Some of them contributed scientific expertise, others were always there to cheer me up and, encourage to hard work. I hereby take the opportunity to express my sincere thanks to all of them.

First of all, I would not be able to start without my supervisor Professor George Liu who helped me to formulate my topic. Moreover, his assistance was extremely valuable during development of my project.

Second, life would be so empty without friends. Mine always served as great partners during countless coffee breaks, parties and meetings. I would like to thank my roommate – Piotr Mroczkowski – for our conversations, coffee breaks and crazy parties. I also express my thanks to Fatih Kavak - my neighbour at the university and close friend. Thank you for hundreds of coffee breaks, for political and historical lectures and for being “master of complaining”. Finally, thank you for being supportive and for your famous sentence: “don’t worry man”. It is also impossible not to mention my corridor mate and friend Arnold Rombo. Thank you for who you are.

Other people who made my stay in Linköping so wonderful and unforgettable are: Michał Stefański, Kuba Andrzejewski, Mariusz Wzorek, Piotr Rudol, Justyna Nierzwicka, Magda Beszczyńska, Ania Chylińska, Mumin, Krzysztof Bliszczak, Waldemar Basiak, Olga, Ewa Prządka and Dorota Dorożyńska.

Without my beloved parents my studies in Sweden would be impossible. Thank you for your financial support and motivating me. You are the best!

Finally, thank you Ania for your patience...
## Contents

Abstract ........................................................................................................................................ 2  
Acknowledgements .................................................................................................................. 3  
Contents ...................................................................................................................................... 4  
1. Introduction .......................................................................................................................... 5  
2. Background .......................................................................................................................... 7  
   2.1. The Protocol ZOO .......................................................................................................... 7  
   2.2. SIP Infrastructure ......................................................................................................... 8  
   2.3. Description of the protocol syntax ............................................................................. 11  
   2.4. Typical operation scenarios ....................................................................................... 14  
3. Caller Preferences in Session Initiation Protocol .......................................................... 16  
   3.1. Benefits from Caller Preferences ............................................................................... 16  
   3.2. Caller Preferences extension ....................................................................................... 18  
      3.2.1. Accept-Contact and Reject-Contact header fields ........................................... 19  
      3.2.2. Request Disposition ......................................................................................... 21  
   3.3. Matching algorithm ..................................................................................................... 23  
   3.4. Examples ...................................................................................................................... 25  
      3.4.1. Comprehensive example .............................................................................. 25  
      3.4.2. Forcing Audio/Video ...................................................................................... 27  
      3.4.3. Languages .................................................................................................... 28  
      3.4.4. I hate people! ................................................................................................. 29  
4. Software ............................................................................................................................... 30  
   4.1. Survey of the SIP software market .......................................................................... 30  
   4.2. Description of the software used .............................................................................. 31  
      4.2.1. SIP Express Router ....................................................................................... 32  
      4.2.2. KPhone ........................................................................................................ 33  
5. Implementation of Caller Preferences ........................................................................ 35  
   5.1. Changes in KPhone .................................................................................................. 35  
   5.2. Changes in SIP Express Router ............................................................................. 38  
      5.2.1. Parser ............................................................................................................ 38  
      5.2.2. Matching algorithm ....................................................................................... 40  
      5.2.3. Directing to destination set ........................................................................... 40  
6. Testing .................................................................................................................................. 42  
   6.1. “Comprehensive example” test ............................................................................. 43  
   6.2. “Forcing Audio/Video” test .................................................................................... 46  
   6.3. “Languages” test ..................................................................................................... 49  
   6.4. “I hate people!” test .............................................................................................. 50  
7. Conclusions ......................................................................................................................... 52  
Appendix A .............................................................................................................................. 54  
Appendix B .............................................................................................................................. 55  
Appendix C .............................................................................................................................. 57  
List of tables .......................................................................................................................... 60  
List of figures ........................................................................................................................ 61  
References ............................................................................................................................... 63
1. Introduction

The Session Initiation Protocol (SIP) was standardised by the Internet Engineering Task Force (IETF) in early 1999. It initiates, modifies, and terminates network sessions. Currently, most applications of SIP are inherited from traditional telephony. However, the protocol may be used with any kind of sessions. Enough to mention video calls, conferencing, instant messaging, voicemail and distributed games.

In setting up sessions, SIP acts as a signalling protocol, offering services similar to telephony signalling protocols such as Q.931 or ISUP, but in an Internet context. SIP greatly extends the functionality of its predecessors and differs from them in that it does not reserve resources or establish circuits (virtual or real) in the network.

SIP is part of the overall IETF multimedia architecture that has emerged over the past few years. This architecture includes:

- Real-Time Transport Protocol (RTP) for transporting audio, video and other time-sensitive data,
- Real-Time Streaming Protocol (RTSP) for setting up and controlling on-demand media streams,
- Media Gateway Control Protocol (MGCP) and Megaco (also known as H.248) for controlling media gateways,
- Session Description Protocol (SDP) for describing multimedia sessions,
- Session Announcement Protocol (SAP) for announcing multicast sessions,
Telephony Routing over IP (TRIP) for locating the best gateway between the Internet and the PSTN.

SIP is meant primarily to initialise sessions between humans. They will be referred in this project as users. Every user in SIP environment is identified by email-like address. Moreover, any device that can be represented by the host address (most often IP or domain name) can be a part of the session.

The process of session set-up involves the discovery of a user wherever located so that a description of the session (SDP) can be delivered to him. This description is packed as a SIP-request body. Important to note is that session characteristics do not influence the process in any way. The protocol is fully independent and moves most of decisions to the edge of the network.

In SIP users can maintain the same identifier regardless on the number of personal devices attached to the network (personal mobility). URI is assigned by the local SIP-provider. Conversely, a single network terminal or user may be reachable using multiple identifiers (similar to e-mail). The logic in SIP-network elements determines whether requests are delivered to any or all of these network locations. Thus, SIP could be thought of as an application-layer router.

Session initiation also depends on the ability of the called party to make a decision on whether to join or not. Thus, SIP includes information about the caller, the purpose for the invitation, its urgency, and parameters of the session itself.

SIP was published as an IETF proposed standard (RFC 2543) in March 1999. The latest standard (RFC 3261) was released in June 2002 [5]. Since 1999 number of bake-off meetings have taken place in interval of four months. The April 2000 bake-off for example, featured about 60 implementations from 45 different companies. [2].
2. Background

2.1. The Protocol ZOO

The Session Initiation Protocol itself does not provide any real multimedia services. It just does what it was designed for i.e. sets up and tears down the connections. In addition, SIP exchanges information about status of the users in the way similar to Instant Messaging Systems (online, offline etc.). Most popular implementations of SIP use UDP as a transport layer. However, transport over TCP is also possible.

Providing multimedia services over Internet requires the suite of protocols depicted on Figure 2-1.

![Figure 2-1 SIP and accompanying protocols](image)

Above the transport layer one can find a set of real-time protocols. RTP provides end-to-end network transport functions suitable for applications transmitting
real-time data, such as audio, video or simulation data, over multicast or unicast network services.

The data transport is extended by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. ReSerVation Protocol (RSVP) can further extend the Quality of Service. It is used for reservation of resources over the network [3].

The Real Time Streaming Protocol, or RTSP, is an application-layer 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 [4].

### 2.2. SIP Infrastructure

In SIP infrastructure there are a few main functional entities which are necessary for setting up the connection. The RFC 3261 defines them as follows:

- **User Agent Client (UAC)**: A user agent client is a logical entity that creates a new request. The role of UAC lasts only for the duration of that transaction. In other words, if a piece of software initiates a request, it acts as a UAC for the duration of that transaction. If it receives a request later, it assumes the role of a user agent server for the processing of that transaction.

- **User Agent Server (UAS)**: A user agent server is a logical entity that generates a response to a SIP request. The response accepts, rejects, or redirects the request. This role lasts only for the duration of that transaction. In other words, if a piece of software responds to a request, it
acts as a UAS for the duration of that transaction. If it generates a request later, it assumes the role of a user agent client for the processing of that transaction.

- **User Agent (UA):** A logical entity that can act as both a user agent client and user agent server.

