Service Policy Management for User User-Centric Services in Heterogeneous Mobile Networks

KONSTANTINOS AVGEROPOULOS

KTH Microelectronics
and Information Technology

Master of Science Thesis
Stockholm, Sweden 2004

IMIT/LCN 2004-04

KTH Microelectronics
and Information Technology

Master of Science Thesis

Service Policy Management for User-Centric Services in Heterogeneous Mobile Networks

Konstantinos Avgeropoulos

March 31, 2004

Supervisor: **Professor Gerald Q. Maguire Jr.**
Advisors: **Dr Theo Kanter, Andreas Wennlund M. Sc.**
Sponsor: **R2Meteon**

Abstract

The Session Initiation Protocol (SIP) is a signaling protocol for IP-based media services that will be the de facto standard for future media-over-IP services. Since SIP User Agents (UAs) support a limited number of service types (usually one or two), we assume that the future user will need to operate several UAs simultaneously. These UAs will constitute the user's personal service network. In this thesis, we investigate architectures for policy-based management of this network so that it can be used in an efficient manner. To achieve this, we propose a new SIP entity, called the SIP Service Manager (SSM), which lies in the core of the management system. Finally, we evaluate our proposal by implementing one version of the SIP Service Manager.

Sammanfattning

Session Initiation Protocol (SIP) är ett signaleringsprotokoll för IP-baserade mediatjänster som kommer att bli "de facto"-standard för framtida "media-över-IP"-tjänster. Då SIP User Agents (UAs) stöder ett begränsat antal typer av tjänster (oftast en eller två), antar vi att framtida användare kommer att behöva använda sig av flera UAs samtidigt. Dessa UAs utgör en användares personliga tjänstenätverk. I denna rapport, undersöker vi arkitekturer för policybaserad hantering (management) av detta nätverk, så att det kan användas effektivt. För att uppnå detta, föreslår vi en ny SIP-enhet, kallad SIP Service Manager (SSM), som är placerad i kärnan av hanteringssystemet. Slutligen utvärderar vi vår föreslagna lösning genom att implementera en version av en SIP Service Manager.

Acknowledgments

First of all, I would like to thank my supervisor Professor Gerald Q. Maguire Jr. and the Adaptive and Context Aware Services (ACAS) project for allowing me to work on this thesis project. Furthermore, I would like to thank Professor Maguire for his immediate help whenever I needed it, and also for sharing his insight and experience with me, making this thesis a learning experience.

A great thanks to my advisers Dr. Theo Kanter and Andreas Wennlund M. Sc. for their guidance throughout the project. Andreas Wennlund was always by my side helping me to avoid deadlocks, especially during the implementation phase of the project.

This project owes a great deal to my sponsor R2Meteon and especially Ulf Börjeson. I would like to thank him for his trust, his patience and his support at times when it was needed. I hope that this thesis is a contribution to R2Meteon's research.

I would also like to thank Intel their generous grants to Wireless@KTH and more specifically for the donation of Dell Precision 450 workstations (used as infrastructure computers and for testing the implementation), and the Dell True Mobile 1170 Access Points (used for testing SIP across WLANs). In addition, I would like to thank Hewlett-Packard's Applied Mobile Technology Solutions in Learning Environments - Grant 2003, for the donation of the iPAQ 5550 (used with the X-lite User Agent and the Dell Access points as monitored services).

All software used to perform this project (even for writing this report) is free and most of it open source. Therefore, I would like to express my gratitude to the various open source and free software organizations and communities.

Finally, I would like to thank my fellow students Gustavo Zago Basilio, Philippe Villeneuve and Diego Urdiales Delgado for their help in practical matters related to Java programming.

This work is entirely devoted to my secret project mates: my family.

Table of Contents

1. INTRODUCTION

1.1 IP and mobile communications

After more than a decade of rapid growth, the Information Technology and Telecommunications world is facing a major turning point. During these rather exciting years for both industry and academic community, a great number of new technologies in the area emerged. Some of them were quickly abandoned while others were met with global acceptance. The computer networking world finally converged to the Internet Protocol (IP), specified by the Internet Engineering Task Force (IETF), which became the main computer networking protocol and formed the basis of the global computer network known as the Internet. Due to its simplicity, flexibility, and perhaps the open standards strategy followed by the IETF, IP and its related protocols swiftly conquered the globe.