- **Redirect Server:** A redirect server is a user agent server that generates 3xx responses to requests it receives, directing the client to contact and alternate set of URIs.

- **Registrar:** A registrar is a server that accepts REGISTER requests and places the information it receives in those requests into the location service for the domain it handles.

- **Proxy, Proxy Server:** An intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. A proxy server primarily plays the role of routing, which means its job is to ensure that a request is sent to another entity "closer" to the targeted user. Proxies are also useful for enforcing policy (for example, making sure a user is allowed to make a call). A proxy interprets, and, if necessary, rewrites specific parts of a request message before forwarding it.

- **Outbound Proxy:** A proxy that receives requests from a client, even though it may not be the server resolved by the Request-URI. Typically, a
UA is manually configured with an outbound proxy, or can learn about one through auto-configuration protocols. [5].

The basic process of setting up the call is shown on the Figure 2-2.

Agent of user 2 (UA2) registers his identity in his local registrar server (step 1). Registrar then looks for the Contact header field in the message (i.e. Contact: UA2@192.168.0.1) and stores current location of the user in Location Server’s database (step 2). Location server usually works on the same machine as registrar. When UA1 wants to issue the call to UA2 it sends the message that traverses through outbound proxy of UA1 (step 3) and proxy of UA2 (step 4). Proxy server of UA2 queries local database (step 5 and 6). If there is a valid location of user 2 proxy forwards message to the particular machine (step 7). Responses sent back from UA2 would typically be 180-Ringing followed by
200-OK. Those responses travel through the same set of proxies (step 8). After this procedure UA1 can finally send ACK message directly to UA2 and so the connection is set up (step 9). The real multimedia transmission is then handled by a set of RTP protocols (step 10).

## 2.3. Description of the protocol syntax

Session Initiation Protocol is ASCII based and in its syntax is very similar to HTTP protocol. Every message consists of three main parts

- method (in requests) or status code (in responses),
- set of header fields,
- empty line (obligatory),
- message body (description of session type in SDP, plain text, html et al.).

The RFC 3261 defines six main methods (REGISTER, INVITE, ACK, OPTIONS, CANCEL and BYE) and number of response codes (200-OK, 180-Ringing, 404-Not Found etc.). Appendix B gives the complete information about methods and response codes.

In the typical case, SIP message header fields convey such information as:

- From and To: addresses of caller and callee,
- Via: routing information,
- Max-Forwards: lifetime of the message (expressed by the maximum number of network nodes that it should go through),
- Call-ID: globally unique identification number for the call,
- **CSeq**: similar to sequence number known from other protocols like TCP. It is incremented for every single message within current call (dialog).

- **Contact**: information about current location of the user (username@hostaddress),

- **Content-Type**: type of the message body (i.e. application/sdp),

- **Content-Length**: number of characters in message body.

The complete list counts tens of header fields. For the sake of brevity they are omitted here and described in Appendix C.

The purpose of message body is to deliver information about multimedia properties of the session to be set up. It is usually written with respect to SDP (Session Description Protocol) syntax. However, it can also be in plain text or html format. We can consider the part of SDP body for instance:

```
c=IN IP4 100.101.102.103
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000.
```

They mean that the user is willing to receive RTP G.711-encoded audio on IP address “100.101.102.103” and port 49172. SDP issues are out of scope of this paper so interested reader should refer to RFC 2327 [6].

When UAC wants to set up the call, tear it down or perform any other kind of action it issues the request message. UAS in turn sends back the response message. The basic process described in previous chapter shows the order of actions taken by SIP elements. It is important to mention that each step on Figure 2-2 usually means transport of slightly different message. By different, one should understand change of method or response code. To some extent header fields are also modified or added during the lifecycle of the message. For
example each server may add *Via* field and by this indicate the exact route of the message. The part that should not change is typically the message body.

To give the reader a feeling about SIP syntax Figure 2-3 shows the popular example of exchange of INVITE and OK messages.

![Figure 2-3 Example request and response message](image)

**INVITE** sip:UserA@there.com SIP/2.0
Via: SIP/2.0/UDP there.com:5060
From: BigGuy <sip:UserA@there.com>
To: LittleGuy <sip:UserB@there.com>
Call-ID: 12345601@there.com
CSeq: 1 INVITE
Subject: Happy Christmas
Contact: BigGuy <sip:UserA@there.com>
Content-Type: application/sdp
Content-Length: 147

v=0
o=UserA 2890844526 2890844526 IN IP4 there.com
s=Session SDP
c=IN IP4 100.101.102.103
t=0 0
m=audio 49172 RTP/AVP 0
a=rtpmap:0 PCMU/8000

**SIP/2.0 200 OK**
Via: SIP/2.0/UDP there.com:5060
From: BigGuy <sip:UserA@there.com>
To: LittleGuy <sip:UserB@there.com>;tag=65a35
Call-ID: 12345601@there.com
CSeq: 1 INVITE
Subject: Happy Christmas
Contact: LittleGuy <sip:UserB@there.com>
Content-Type: application/sdp
Content-Length: 134

v=0
o=UserB 2890844527 2890844527 IN IP4 there.com
s=Session SDP
c=IN IP4 110.111.112.113
t=0 0
m=audio 3456 RTP/AVP 0
a=rtpmap:0 PCMU/8000
2.4. Typical operation scenarios

The simplest scenario is the so called back-to-back (B2B) connection. Caller and callee exchange messages directly without any intermediate servers. The course of the process is shown on the Figure 2-4.

The scenario with two proxy servers and two clients is called trapezoid. In this case order of exchanged messages looks like on Figure 2-5.
Another case takes place when User Agent registers its identity in registrar server. UAC issues message with REGISTER method and includes information about the current location (*Contact* header field). If everything succeeds UAS sends back the OK response. Otherwise response will have appropriate status code - for instance “401-Unauthorized”. Figure 2-6 shows registration process.

![Registration Diagram](image)

*Figure 2-6 Registration*

Session Initiation Protocol incorporates more methods than those mentioned above. Also there are many more scenarios possible. For the clarity only the most typical cases were given here.
3. Caller Preferences in Session Initiation Protocol

3.1. Benefits from Caller Preferences

At its core, SIP is a protocol to facilitate rendezvous of users. The caller and callee need to find each other and then set up the session in order to exchange messages, make a phone calls or video calls. In the future this rendezvous process will be more complicated since users will possess more than one device connected to the network. As an example, they can have cell phone, PDA, fixed SIP phone and PC with one or more communication applications. The question that arises is to which device the call should be forwarded.

The simplest way would be to route the call to all of those devices (the process called parallel forking). However, it is clear that this is not a desired behaviour. Naturally the caller has some preferences about the device he wants to reach. For instance, the choice may be determined by economical considerations (fixed phone may be cheaper then cell phone). The need for caller preferences is even higher on the callee side. Ringing of all the devices at the same time would be definitely too confusing or even annoying. Caller preferences offer more sophisticated solution for this problem. For instance the person who wants to initiate the communication may tell the SIP servers to try to reach fixed phone first, if it fails server may try cell phone and when the callee is still unreachable voicemail message could be sent as the last alternative. These variations are all referred to as find-me/follow-me features.

SIP supports find-me/follow-me features through the SIP registration process. Each device at which a user can be contacted registers to the network. Registrar server keeps all the contacts under user’s Addresses of Record (AOR). AOR is
the SIP URI that in its form resembles email address. The example of AOR is shown below:

<table>
<thead>
<tr>
<th>Contact</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contact 1</td>
<td><a href="mailto:marcin@mobile_company.com">marcin@mobile_company.com</a></td>
</tr>
<tr>
<td>Contact 2</td>
<td><a href="mailto:marcin@pstn_company.com">marcin@pstn_company.com</a></td>
</tr>
<tr>
<td>Contact 3</td>
<td>my_pc@192.168.82.205</td>
</tr>
</tbody>
</table>

Table 3-1 Example address of record

In this case company.com is SIP provider while mobile_company is mobile network operator, pstn_company is PSTN network and the last contact is the PC connected to any IP-based network.

It is important to note that there is already a solution for expressing preferences of callee. Call Processing Language (CPL) scripts let users to formulate the routing algorithm and send it to their local SIP server. However, the called party is not the only one with preferences. As was mentioned before, a caller will also have preferences for how they want their call to be routed.

In today's circuit-switched telephony networks, users have multiple devices, and each device is associated with its own phone number. All these numbers have to be exchanged between friends or business partners. Moreover, when you want to make a call you are forced to dial one number after another manually. This is inefficient, and difficult.
As an alternative, a user can have a single address as shown in the Table 3-1. This is the one and only address they give out to other users on their business cards. The caller willing to call somebody just chooses the URI address and expresses his preferences in INVITE message (i.e. mobile, fixed, audio or video). Of course, the user does not need to know SIP syntax. He may just be prompted to choose destination device from the pop-up list.

This one number service is the most important justification of use of caller preferences. [9]

### 3.2. Caller Preferences extension

Caller preferences extension has the status of Internet draft and is published by “SIP charter” working group. [8]

The paper defines three new header fields: `Accept-Contact`, `Reject-Contact` and `Request-Disposition`. Preferences included in `Accept-Contact` and `Reject-Contact` are called feature preferences. They describe the desired properties of a UA that the request is to be routed to. On the other hand, `Request-Disposition` header specifies the preferred treatment of the message at the server.

When user wants to express his preferences its UAC has to attach one or more `Accept-Contact/Reject-Contact` fields and/or one `Request-Disposition` field to the message. They can be put into any message except those with REGISTER method. In fact, no more actions are required from the client.

On the other hand, when UAS compliant with caller preferences extension receives request that corresponds to one of its registered contacts it should behave as if it were the proxy for the domain in request URI [1]. The processing
can be split into two parts. One of them applies to directives in
Request-Disposition (see chapter 3.2.1.). Those are related strictly to message
routing. On the other hand, when Accept-Contact or Reject-Contact header
fields are present, and server has the particular AOR (Address of Record), it
should extract all preferences and build the so called matching predicate. Next
the server builds predicates for all contacts stored under the AOR. After that,
server compares matching predicate (caller preferences) against contact
predicates (callee capabilities). Matching operation is performed according to
special algorithm described in chapter 3.3.

It is important to note that proxy server must not modify Accept-Contact and
Reject-Contact headers. However, it may add one if not present, or add a value
to an existing header field, as if it were a UAC. This is useful for a proxy to
request processing in downstream proxies in the implementation of a feature [1].

3.2.1. Accept-Contact and Reject-Contact header fields

Accept-Contact and Reject-Contact carry the set of parameters. Header field
body starts with “*” followed by list of parameters separated with semicolon.
Star mark is required by the guidelines for SIP extensions. It lets servers that are
not compliant with caller preferences to skip those header fields and so prevents
them from syntax error. Example of Accept-Contact and Reject-Contact fields is
shown below:

Accept-Contact: *;mobility="mobile";methods="INVITE"
Reject-Contact: *;actor="msg-taker";video
As can be seen from the example every parameter may be followed by the value. There are four types of values allowed: string, token, numerical and boolean. Strings and tokens are similar with the exception that token is chosen from the fixed set of values while string can be anything. Moreover, tokens are treated in case-insensitive and strings in case-sensitive manner. Numerical parameters are followed by relation mark ("=", ">=" or "<=") and value. Boolean parameters can be considered as tokens with two possible values: “true” and “false”. If not followed by any value they are treated as “true”.

It is obvious that parameters names cannot be chosen freely. If they were, matching operation would almost ever fail. Instead, the names should be to some extent standardised. Most of parameters suggested by IETF can be found in Callee-Capabilities internet draft [10]. Those are: "audio", "automata", "class", "duplex", "data", "control", "mobility", "description", "events", "priority", "methods", "schemes", "application", "video", "actor", "language", "isfocus" and "type".

Not to be too prohibitive, IETF engineers allowed presence of parameters which are not defined under condition that they must be preceded by “+” sign.

There are two very important parameters that influence the results of matching operation in special way. One is “require” and another is “explicit”. Because of their significance they will be described in the “Matching algorithm” section (see chapter 3.3).
Table 3-2 gives examples of parameters.

<table>
<thead>
<tr>
<th>Example</th>
<th>Type of value</th>
<th>Processing</th>
</tr>
</thead>
<tbody>
<tr>
<td>audio</td>
<td>boolean</td>
<td>Will be assigned “true” value first</td>
</tr>
<tr>
<td>audio=“true”</td>
<td>boolean</td>
<td>Check if true or false, case insensitive</td>
</tr>
<tr>
<td>audio=“false”</td>
<td>boolean</td>
<td></td>
</tr>
<tr>
<td>description=“happy birthday”</td>
<td>string</td>
<td>Case sensitive matching</td>
</tr>
<tr>
<td>class=“business”</td>
<td>token</td>
<td>Case insensitive matching</td>
</tr>
<tr>
<td>class=“personal”</td>
<td>token</td>
<td></td>
</tr>
<tr>
<td>mobility=“fixed”</td>
<td>token</td>
<td></td>
</tr>
<tr>
<td>mobility=“mobile”</td>
<td>token</td>
<td></td>
</tr>
<tr>
<td>priority=10</td>
<td>numerical</td>
<td>Matching according to arithmetic rules</td>
</tr>
<tr>
<td>q=0.1</td>
<td>numerical</td>
<td></td>
</tr>
<tr>
<td>+other_param&gt;=10</td>
<td>other numerical</td>
<td></td>
</tr>
<tr>
<td>+other_param=“foo”</td>
<td>other string</td>
<td>Case sensitive matching</td>
</tr>
</tbody>
</table>

**Table 3-2** Accept-Contact and Reject-Contact parameters examples

### 3.2.2. Request Disposition

*Request-Disposition* syntax is relatively simple. Server needs only to detect six possible tokens: “proxy”/”redirect”, “cancel”/”no-cancel”, “fork”/”no-fork”, “recurse”/”no-recurse”, “parallel”/”sequential”, ”queue”/”no-queue”. Again they are preceded by “*” mark. Example Request-Disposition header field may look like the one below:

Request-Disposition: *;proxy; recurse; parallel

The definitions of the tokens are as follows:

- **proxy-directive**: indicates whether the caller would like each server to proxy ("proxy") or redirect ("redirect"). Proxy simply forwards message to another hop trying to find the destination while redirect server sends the response with alternative destination back to UAC.
- **cancel-directive**: when “cancel” is present each server sends CANCEL message downstream which causes all unused branches to be removed (normal behaviour). When caller sends “no-cancel” directive server should not cancel any outstanding branches on receipt of a 2xx.

- **fork-directive**: indicates whether server should ("fork") request to all known contacts, or proxy to only a single best address ("no-fork"). The directive is ignored if "redirect" has been requested.

- **recurse-directive**: if “recurse” directive is present the proxy receiving redirect response (3xx) should send requests to all addresses listed in this response. Otherwise, it would forward the list of addresses upstream towards the caller ("no-recurse"). The directive is ignored if "redirect" has been requested.

- **parallel-directive**: If the server is forking it can send all the request at once ("parallel") or send the request only after it received response for the previous one ("sequential"). The directive is ignored if "redirect" has been requested.

- **queue-directive**: If UAS is busy, the caller can indicate that it wants to have its call queued ("queue") or rejected immediately ("no-queue"). If the call is queued, the server returns "182 Queued".[1]
3.3. Matching algorithm

As was mentioned in Chapter 3.2 when proxy server receives request it should apply matching algorithm whenever the following conditions are fulfilled:

- proxy is the owner of the domain from request-URI,
- there are one or more Accept-Contact/Reject-Contact header fields in the request,
- there is Address of Record that matches request-URI and contacts stored under this AOR.

When such situation happen server has to create the list of contacts ordered from the best one to the worst one. This list is called destination set.

In the first step server temporarily removes contacts that are immune for the caller preferences i.e. have no callee capabilities defined (address without any parameters). Remaining contacts are matched against Reject-Contact header fields. If all the parameters in Reject-Contact are the same as contact’s parameters this particular contact is removed and will be not considered later. After that there is time for computing score for each contact. This is done based on Accept-Contact parameters. Each contact is compared against each Accept-Contact header field. Let’s assume that there are N parameters in the list. Every single parameter that matches increments score by 1/N. Thus for every single header field the contact score varies from 0 to 1. This operation is repeated for each Accept-Contact. The final value is the average from all Accept-Contact fields. This value is referred to as Qa.