In the wide area of Wireless and Mobile Communications, a limited set of technologies prevailed, with the Global System for Mobile communications (GSM[1]), standardized by the European Telecommunications Standards Institute (ETSI), as the most widely accepted standard for wireless and mobile personal communication. Like the traditional Public Switched Telephony Network (PSTN), GSM's main function was to provide mobile telephony service to its users. However, the need for richer services such as data communication was soon realized. In addition, the Internet's impressive expansion and usefulness made it attractive for mobile users. In response to these new demands, new extensions to the original GSM were introduced such as the Short Message Service (SMS) that allows for short message transmission, and the Wireless Access Protocol (WAP), which allows for a type of Internet access. However, it was quickly realized that these extensions were limited: SMS provides for very short, text-only messages and WAP's Internet access did not prove very successful. To cope with these limitations, a new extension to GSM was introduced, known as the General Packet Radio Service (GPRS) that can handle GSM inter-networking with other packet switched networks, such as IP networks. The connection between the two dominant technologies in contemporary communications was made possible and IP officially entered the cellular communications world.

Meanwhile, other shorter-range wireless technologies were introduced such as Bluetooth, originally developed by Ericsson Mobile Communications, and the 802.11 Wireless Local Area Network (WLAN) standards series by the Institute of Electrical and Electronics Engineers (IEEE). These technologies provide higher bit rate links than cellular systems did at the time they were developed and could accommodate IP as their networking protocol. Finally, 3rd generation mobile communications systems, such as the Universal Mobile Telecommunications System (UMTS) were assigned to support IP.

These various developments make IP's dominance apparent. It appears that global communications is gradually becoming unified and IP is their common feature. Though the actual unification of contemporary communications under IP is far from complete, it is expected

1. The acronym GSM originally stood for Groupe Speciale Mobile, a name established by the Conference for European Posts and Telecommunications (CEPT) committee that started the GSM standardization.

to happen in the near future, resulting to a vast, largely heterogeneous communications network that is interconnected using IP.

IP's capabilities, however, are not infinite. In fact, IP was originally designed for fixed data networks and when used in mobile networks it quickly shows a number of weaknesses, most of which have to do with IP's location dependence. To cope with these weaknesses, certain extensions have been proposed for IP such as Mobile IP [1] and Session Initiation Protocol (SIP) [2][1]. The latter technology is the main subject of this thesis and will be discussed further in the following chapters.

1.2 Modern services in a heterogeneous environment

As technology grows, so does the richness and complexity of new services. For the Internet, increased connection speeds offered to home users have turned the global network from a means for simple information exchange (e.g. web browsing and e-mail) into a major source for multimedia services and entertainment (messaging, telephony, radio, television, online gaming, etc.). On the mobile communications side, devices constantly grow in complexity and capabilities. The introduction of appropriate, small footprint operating systems as well as various peripherals (cameras, sensors, etc.) aim to turn mobile devices from simple phones into mobile computers, dramatically increasing their usage and therefore increasing the service options for their users.

In this thesis we claim that SIP, due to its simplicity and extensibility, can be at the heart of this extremely diverse service network. Its quick adoption by certain parts of industry reinforces, if not verifies, this claim. A future user will need to maintain a number of services simultaneously, such as: a personal Internet radio (or playlist), an online game, an instant messenger, and so on, in addition to one or more phones, a fax, etc. Today SIP client applications (also called SIP User Agents) often have one or two services attached (e.g. a client with a SIP phone and a videophone). This means that in order to cover all these new service demands (described above), a user will have to operate various SIP clients at the same time. This thesis studies possible means for efficient management of the services that these clients provide.

In the next chapter, we provide an introduction to the research area and present SIP, as well as the IP Multimedia Subsystem (IMS), which is a SIP facility designed for the ETSI-specified GSM/EDGE Radio Access Networks (GERANs) and UMTS Terrestrial Radio Access Networks (UTRANs). In chapter 3 we discuss management schemes for generic SIP networks and IMS. In chapter 4, we propose a new service management scheme based on a new SIP entity that we call the SIP Service Manager. Finally, in chapter 5, we present our implementation of the SIP Service Manager, then evaluate it and draw conclusions about its operation.