There is a special role of two parameters mentioned earlier in chapter 3.2.1 i.e. “require” and “explicit”. The Figure 3-1 explains how those parameters influence destination set.
If the score for particular header is lower than 1 and “explicit” is present the score is set to 0. If there is also “require” parameter the contact is dropped. When there is only “require” parameter the contact is dropped if and only if callee explicitly shown that it doesn’t support the particular feature. For example:

Accept-Contact: *;video;require

would eliminate the contact:
contact1@example.com;video="false",
but, would not eliminate the contact:
contact2@example.com;audio.

As another example:
Accept-Contact: *;video;explicit;require

would eliminate the contact:
contact1@example.com;audio
but, would not eliminate the contact:
contact2@example.com;audio;video.

The usage of “explicit” and “require” parameters will become more clear after thorough analysis of examples given in next chapter.

At this stage contacts immune to caller preferences are put back to destination set with Qa=1. Next, destination set is sorted. In the first round the sorting is done with respect to q-value. This is the parameter that callee can assign to its contacts and by this indicate which contact is most important for him. When no q parameter is present it is assigned 1.

Equivalence classes (group of contacts with the same q-value) are then sorted according to Qa. The final destination set is then used for routing of the current request.

The simplified diagram of matching algorithm is shown on the Figure 3-1.
3.4. Examples

3.4.1. Comprehensive example

AOR
sip:user@example.com

Registered contacts
sip:u1@h.example.com;audio;video;methods="INVITE,BYE";q=0.2
sip:u2@h.example.com;audio="FALSE";methods="INVITE";actor="msg-taker";q=0.2
sip:u3@h.example.com;audio;actor="msg-taker";methods="INVITE";video;q=0.3
sip:u4@h.example.com;audio;methods="INVITE,OPTIONS";q=0.2
INVITE message (only a part) sent to sip:user@example.com

Reject-Contact: *;actor="msg-taker";video
Accept-Contact: *;audio;require
Accept-Contact: *;video;explicit
Accept-Contact: *;methods="BYE";class="business";q=1.0

Processing at the server example.com.

Server first removes u5 because it is immune from caller preferences ("q" doesn’t count as a caller preferences parameter). The next step – processing of Reject-Contact – results in removal of u3 because only this contact explicitly declared that it is a “msg-taker” and it supports “video”. We have now u1, u2 and u4 left.

Accept-Contact is applied to those contacts. Matching algorithm should give results like in the Table 3-3

<table>
<thead>
<tr>
<th>Accept-Contact: *;audio;require</th>
<th>u1</th>
<th>u2</th>
<th>u4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accept-Contact: *;video;explicit</th>
<th>u1</th>
<th>u2</th>
<th>u4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Accept-Contact: *;methods=&quot;BYE&quot;;class=&quot;business&quot;;q=1.0</th>
<th>u1</th>
<th>u2</th>
<th>u4</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>-</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

| Qa: 0.83 discarded | 0.5 |

Table 3-3 Results of the matching algorithm

After processing of Accept-Contact we have u1 and u4 left. At this stage u5 is put back to destination set with default Qa=1. The situation is now:

- u1, q=0.2, Qa=0.83
- u4, q=0.2, Qa=0.5
- u5, q=0.5, Qa=1
After grouping into equivalence classes (q-value) and sorting by Qa we come to final destination set:

- u5 – will be tried first,
- u1 – will be tried second,
- u4 – will be tried third [1].

### 3.4.2. Forcing Audio/Video

The user Y registered contact Y1:

REGISTER sip:example.com SIP/2.0
To: sip:Y@example.com
Contact: <sip:Y1@pc.example.com>;q=1.0
;methods="INVITE,BYE,OPTIONS,ACK,CANCEL"
;uri-user="<Y1>"
;uri-domain="example.com"
;audio
;schemes="sip,tel"
;mobility="fixed"
;class="business"

and Y2:

REGISTER sip:example.com SIP/2.0
To: sip:Y@example.com
Contact: <sip:Y2@pc.example.com>;q=0.6
;methods="INVITE,BYE,OPTIONS,ACK,CANCEL"
;uri-user="<Y2>"
;uri-domain="example.com"
;audio
;video
;schemes="sip,tel"
;mobility="fixed"
;class="business"

Let’s assume that we want the call to fail unless the callee can support video communication. The INVITE message should look like below:

INVITE sip:Y@example.com SIP/2.0
Accept-Contact: *;video;require;explicit
There are two contacts for Y@example.com. However, the call would be forwarded only to Y2 since this contact explicitly declared support for “video”. “Require” and “explicit” flags cause Y1 to be discarded. If they were not present both contacts would be tried. Y1 would earn the score of 0.5 and Y2 the score of 1. However, because of higher q-value Y1 would be tried first (although it doesn’t support video). Thus, the desired behaviour would not be achieved [9].

3.4.3. Languages

Suppose, there is an international office with some people speaking fluent English, some fluent Spanish and some fluent Spanish and a little English. So, the English speaking employee would register as:

REGISTER sip:example.com SIP/2.0
To: sip:Y@example.com
Contact: sip:Y1@pc.example.com;languages="en"

Employee that speaks Spanish and a little bit of English would register as:

REGISTER sip:example.com SIP/2.0
To: sip:Y@example.com
Contact: sip:Y2@pc2.example.com;languages="es"
Contact: <sip:Y2-en@pc2.example.com>;languages="en";q=0.2

And finally employee that speaks Spanish and English fluently would register as:

REGISTER sip:example.com SIP/2.0
To: sip:Y@example.com
Contact: sip:Y3@pc3.example.com;languages="es,en"

The INVITE message send to the company is shown below:

INVITE sip:Y@example.com SIP/2.0
Accept-Contact: *;languages="en";require

As we can see caller wishes to speak in English. Contact Y1, Y2-en and Y3 all match the requested preferences with Qa=1. Y2 doesn’t match and because of
“require” flag it is discarded. Now, remaining contacts are sorted. One equivalence class is Y1 and Y3 with default q-value=1. Sorting doesn’t change anything because all the contacts have Qa=1. Y2-en has q-value=0.2 and so it is in second equivalence class. Thus, Y1 and Y3 (fluent English) would be tried first and if the call fails Y2-en will be the last choice (a little bit of English). This perhaps the behaviour that potential customer would desire [9].

3.4.4. I hate people!

The caller that doesn’t actually want to speak to people and just wishes to leave the message can send INVITE with the form:

INVITE sip:Y@example.com SIP/2.0
Accept-Contact: *;msgserver;require;explicit

The result is that all the contacts would fail except those that had explicitly shown they had msgserver functionality [9].
4. Software

4.1. Survey of the SIP software market

Although SIP is relatively new technology there is already plenty of implementations available on the market. We can divide them with respect to the programming language or type of the software. As long as the language is concerned there are two main groups. Java society promotes its special SIP API that is part of the JAIN technology. JAIN is the set of API’s for rapid development of telecommunication protocols. On the other hand number of developers still use C/C++ as the language of choice. Mostly, due to its speed and performance.

Apart from programming technique we can distinguish two types of SIP software: complete application on one side and programming interface (API) on the other. Programming interfaces usually implement the SIP stack that let programmer focus on functionality rather than on protocol syntax and structure. The other group is represented by ready-to-use aplications like SIP phones and SIP servers.

Table 4-1 shortly summarizes some of the solutions available on the SIP market.
4.2. Description of the software used

The choice of software for my project was determined mostly by the operating system i.e. Linux Red Hat 8. Of course, it must be open source software. After extensive Internet search the following two products were chosen: SIP Express Router (SER) and KPhone. SER server got really good reviews from its users and besides it provided ways to extend the functionality, which was highly desirable. On the client side number of applications were evaluated and KPhone seemed to be of highest quality among them.

<table>
<thead>
<tr>
<th>Company / software name</th>
<th>Webpage</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SIP stacks</strong></td>
<td></td>
</tr>
<tr>
<td>GNU oSIP</td>
<td><a href="http://www.gnu.org/software/osip/">http://www.gnu.org/software/osip/</a></td>
</tr>
<tr>
<td>GNU eXoSIP</td>
<td><a href="http://savannah.nongnu.org/projects/exosip/">http://savannah.nongnu.org/projects/exosip/</a></td>
</tr>
<tr>
<td>SIP from vovida.org</td>
<td><a href="http://www.vovida.org/protocols/downloads/sip/">http://www.vovida.org/protocols/downloads/sip/</a></td>
</tr>
<tr>
<td>resiprocate</td>
<td><a href="http://resiprocate.sourceforge.net/">http://resiprocate.sourceforge.net/</a></td>
</tr>
<tr>
<td>Microsoft RTC API</td>
<td><a href="http://www.microsoft.com">http://www.microsoft.com</a></td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td></td>
</tr>
<tr>
<td>Linphone (client)</td>
<td><a href="http://www.linphone.org/">http://www.linphone.org/</a></td>
</tr>
<tr>
<td>Kphone (client)</td>
<td><a href="http://www.wirlab.net/kphone/index.html">http://www.wirlab.net/kphone/index.html</a></td>
</tr>
<tr>
<td>Siphon (client)</td>
<td><a href="http://siphon.sourceforge.net/index.html">http://siphon.sourceforge.net/index.html</a></td>
</tr>
<tr>
<td>Partysip (server)</td>
<td><a href="http://www.nongnu.org/partysip/partysip.html">http://www.nongnu.org/partysip/partysip.html</a></td>
</tr>
<tr>
<td>SIP Express Router (server)</td>
<td><a href="http://www.iptel.org/ser/">http://www.iptel.org/ser/</a></td>
</tr>
<tr>
<td>CINEMA</td>
<td><a href="http://www.cs.columbia.edu/IRT/cinema/">http://www.cs.columbia.edu/IRT/cinema/</a></td>
</tr>
<tr>
<td><strong>JAVA</strong></td>
<td></td>
</tr>
<tr>
<td>JAIN-SIP</td>
<td><a href="https://jain-sip.dev.java.net/">https://jain-sip.dev.java.net/</a></td>
</tr>
<tr>
<td>Dynamicsoft SIP Stack</td>
<td><a href="http://www.nine-9s.com/prod_SIP_protocol_stack.htm">http://www.nine-9s.com/prod_SIP_protocol_stack.htm</a></td>
</tr>
<tr>
<td><strong>Applications</strong></td>
<td></td>
</tr>
<tr>
<td>SIP communicator</td>
<td><a href="https://sip-communicator.dev.java.net/">https://sip-communicator.dev.java.net/</a></td>
</tr>
<tr>
<td>SIP Center Ubiquity UA</td>
<td><a href="http://www">http://www</a> sipcenter.com/testarea/testdownload.s.html</td>
</tr>
</tbody>
</table>

**Table 4-1 SIP implementations**
4.2.1. SIP Express Router

SIP Express Router is the complete and highly flexible server developed by iptel.org group [6]. It is written in pure C and designed for Linux platform. It can act as a proxy, redirect or registrar server. SER consists of the core and set of modules. Some of the modules are: registrar (registrar server), usrloc (usr location interface), dbtext (for handling database control), sms (Short Message System), textops (for text operations like searching, comparison etc.). Some of the modules export functions that may be later used in special script language provided in SER.

One way of extending SER functionality is to write configuration script. More advanced users can create their own modules and port them to the rest of package. Such a module when exports the functions or variables can be used in configuration files. Nice example is the textops module that simplifies the text operations. It has functions for searching the particular header field, string etc. Furthermore, with the configuration files one can determine whether the server should behave as the registrar server, redirect server, SMS gateway and many more. The example of redirect server implementation is given below in Figure 3-1.
KPhone was written by Billy Biggs from Wirlab [7]. It is designed for Linux platform and written in C++. It uses its own SIP stack called “dissipate2”. The functionality encompasses implementation of SIP compatible with latest RFC 3261, presence, instant messaging, audio calls and to some extent video calls. The Figure 3-2 shows the screenshot of the KPhone in action.

**Figure 3-1** Example of SER configuration

```plaintext
# $Id: redirect.cfg,v 1.5 2002/12/09 02:32:57 jiri Exp $
#
# this example shows use of ser as stateless redirect server
#
# ------------------ module loading ----------------------------------
loadmodule "modules/sl/sl.so"

# ------------------------- request routing logic -------------------
# main routing logic
route {
    # for testing purposes, simply okay all REGISTERs
    if (method="REGISTER") {
        log("REGISTER");
        sl_send_reply("200", "ok");
        break;
    };
    # rewrite current URI, which is always part of destination ser
    rewriteuri("sip:parallel@iptel.org:9");
    # append one more URI to the destination ser
    append_branch("sip:redirect@iptel.org:9");
    # redirect now
    sl_send_reply("300", "Redirect");
}
```

**4.2.2. KPhone**

KPhone was written by Billy Biggs from Wirlab [7]. It is designed for Linux platform and written in C++. It uses its own SIP stack called “dissipate2”. The functionality encompasses implementation of SIP compatible with latest RFC 3261, presence, instant messaging, audio calls and to some extent video calls. The Figure 3-2 shows the screenshot of the KPhone in action.
Figure 3-2 Screenshot of KPhone
5. Implementation of Caller Preferences

Implementation of any protocol standardised by IETF does not leave much freedom to developer. One has to follow the RFC papers or Internet drafts very carefully in order to be consistent with variety of applications from different vendors. This was also the case in my project. The paper, which I referred to most often, was Internet draft describing Caller Preferences extension [1].

Moreover, I had to study Session Initiation Protocol standard from RFC 3261.

Finally, I had to spend hundreds of hours studying programmer’s manual for SIP Express Router, and tens of C/C++ files from both SER server and KPhone distributions.

Only after those preparations it was possible to decide on a strategy for implementation of Caller Preferences.

5.1. Changes in KPhone

Following the Caller Preferences draft the first thing to do was the change of UAC behaviour. The version of KPhone I started to work with (3.14) was consistent with RFC 3261. However, there were two things to implement. First of all, the application had to be able to attach Accept-Contact, Reject-Contact and Request-Disposition header fields when required. Since these header fields are not allowed in any request (i.e. they are not present in REGISTER message) and they should be attached only when user wishes to do so, some interference in SIP stack was necessary.

On the application level additional dialog box was necessary for manual configuration of new header fields (see Figure 5-1). Manual configuration was a
choice determined by two factors. First, the project is based on the draft, which is under constant modifications. Second, the testing that was to come. Because of that, hard coded and fully automatic solution was not acceptable.

Another part of KPhone that had to be changed was registration. The original application sent REGISTER request with contact URI and “q” parameter only. Testing of caller preferences however, required some more parameters (callee capabilities) to be registered. The modified dialog box is shown on Figure 5-2.
After these modifications, the KPhone client was ready to send messages consistent with “Caller Preferences” extension. The pictures below show the sample messages sent by the modified application. One can note additional header fields on Figure 5-3 and REGISTER message with Contact header fields and set of parameters on Figure 5-4.

**Figure 5-2** Additional field for registering contact with parameters

**Figure 5-3** Example message with new header fields
5.2. Changes in SIP Express Router

5.2.1. Parser

The most critical part at the server side (in terms of speed) is the parser. Since, SER is a fully functional server it already has very fast parser built-in. The native SER solution is based on hash table. The header fields are translated into integers and arithmetic comparison is used instead of “strcmp” function. This speeds up the parsing operation approximately three times.

Native parser however, had to be adapted to store the new headers (Accept-Contact, Reject-Contact and Request-Disposition). Server puts all unknown headers into special list, but searching through this list in further
processing would be ineffective. Thus the additional items to the hash table were added.

Next task was to extract all the parameters from contacts stored under particular AOR and from *Accept-Contact* and *Reject-Contact*. This functionality was built into “registrar” module. SER Registrar has two important functions i.e. “save” and “lookup”. Save function is used when REGISTER request arrives and server has to store the contact. Lookup function searches through all stored contacts that match request-URI. Both “save” and “lookup” functions were extended to extract contact and request parameters respectively. Both of them call the parser which extracts all information strictly according to syntax rules. After extraction all parameters are put into the one-way list.