1. Note that Mobile IP and SIP solve entirely different problems. To summarize, Mobile IP is a routing scheme which operates at the Network Layer, whereas SIP is an Application Layer protocol.

2. BACKGROUND

The Session Initiation Protocol (SIP) is the focus of this thesis. In section 2.1 we present the basics of the protocol, its messages, functions, and a few notes that clarify some of its operations. Following this, we give a brief description of the IP Multimedia Subsystem (IMS), an architecture proposed by the 3rd Generation Partnership Project (3GPP) in order to provide SIP facilities in 3rd Generation mobile networks.

2.1 Session Initiation Protocol (SIP)

Before going into the details of the protocol, we will explain the basic function of SIP with a short example. Suppose that Alice wishes to make a call to Bob over an IP network. Before Alice can speak to Bob she must first learn the location of Bob's phone or, in IP terms, know its IP address. This can be easy if Bob's phone has a static IP address that is known to Alice. However, in mobile networks this is rarely the case. One way to solve this problem would be by using the Domain Name System (DNS). In this case, Alice would know a Fully Qualified Domain Name (FQDN) that would point to the IP address of Bob's device. If Bob's device changes its IP address, he could inform his Domain Name Server by issuing DNS updates (for example via DNSsec). SIP offers its own way of locating users. When SIP is used, Alice would know Bob's SIP Uniform Resource Identifier (URI) rather than a FQDN. A simple SIP URI is a string of the form: *<name>@<domain name>*, similar to an e-mail address (see paragraph 2.1.1). An example URI might be: bob@sip.kth.se. URIs in SIP are used in a way similar to the way e-mail uses e-mail addresses. As we will see later, SIP may in turn use the DNS in order to locate users.

As soon as Alice knows the IP address of Bob's device, there must be some signaling (call notifications, ringing tones, busy tones, etc.) that will allow the two endpoints to establish a call. SIP's main function is to provide this signaling. Moreover, the two devices need to negotiate certain aspects of their communication (such as the codec used, bit rates, and so on), before starting a call. While SIP does not handle this negotiation by itself, it provides transport for other protocols, mainly the Session Description Protocol (SDP), that do perform this task.

The core of the Session Initiation Protocol is specified in the IETF Request For Comments (RFC) 3261 [2], which currently has the status of a Proposed Standard. SIP is an application layer protocol. Its function is to provide session signaling for IP (versions 4 and 6) networks. As explained previously, SIP does not provide any particular service. Instead, its purpose is to provide for **service location**, **session establishment,** and **session termination** between end nodes. A typical example of a service that utilizes SIP for its session establishment is IP telephony. In the following paragraphs, we will go through the entities of a SIP architecture and the protocol's basic operations. Further information about the protocol can be found in RFC 3261 [2] and relevant RFCs.

2.1.1 Uniform Resource Identifier (URI)

An important component of SIP is the SIP Uniform Resource Identifier[1] (URI). A SIP URI is a string that identifies a communications resource in a network. It contains sufficient information to initiate and maintain a communication session with the resource. The full format of a SIP URI as specified in [2] is:

```
<scheme>:<user>:<password>@<host>:<port>;<uri-parameters>?<headers>
```

Where the contents in brackets (<>) have an arbitrary string value. For SIP URIs the *scheme* component can have two values which are 'sip' and 'sips'. A 'sips' scheme indicates that the URI is a SIPS URI. A SIPS URI indicates that the resource the URI maps to must be accessed in a secure manner, using Transport Layer Security (TLS). This essentially means that a User Agent Client (see paragraph 2.1.1) accessing a resource using the SIPS URI must establish a TLS connection with the domain that owns the SIPS URI.

The names of the various parts of the URI can be misleading. The *user* component does not necessarily identify a person, but rather a certain resource on the specified *host*. Furthermore, the *host* component does not always refer to an actual host since it often refers to an entire domain. The *port* component indicates the UDP and TCP port on which the resource can be accessed. In addition, a SIP URI may contain a number of extra parameters and headers that provide more information.

A URI is interpreted according to its context (i. e. the place where it appears). Its context also determines whether a component is optional, mandatory or if it is meaningful or not (and therefore if it is allowed or not). In other words, a URI does not mean anything by itself. Its interpretation (according to its context) is what makes it valuable.