Every list was associated with the data structures already used in SER. Particularly those representing contacts and header fields. Thanks to this, parameters could be accessed from other functions. For instance from matching algorithm.

For the sake of *Request-Disposition* header field the textops module was extended with “is_present_param” function, and registrar module with special parameter. The function is specialized to search for tokens in *Request-Disposition* field i.e. “proxy”, “cancel”, “queue” etc. Parameter is used to activate and deactivate Caller Preferences extension which is useful for debug purposes.

Although this way of parsing *Request-Disposition* is not the fastest it was desired because of the planned use in configuration script. As was mentioned before these scripts let user to define routing logic. Since *Request-Disposition* directives are purely routing-related, the special function altogether with script language functionality provided an elegant solution.
5.2.2. Matching algorithm

Matching algorithm was explained in detail in Section 3.3. My implementation followed exactly guidelines from Caller Preferences draft. Matching algorithm was written in separate file called “score.c” which has four functions: score, is_match, sort_target_set and print_target_set.

Score function is called only when server finds Caller Preferences headers. It goes through all the contacts, all Reject-Contact and Accept-Contact header fields and all parameters within those headers. It eliminates contacts that are immune to caller preferences and drops contacts that match Reject-Contact.

Is_match function do the actual matching on every rule-entry pair. Rule means the parameter from the request and entry means parameter from contact stored on the server. This function also detects “require” and “explicit” flag. Matching is based on hash tables to speed-up execution of the algorithm.

Sort_target_set sorts the remaining contacts (destination set).

Print_target_set is used only for debug purposes and shows the destination set and its ordering.

5.2.3. Directing to destination set

Matching algorithm is executed on arrival on every request compliant with Caller Preferences. It means that destination set should be built every time such event happens. However, server should keep all the contacts in database no matter if destination set was empty or included part of the contacts or all of
them. Creating separate data structure for each destination set would be inefficient in terms of memory usage and speed.

Therefore a special flag was added to each contact. Only this flag indicates whether the particular contact is included in current destination set or not. Registrar’s lookup function then appends branches only for those URIs that are marked.
6. Testing

The testbed was set up on two computers:

- Server and Client: Pentium 4, 2.8 GHz with 512MB memory and Linux RedHat 8 system
- Client, AMD K6 450MHz and Linux Debian system.

As was mentioned before the most critical changes were made in the server software. Written entirely in C SIP Express Router incorporated complicated memory management. The functionality implemented by me required a new data structures. Some of them, like “contact” data structures had to be persistent over time limited by “expire” parameter (typically hundreds of seconds). Others had to be available for the time of the current transaction. There were also local data structures valid only within the function.

All of these reasons required careful planning of memory allocation and release in order to avoid memory leaks. A lot of string operations made the memory management even more complicated. However, after removing all the bugs and memory leaks the server worked stable and its performance was not influenced by changes made.

The correctness of caller preferences extension (parsing, matching algorithm and directing to the destination set) was observed in debug mode and proved to work properly. Matching algorithm was tested with the examples given in [9]. For all of them the desired behaviour was achieved.

Below are some screenshots taken for the four examples given before in chapter 3.4.
Since there was no domain name for the server, hosts were identified by IP addresses instead. Separate clients can be recognized by port numbers in Contact header field (5062 and higher).

### 6.1. “Comprehensive example” test

Figures 6-1 to 6-5 show the registration messages sent from five clients to the server. They are equal to the contacts given in chapter 3.4.1. except that KPhone client adds the “methods” parameter with all supported methods automatically. It causes that the last contact is not immune to caller preferences and so, it is the subject to matching algorithm. That is why the score for this contact is not 1.

**Figure 6-1** “Comprehensive example” - registration of client 1

```plaintext
SipClient: Sending: 18:34:45.214
REGISTER sip:192.168.82.205 SIP/2.0
Via: SIP/2.0/UDP 192.168.82.205:5062;rport
CSeq: 2220 REGISTER
To: <sip:dziwias0192.168.82.205>
Expires: 900
From: <sip:dziwias0192.168.82.205>
Call-ID: 1133297760192.168.82.205
Content-Length: 0
User-Agent: KPhone/3.14
Event: registration
Allow-Events: presence
Contact: <sip:mark5810@192.168.82.205;transport=udp>;q=0.2;method="INVITE,MESSAGE,INFO,SUBSCRIBE,OPTIONS,BYE,CANCEL,NOTIFY,ACK";audio;video
```

**Figure 6-2** “Comprehensive example” - registration of client 2

```plaintext
SipClient: Sending: 18:34:57.784
REGISTER sip:192.168.82.205 SIP/2.0
Via: SIP/2.0/UDP 192.168.82.205:5064;rport
CSeq: 7570 REGISTER
To: <sip:dziwias192.168.82.205>
Expires: 900
From: <sip:dziwias0192.168.82.205>
Call-ID: 109721822192.168.82.205
Content-Length: 0
User-Agent: KPhone/3.14
Event: registration
Allow-Events: presence
Contact: <sip:mark5810192.168.82.205:5064;transport=udp>;q=0.2;method="INVITE,MESSAGE,INFO,SUBSCRIBE,OPTIONS,BYE,CANCEL,NOTIFY,ACK";audio="FALSE";msgserver
```
Figure 6-3 “Comprehensive example” - registration of client 3

Figure 6-4 “Comprehensive example” - registration of client 4

Figure 6-5 “Comprehensive example” - registration of client 5
Figure 6-6 shows the message sent to the server. Again, \textit{Accept-Contact} and \textit{Reject-Contact} headers are identical to those given in chapter 3.4.1.

Finally figure 6-7 shows the screenshot of the server in debug mode. One can observe the parameters extracted from the request and the ordered destination set. “Destination set” flag (equal 1) means that message will be routed to this particular contact.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6-6.png}
\caption{Figure 6-6 “Comprehensive example” - request message}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure6-7.png}
\caption{Figure 6-7 “Comprehensive example” - server output}
\end{figure}
6.2. “Forcing Audio/Video” test

Description of the following example can be found in chapter 3.4.2. The registration messages for contacts Y1 and Y2 are shown below. Y1 supports only “audio” while Y2 supports “audio” and “video” (Contact header field).

```
SIPClient: Sending: 18:48:59.725
REGISTER sip:192.168.82.205 SIP/2.0
Via: SIP/2.0/UDP 192.168.82.205;5064;rport
CSeq: 8712 REGISTER
To: <sip:dziewas@192.168.82.205>
Expires: 900
From: <sip:dziewas@192.168.82.205>
Call-ID: 10830351916192.168.82.205
Content-Length: 0
User-Agent: KPhone/3.14
Event: registration
Allow-Events: presence
Contact: <sip:nardz@192.168.82.205:5064;transport=udp>;q=0.6;method="INVITE,MESSAGE,INFO,SUBSCRIBE,OPTIONS,BYE,CANCEL,NOTIFY,ACK";uri-user="<Y1">;uri-domain="example.com";audio;video;scheme="sip,tel";mobility="fixed";class="business"
```

**Figure 6-8** “Forcing audio/video” example – registration of contact 1

```
SIPClient: Sending: 18:44:06.520
REGISTER sip:192.168.82.205 SIP/2.0
Via: SIP/2.0/UDP 192.168.82.205;5062;rport
CSeq: 6202 REGISTER
To: <sip:dziewas@192.168.82.205>
Expires: 900
From: <sip:dziewas@192.168.82.205>
Call-ID: 1967240416@192.168.82.205
Content-Length: 0
User-Agent: KPhone/3.14
Event: registration
Allow-Events: presence
Contact: <sip:nardz@192.168.82.205:5062;transport=udp>;q=0.6;method="INVITE,MESSAGE,INFO,SUBSCRIBE,OPTIONS,BYE,CANCEL,NOTIFY,ACK";uri-user="<Y2">;uri-domain="example.com";audio;video;scheme="sip,tel";mobility="fixed";class="business"
```

**Figure 6-9** “Forcing audio/video” example – registration of contact 2
Message sent without “require” and “explicit” flag is shown on figure 6-10.