An important type of SIP URI is the **Address-of-Record** (AOR). The host part of the AOR points to a domain with a location service that can map the AOR to another URI that indicates the location where a user might be available. An AOR is frequently thought of as a user's “public” address and it is often part of the contract made between a customer and a SIP service provider. Callers will typically use a user's AOR to contact him. An example AOR is bob@company.com (user *bob* at domain *company.com*).

Note that SIP is not the only protocol that defines URIs. Other types of URIs are defined and used by other protocols. A generic syntax for URIs is defined in [3].

2.1.2 SIP entities

A SIP entity is a node in a network that is able to understand SIP. It can act as a **S**IP server, as a **S**IP client, or both. A **SIP server** is a SIP entity that receives and processes requests and sends

1. The acronym URI has three explanations: Uniform Resource Identifier, Uniform Resource Indicator, and Universal Resource Identifier. In the context of SIP the first two are mainly used. In this document we will use only the first one to avoid confusion.

back replies. A **SIP client** sends requests and receives and processes replies[1]. There are four basic types of SIP entities.

User Agent: A SIP User Agent (UA) typically resides on the user's device. It acts both as a SIP server and a SIP client. Its server component is called a User Agent Server (UAS) and its client component is called a User Agent Client (UAC). A SIP UA handles all SIP transactions on behalf of the user. It is able to initiate calls, accept incoming SIP messages, process signaling messages, and so on. SIP phones and softphones include User Agents.

Proxy: A SIP proxy is a SIP entity that acts both as a server and a client. As a server, it accepts SIP messages and sends appropriate replies. As a client, it makes requests on behalf of other clients rather than generating its own requests like a normal UAC. The SIP proxy's main function is similar to routing. Its purpose is to receive a message and send it to another SIP entity that is presumed to be "closer" to the message's destination. A SIP proxy can be stateful, which means that it may keep track of SIP transactions (see also paragraph 2.1.8) or stateless. A SIP proxy is also where policies are enforced such as if the calling user is allowed to make this specific call. Finally a SIP proxy may modify messages before forwarding them.

Redirect Server: A SIP Redirect Server is a server that accepts requests and sends redirection responses that contain information on an alternative set of URIs where the destination can be reached. After receiving a redirection response, a client typically resends the request to the destination indicated by the Redirect Server.

Registrar: A SIP Registrar is a special server that helps locate users or resources. It accepts registration requests from clients and stores the information it receives in a **location service**. The location service is an abstract service and not a SIP entity. In practice, the location service can be a database that stores information which can then be queried by SIP Proxies and Redirect Servers in order to locate users. Entries in this database map a URI to a set of other URIs that indicate how the user or resource can be accessed (location, methods, etc.).

2.1.3 SIP requests

A **request** is a message sent by a SIP client to a SIP server that typically invokes a function on the server that will eventually produce a response. The content of a SIP request header is written in human readable form, similar to HyperText Transfer Protocol (HTTP) headers. The first line of such a header is called the **Request Line**. The Request-Line has the following format:

```
<Method><SP><Request-URI><SP><SIP version><SS><CRLF>
```

Where SP stands for a single space and CRLF for carriage return line feed. Below are two examples:

1. SIP is not a traditional client-server protocol, such as HTTP, meaning that SIP entities often act as both servers and clients. Understanding the notions of *server* and *client* in SIP is fundamental to understanding the protocol itself.

```
REGISTER sip:sip.wireless.kth.se SIP/2.0
INVITE sip:sipphone@sip.wireless.kth.se SIP/2.0
```

The values of the *Method* component are explained below. The *Request URI* can be a SIP or SIPS URI discussed in paragraph 2.1.1 or a generic URI as specified in [3], depending on the method. Following the Request-Line is a list of headers that contain further information. These headers are discussed in paragraph 2.1.5.

As the Request-Line format above indicates, each request must contain a request method. A **request method** is the primary function that a request is meant to invoke on a SIP server. The method appears as a simple string in the SIP request message itself. At the time of this writing there have been 13 methods specified by the IETF. These are listed in table along with references to their specifications. Here we will give a short description of these methods.

INVITE: INVITE requests are sent by a UAC in order to begin session establishment. They are the first messages to be sent in a typical SIP transaction. The UAC sending the request is called the **inviter** whereas the final destination of the request is called the **invitee**. An INVITE request typically contains information about the type of the session the UAC intends to initiate as payload. The actual session description is performed by other protocols, mainly the Session Description Protocol (SDP) (see also subsection 2.1.6). INVITE requests can also be sent during an established session to change session parameters without tearing it down. These messages are known as **re-INVITE** messages.

ACK: ACK requests are send from the inviter to the invitee in order to acknowledge the reception of a response to a previous INVITE request. This means that an ACK message may only be sent after an INVITE request.

CANCEL: CANCEL requests are sent by a client to cancel the processing of a previously made request. A CANCEL request has no effect if the client has already received a response to its previously made request. This means that a CANCEL is meaningful only if the previously made request requires a “long” time to produce a response, such as an INVITE request.

BYE: A BYE request is sent whenever a UAC wishes to terminate an established session.

OPTIONS: An OPTIONS request is made whenever a UAC wishes to discover the capabilities of a SIP entity, without initiating a session (using an INVITE request). This method is especially useful whenever a caller wishes to use a specific type of communication that might not be supported by the callee. The caller can query for capabilities without “ringing” the callee. A final (i.e. non-provisional) response to an OPTIONS request must be the same as if the request was an INVITE (see [2] section 11.2.)

Method	*Description*	*Specified in*
INVITE	Session establishment	RFC 3261 [2]
ACK	Acknowledgment of final response to INVITE	RFC 3261 [2]
CANCEL	Cancel a previously sent request that initiated a SIP transaction	RFC 3261 [2]
BYE	Terminate a session	RFC 3261 [2]
OPTIONS	Query a User Agent Server about its capabilities	RFC 3261 [2]
REGISTER	Register Contact information	RFC 3261 [2]
REFER	Provide the recipient with third party contact information	RFC 3515 [4]
MESSAGE	Send an instant message attached to the request	RFC 3428 [5]
UPDATE	Update session parameters	RFC 3311 [6]
INFO	Transport session related control information during a session	RFC 2976 [7]
PRACK	Provisional response acknowledgment	RFC 3262 [8]
SUBSCRIBE	Request notification upon a certain event	RFC 3265 [9]
NOTIFY	Transport of an event notification towards a subscriber	RFC 3265 [9]

Table 2.1 - SIP request methods

REGISTER: A REGISTER request is sent by a client to a Registrar Server to update information in the location service. The request contains contact information that is bound to the sender's URI in the location service. The contact information is contained as a set of URIs in the *Contact* header of the message.

REFER: A REFER request is sent whenever one of the two parties involved in a session wishes to provide its peer with third party contact information. This information is a set of URIs included in the *Refer-To* header of the REFER message.

MESSAGE: MESSAGE requests are used for Instant Messaging purposes. A MESSAGE request is sent whenever a User Agent wishes to send a near real-time message to other User Agents.

UPDATE: An UPDATE request is sent whenever a client wishes to change a session's parameters. Its differs from a re-INVITE request in that an UPDATE does not require that the session has already been established. Thus, an UPDATE message can be used to change a session's parameters *before* a call is answered.

INFO: An INFO request is used to send session control information during an established session. The information carried in this request is often related to other signaling protocols such as the Integrated Services Digital Network (ISDN) User Part (ISUP) of Signaling System No. 7 (SS7). INFO requests do not necessarily affect the state of the SIP session. They just provide additional control information.

PRACK: A PRACK request is sent to acknowledge the reception of a provisional response. A provisional response provides information on the processing of a previously made request.

SUBSCRIBE: This request is sent when a client wishes to be notified upon a certain event. Events are defined by **event packages**, which are specifications that describe a certain set of events. The sender of a SUBSCRIBE request is called the **subscriber**.

NOTIFY: A NOTIFY request is an event notification typically sent to a subscriber. The sender of the NOTIFY message is called the **notifier**. It is presumed that the event notification follows a subscription previously made for that event.

All requests in SIP must be followed by at least one response. An exception to this rule is the ACK request, which does not require a response. If a response is not sent to a request, a SIP entity usually retries a certain number of times until it stops sending (which triggers a timeout). A UAS may not support processing of all requests. If a UAC makes a request that is not supported or allowed the UAS may reply with a list of allowed methods, included in the *Allow* header. In addition, a UAC can find out the methods supported by a UAS by making an OPTIONS request.