In result of this message server puts both contacts into destination set (see figure 6-11). Even worse, Y1 which does not support “video” is tried first which is definitely not desired.

Figure 6-10 “Forcing audio/video” example – request message 1

---

Features extracted from the Contacts

| sip:marcin@192.168.82.205:5066;transport=udp |
| class, value: business, |
| mobility, value: fixed, |
| schemes, value: sip,tel, |
| audio, value: TRUE, |
| uri-domain, value: example.com, |
| uri-user, value: <Y1>, |
| methods, value: INVITE,MESSAGE,INFO,SUBSCRIBE,OPTIONS,BYE,CANCEL,NOTIFY,ACK, |

Features extracted from Accept-Contact headers

1 Accept-Contact
| methods, value: INVITE, |
| video, value: TRUE, |

Features extracted from Reject-Contact headers

Target set

| sip:mardz58192168.82.205:5066;transport=udp, q-value: 1.000000, Ga: 0.500000, destination set: 1 |
| sip:mardz58192168.82.205:5066;transport=udp, q-value: 0.000000, Ga: 1.000000, destination set: 1 |

Figure 6-11 “Forcing audio/video” example – server output 1
When the message with “require” and “explicit” flag was sent the contact Y1 was removed from destination set which was desired and at the same time proved that the matching algorithm worked as expected. Figures 6-12 and 6-13 show the modified request message and the screenshot from the server.

![Figure 6-12 “Forcing audio/video” example – request message 2](image1)

![Figure 6-13 “Forcing audio/video” example – server output 2](image2)
6.3. “Languages” test

In the example from chapter 3.4.3. we assumed that there is a multilingual company and three of its employees register their identities. Y1 and Y3 indicate that they speak English. Y2 registers two contacts. One of them specifies that he speaks fluent Spanish, and other specifies that he also knows a little bit of English (q-value equal 0.2). The customer indicates in his message that he wants to talk with English speaking person (Figure 6-14).

Figure 6-15 shows the resulting destination set. One can observe that only contacts which support English language were chosen (“require” flag eliminates other). Moreover, the employee which declared that he speaks only a little of English (lower q-value) is tried in the last turn.
6.4. “I hate people!” test

In this fairly simple example there is a couple of contacts registered with different parameters. Only one of them declared it is a message server. The message shown on figure 6-16 could be sent by the person who does not want to speak and prefers to leave a message.
The screenshot from the desktop shows that message was directed only to the client which had “msgserver” parameter in its *Contact* header field.

**Figure 6-17** “I hate people!” example – desktop screenshot
7. Conclusions

The purpose of this project was to show advantages of Session Initiation Protocol and its Caller Preferences extension in particular. The SIP is an emerging standard created by IETF and has been already approved for applications in 3rd generation networks.

When analysing SIP advantages one should look at this protocol from the perspective of IP-convergence. Convergence is nowadays the main issue in telecommunication. There is a huge pressure for the shift from separated PSTN, ISDN and Internet infrastructure to all-IP network. The ubiquitous and unified network with its flexibility and variety of services would be perhaps the real revolution in telecommunication.

In this context, softswitch is the term that cannot be neglected. Softswitch is the counterpart of PSTN/ISDN switches. In contrast to centralised architecture with the “big-steel” devices occupying whole buildings, SIP and softswitching technique are rather about distributed, Internet-based infrastructure. Switching workload is moved to the edges of network and so, can be handled by any middle-class server.

In fact, SIP is all about switching or more appropriately – routing. In setting-up the session it does not create circuits or channels. Instead, SIP proxies route requests or responses from server to server until destination is discovered. Moreover, because of its lightweight and http-like syntax, creation of new services is as easy as unlimited.

The Caller Preferences is a small, however very important extension of the Session Initiation Protocol. It brings main important advantage. Users can use single email-like address for all their network terminals. The calls are routed to the desired device or to the group of devices.
In this project Caller Preferences were successfully implemented in the server and client software. All modifications were made with respect to the Internet draft guidelines. What is more, testing showed that desired behaviour was achieved. Messages were directed to the right devices, which in this case were simulated by a few copies of the same UAC registered under contacts with different parameters.

Of course the work is not complete and there are some future tasks that should be kept in mind. First of all the author will keep track of changes in SIP internet drafts. Moreover, in end-user application setting Caller Preferences and registering should be done in more automatic matter or to some extent in the background. Finally, more advanced testing should take place in which server performance could be measured under high workload. Perhaps, setting up the testbed in bigger network could also be useful.

The author believes that Caller Preferences and single user’s address will be the very important factor in development of the future networks. However, the reality of ubiquitous network is still couple of years away...
Appendix A

Abbreviations:

3GPP - 3rd Generation Partnership Project
CPL – Call Processing Language
IETF – Internet Engineering Task Force
MGCP – Media Gateway Control Protocol
RSVP – ReSerVation Protocol
RTCP – Real-Time Control Protocol
RTP – Real Time Protocol
RTSP – Real-time Streaming Protocol
SAP – Session Announcement Protocol
SDP – Session Description Protocol
SER – SIP Express Router
SIP – Session Initiation Protocol
TRIP – Telephony Routing over IP
UAC – User Agent Client
UAS – User Agent Server
Appendix B

Request methods

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INVITE</td>
<td>Invites user to a call.</td>
</tr>
<tr>
<td>ACK</td>
<td>Acknowledges reception of the response. For instance after arrival of 200-OK response user sends back the ACK request.</td>
</tr>
<tr>
<td>BYE</td>
<td>Terminates a call.</td>
</tr>
<tr>
<td>CANCEL</td>
<td>Cancels the pending call i.e. INVITE message that has not been responded yet.</td>
</tr>
<tr>
<td>REGISTER</td>
<td>Registers one of user’s locations.</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Queries the capabilities of UA in the background (without actual ringing). This method can be directed to either client or server. UAS responses to OPTIONS request are the same as for any other request.</td>
</tr>
</tbody>
</table>