2.1.4 SIP responses

A **response** is a message sent by a server to a client in response to a request. As in SIP requests, SIP response headers are written in human readable form. The first line of a SIP response header is called a **Status-Line** and has the following format:

```
<SIP version><SP><Status-Code><SP><Reason-Phrase><CRLF>
```

Where <SP> stands for a single space and <CRLF> for carriage return and/or line feed. Below are three examples:

```
SIP/2.0 180 Ringing
SIP/2.0 200 OK
SIP/2.0 404 Not Found
```

The *Status Code* is a three digit decimal integer result code that indicates the outcome of a request processing. The *Reason Phrase* is a textual description of the *Status Code*. SIP defines six categories of responses identified by the first digit of the *Status Code*. The two last digits indicate the actual response. The response categories are listed in table 2.2.

Status-Code	***Response category***	***Description***
1xx	Provisional	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 by the requester to complete the request

Status-Code	*Response category*	*Description*
4xx	Client Error	The request contains bad syntax and 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 2.2 - SIP response Status Codes

2.1.5 SIP message headers

SIP message headers are a component of both SIP requests and responses. They follow the *Request Line* in SIP requests and the *Status Line* in SIP responses. A header consists of a set of Header Fields. A **Header Field** is a line in the SIP message header that specifies a value or a number of values of a specific attribute. A Header Field has the following format:

```
<field-name><SP>*:<SP>*<field-value>(<SP>*,<SP>*<field-value>)*
```

Where (format)* stands for zero or more repetitions of the format. Below are two examples:

```
Call-ID: 1120291530@125.10.45.120
Route: <sip:maria@chalmers.se>, <sip:george@kth.se>,
<sip:michael@it.kth.se>
```

Note that header field values can occupy more than one row in the message. Some header fields that require parameters add the parameter name and value after a semicolon. Their format is shown below:

```
<field-name><SP>*:<SP>*<field-value><SP>*(;<par-name>=<par-value>)
```

For example:

```
From: “Bob mobile” <sip:0701234567@mobprovider.com>;tag=123B234
```

Appendix B lists the currently specified header fields. As is shown by the descriptions, not all fields need be present in all messages. Some of them are meaningful only in certain cases.

2.1.6 SIP and SDP

As mentioned above, SIP does not handle session specific negotiation between SIP endpoints. Instead, this is done by the Session Description Protocol (SDP). The core of SDP is specified in IETF RFC 2327 [10], which currently has the status of *Proposed Standard*.

Currently, SDP is more a format rather than a communication protocol in the sense that it uses other protocols for its transport. SDP messages must include sufficient information about a media session and how it can be accessed. Like SIP, its messages are written in human readable text. An SDP message consists of a number of lines of the following form:

```
<type>=<value>
```

Where *type* is a single, case sensitive character and *value* is a string whose format depends on *type*. A message consists of three parts: the **Session description,** the **Time description**, and the **Media description**. Below we show the structure of an SDP message. Types marked with '*' are optional. Text after '//' is just comments and is **not** part of the SDP format.

//Session description

```
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 below)

```
z=* (time zone adjustments)
k=* (encryption key)
a=* (zero or more session attribute lines)
```

//Zero or more media descriptions (see below)

//Time description

```
t=  (time the session is active)
r=* (zero or more repeat times)
```

//Media description