Response codes

<table>
<thead>
<tr>
<th>Status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>Informational</strong></td>
</tr>
<tr>
<td>100</td>
<td>Trying</td>
</tr>
<tr>
<td>180</td>
<td>Ringing</td>
</tr>
<tr>
<td>181</td>
<td>Call Is Being Forwarded</td>
</tr>
<tr>
<td>182</td>
<td>Queued</td>
</tr>
<tr>
<td>183</td>
<td>Session Progress</td>
</tr>
<tr>
<td></td>
<td><strong>Success</strong></td>
</tr>
<tr>
<td>200</td>
<td>OK</td>
</tr>
<tr>
<td></td>
<td><strong>Redirection</strong></td>
</tr>
<tr>
<td>300</td>
<td>Multiple Choices</td>
</tr>
<tr>
<td>301</td>
<td>Moved Permanently</td>
</tr>
<tr>
<td>302</td>
<td>Moved Temporarily</td>
</tr>
<tr>
<td>305</td>
<td>Use Proxy</td>
</tr>
<tr>
<td>380</td>
<td>Alternative Service</td>
</tr>
<tr>
<td></td>
<td><strong>Client-Error</strong></td>
</tr>
<tr>
<td>400</td>
<td>Bad Request</td>
</tr>
<tr>
<td>401</td>
<td>Unauthorized</td>
</tr>
<tr>
<td>402</td>
<td>Payment Required</td>
</tr>
<tr>
<td>403</td>
<td>Forbidden</td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
</tr>
<tr>
<td>404</td>
<td>Not Found</td>
</tr>
<tr>
<td>405</td>
<td>Method Not Allowed</td>
</tr>
<tr>
<td>406</td>
<td>Not Acceptable</td>
</tr>
<tr>
<td>407</td>
<td>Proxy Authentication Required</td>
</tr>
<tr>
<td>408</td>
<td>Request Timeout</td>
</tr>
<tr>
<td>410</td>
<td>Gone</td>
</tr>
<tr>
<td>413</td>
<td>Request Entity Too Large</td>
</tr>
<tr>
<td>414</td>
<td>Request-URI Too Large</td>
</tr>
<tr>
<td>415</td>
<td>Unsupported Media Type</td>
</tr>
<tr>
<td>416</td>
<td>Unsupported URI Scheme</td>
</tr>
<tr>
<td>420</td>
<td>Bad Extension</td>
</tr>
<tr>
<td>421</td>
<td>Extension Required</td>
</tr>
<tr>
<td>423</td>
<td>Interval Too Brief</td>
</tr>
<tr>
<td>480</td>
<td>Temporarily not available</td>
</tr>
<tr>
<td>481</td>
<td>Call Leg/Transaction Does Not Exist</td>
</tr>
<tr>
<td>482</td>
<td>Loop Detected</td>
</tr>
<tr>
<td>483</td>
<td>Too Many Hops</td>
</tr>
<tr>
<td>484</td>
<td>Address Incomplete</td>
</tr>
<tr>
<td>485</td>
<td>Ambiguous</td>
</tr>
<tr>
<td>486</td>
<td>Busy Here</td>
</tr>
<tr>
<td>487</td>
<td>Request Terminated</td>
</tr>
<tr>
<td>488</td>
<td>Not Acceptable Here</td>
</tr>
<tr>
<td>491</td>
<td>Request Pending</td>
</tr>
<tr>
<td>493</td>
<td>Undecipherable</td>
</tr>
<tr>
<td>500</td>
<td>Internal Server Error</td>
</tr>
<tr>
<td>501</td>
<td>Not Implemented</td>
</tr>
<tr>
<td>502</td>
<td>Bad Gateway</td>
</tr>
<tr>
<td>503</td>
<td>Service Unavailable</td>
</tr>
<tr>
<td>504</td>
<td>Server Time-out</td>
</tr>
<tr>
<td>505</td>
<td>SIP Version not supported</td>
</tr>
<tr>
<td>513</td>
<td>Message Too Large</td>
</tr>
<tr>
<td>600</td>
<td>Busy Everywhere</td>
</tr>
<tr>
<td>603</td>
<td>Decline</td>
</tr>
<tr>
<td>604</td>
<td>Does not exist anywhere</td>
</tr>
<tr>
<td>606</td>
<td>Not Acceptable</td>
</tr>
</tbody>
</table>

**Server-error**

**Global-Failure**
Appendix C

**Accept**
Specifies what content type UA is willing to accept i.e. “application/sdp”.

**Accept-Encoding**
Specifies the type of content coding i.e. “gzip”.

**Accept-Language**
Indicates preferred language for reason phrases, session descriptions and status responses i.e. “en-us”.

**Alert-Info**
Specifies an alternative ring tone for UAS or UAC i.e. „moo.wav”.

**Allow**
Gives the list of methods supported by UA that issued the message i.e. “INVITE,BYE”.

**Authentication-Info**
Key for mutual authentication.

**Authorization**
Authentication credentials i.e. Digest username="Alice", realm="atlanta.com", nonce="84a4cc6f3082121f32b42a2187831a9e", response="7587245234b3434cc3412213e5f113a5432".

**Call-ID**
Globally unique, identification sequence number of the particular invitation (call) or all registrations of the particular client.

**Call-Info**
Can carry additional information about caller or callee i.e. picture or homepage address.

**Contact**
Provides the URI (current location) of the user.

**Content-Disposition**
Indicates how the message body should be interpreted by UA i.e. „session” means that body describes the session and „render” means that the body content should be in some way displayed to the user.

**Content-Encoding**
Specifies the coding that was applied to the message body i.e. „tar”.

**Content-Language**
Language of the body i.e. „fr”.

**Content-Length**
Size of the message body in decimal number of octets.

**Content-Type**
Indicates the media type of the message body i.e. “application/sdp” or “text/html”.

57
Cseq
Includes single decimal sequence number and request method to order transactions within current dialog i.e. “355 INVITE”.

Date
Date and time when the request was first sent.

Error-Info
Additional information about why the error response (4xx, 5xx, 6xx) was issued.

Expires
Gives relative time (in seconds) after which message or content expires.

From
Indicates initiator of the request.

In-Reply-To
Enumerates Call-IDs to which this call references or returns.

Max-Forwards
The limit of proxies or gateways that message can go through.

Min-Expires
Conveys the minimum refresh interval supported for soft-state elements managed by the server.

MIME-Version
i.e. „1.0”.

Organization
Information about the organization to which element issuing the message belongs.

Priority
Urgency of the request i.e. “non-urgent”, “normal”, “urgent” and “emergency”.

Proxy-Authenticate
Contains authentication challenge (proxy-to-user authentication).

Proxy-Authorization
Identification information from the client to the proxy which require authorization.

Proxy-Require
Indicates features that must be supported by the proxy.

Record-Route
Inserted by the proxy to force future requests to be routed through the same server i.e. all proxies may be “recorded” to establish the same return path for the message.

Reply-To
Address, different from the “From” field to which callee may eventually reply.

Require
Features required from UAS by UAC i.e. “100rel”.

Retry-After
Number of seconds indicating the time for which service is going to be unavailable i.e. may be used in 4xx, 5xx or 6xx responses.
**Route**
Forcing routing through the listed set of proxies i.e. “<sip:bigbox3.site3.atlanta.com;lr>,<sip:server10.biloxi.com;lr>”.

**Server**
Information about software used by the server i.e. „SIP Express Router”.

**Subject**
Describes summary, topic or nature of the call.

**Supported**
Enumerates all extension supported by UAC or UAS.

**Timestamp**
Describes when UAC sent the request to UAS.

**To**
Recipient of the request.

**Unsupported**
Features not supported by UAS.

**User-Agent**
Like Server but gives information about UAC software i.e. „Kphone 3.14”.

**Via**
Indicates the path taken by the request so far.

**Warning**
Additional information about status response.

**WWW-Authenticate**
Authentication challenge (user-to-user authentication)
List of tables

Table 3-1 Example address of record ............................................................ 17
Table 3-2 Examples of Accept-Contact and Reject-Contact parameters ..... 21
Table 3-3 Results of the matching algorithm .............................................. 26
Table 4-1 SIP implementations ................................................................... 31
List of figures

Figure 2-1 SIP and accompanying protocols .................................................. 7
Figure 2-2 Basic SIP infrastructure and functional entities ......................... 10
Figure 2-3 Example request and response message ........................................ 13
Figure 2-4 Back to back connection set up ..................................................... 14
Figure 2-5 Trapezoid scenario for connection set up ..................................... 14
Figure 2-6 Registration ................................................................................. 15
Figure 3-1 Matching algorithm ...................................................................... 25
Figure 3-1 Example of SER configuration ..................................................... 33
Figure 3-2 Screenshot of Kphone ................................................................ 34
Figure 5-1 Dialog box for caller preferences configuration ............................ 36
Figure 5-2 Additional field for registering contact with parameters ............ 37
Figure 5-3 Example message with new header fields .................................... 37
Figure 5-4 Example registration message ....................................................... 38
Figure 6-1 “Comprehensive example” - registration of client 1 .................... 43
Figure 6-2 “Comprehensive example” - registration of client 2 .................... 43
Figure 6-3 “Comprehensive example” - registration of client 3 .................... 44
Figure 6-4 “Comprehensive example” - registration of client 4 .................... 44
Figure 6-5 “Comprehensive example” - registration of client 5 .................... 44
Figure 6-6 “Comprehensive example” - request message ......................... 45

Figure 6-7 “Comprehensive example” - server output .............................. 45

Figure 6-8 “Forcing audio/video” example – registration of contact 1 ....... 46

Figure 6-9 “Forcing audio/video” example – registration of contact 2 ....... 46

Figure 6-10 “Forcing audio/video” example – request message 1 ............ 47

Figure 6-11 “Forcing audio/video” example – server output 1 ............... 47

Figure 6-12 “Forcing audio/video” example – request message 2 ............ 48

Figure 6-13 “Forcing audio/video” example – server output 2 ............... 48

Figure 6-14 “Languages” example – request message ............................ 49

Figure 6-15 “Languages” example – server output ................................. 50

Figure 6-16 “I hate people!” example – request message ........................ 50

Figure 6-17 “I hate people!” example – desktop screenshot ....................... 51
References