```
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)
```

SDP was initially meant to be used with protocols other than SIP, such as the Session Announcement Protocol (SAP), related more to conference sessions and IP multicast. Therefore, some of the types shown above are meaningless for SIP and are not used.

Session negotiation using SDP is made using an offer/response model. The initiator of the session offers a number of media types which they support for the session and receives a response with the subset of this list representing the supported media of the other endpoint(s). We will show this procedure in the next paragraph with an example.

Although it is perhaps the only means for session description/negotiation widely used by SIP, SDP has shown a few weaknesses related to its 'clumsy' extensibility and lack of expressiveness. For these reasons, work is in progress within the IETF for a new version of the protocol.

2.1.7 SIP example

To summarize the above, we will give a simple example of SIP's operation. In the following example, Alice is a subscriber to a SIP provider and she is using a SIP phone to call Bob, who is using SIP facilities provided by his company and is running a SIP softphone on a workstation.

2.1.7.1 Registration

Before being able to receive any calls, Bob must register with the SIP registrar of his domain. The registration begins with a REGISTER request sent by Bob's UAC (included in the softphone) to the registrar. The REGISTER request can be sent to the registrar either directly or via a SIP proxy. The location (i. e. IP address or domain name) of the registrar or the SIP proxy is made known to Bob in several ways. Below we list four of them.

a) The *host* part of the Address-of-Record (AOR)

In this case the *host* part of Bob's AOR is the DNS host name of his SIP registrar or proxy. Therefore Bob's UAC will resolve the *host* part of the AOR normally via DNS and in this way learn the IP address of his SIP registrar or proxy.

b) Manual configuration

Bob might have been informed of the registrar's IP address or domain name through other means. In that case he can configure his UAC's “Outbound SIP proxy” with the location of the registrar or the SIP proxy. This applies in cases where the *host* part of the AOR is not the same as the DNS host name of the registrar or proxy. For example an AOR can be: bob@comany.com, whereas the host name of the proxy could be sip32.company.com.

c) Dynamic Host Configuration Protocol (DHCP)

The location of the registrar is made known to Bob's computer via a special option of DHCP (see [11] and [12]), in the same way as the Domain Name Servers or the default IP gateway.

d) Domain Name System SRV Resource Record

Bob could make a DNS query using the SRV Resource Record [13] to find available registrars or SIP proxies for his domain.

In this example we assume that Bob's softphone is manually configured and that its UAC accesses the registrar directly (without using a proxy). The registration process is shown in Figure 2.1. The number in front of each message is a sequence number for easy reference and **not** part of the message itself. Note that the location server (location.company.com) is **not** a SIP entity and therefore messages to and from the location server are **not** SIP messages. The sequence numbers of these messages are marked with '*'.

1. REGISTER

This message contains the following information:

```
REGISTER sip:company.com SIP/2.0
Via: SIP/2.0/UDP 192.2.0.100;rport
CSeq: 7808 REGISTER
To: “Bob” <sip:bob@company.com>
Expires: 900
From: “Bob” <sip:bob@company.com>
Call-ID: 762357098@192.2.0.100
Content-Length: 0
Event: registration
Contact: “Bob” <sip:bob@192.2.0.100;transport=udp;methods=”INVITE,
MESSAGE, INFO, SUBSCRIBE, OPTIONS, BYE, CANCEL,NOTIFY, ACK”>
```

For the specifics of each header field of this message refer to Appendix B and the relevant RFCs.

With this REGISTER request Bob informs the registrar (see the *Contact* field) of his device's IP address, the transport protocol his UA uses for SIP (in this example it is UDP), the duration for which this information is valid (in this example it is 900 s), and the request methods his UAS can process. In other words, the REGISTER message contains adequate information such that Bob's UA can be contacted via SIP.

2. 100 Trying

After the registrar sends the request to the location service, it sends a *100 Trying* response to Bob's UAC to inform it that its request was received and processed properly, but that the invoked function has not finished yet. This message looks like the following:

```
SIP/2.0 100 Trying
Via: SIP/2.0/UDP 192.2.0.100:5060;rport
To: “Bob” <sip:bob@company.com>
From: “Bob” <sip:bob@company.com>
Call-ID: 762357098@192.2.0.100
CSeq: 7808 REGISTER
Content-Length: 0
```

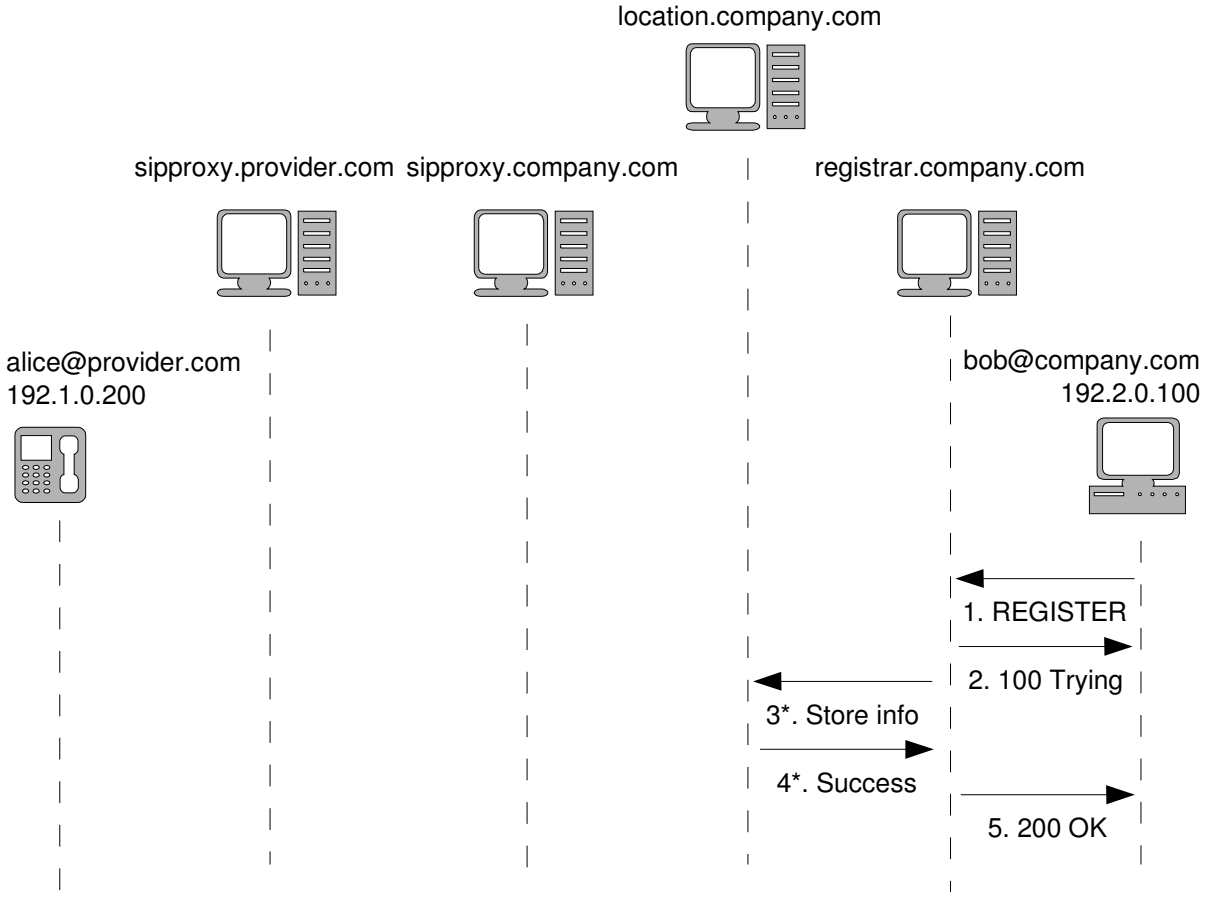

Figure 2.1 - SIP Example Registration

3. Storing registration information in the location server

As mentioned, messages to the location server are not SIP messages. The location server can in practice be a database server that stores contact information about SIP users. An example entry for Bob could be the following:

```
User: Bob
AOR: bob@company.com
IP address: 192.2.0.100
SIP Port number: 5060
Transport: UDP
```

```
Methods: INVITE, MESSAGE, INFO, SUBSCRIBE, OPTIONS, BYE,
CANCEL,NOTIFY, ACK
Expires: 900
```

Where the AOR here equals the AOR passed in the REGISTER request using the *To* header field.

4. Registration information successfully stored notification

The location service informs the registrar that the information sent in message 3 was successfully stored.

5. 200 OK

Via this message the registrar informs Bob's UAC that its request was successfully fulfilled. The message looks like the following:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.2.0.100:5060;rport
To: “Bob” <sip:bob@company.com>
From: “Bob” <sip:bob@comapny.com>
Call-ID: 762357098@192.2.0.100
CSeq: 7808 REGISTER
Expires: 900
Contact: <sip:bob@192.2.0.100;transport=udp>
Content-Length: 0
```

After a successful registration, Bob is accessible via SIP (more precisely via his UAS). In order to prevent the registration from growing stale (note the *Expires* field), Bob's UAC will repeat the registration process at regular intervals. Note that registration is required by SIP **only** when a user wishes to accept incoming calls. For outgoing calls registration is not required by SIP (although administrators and operators might require it).

2.1.7.2 Session initiation and termination

A call setup begins with the caller's (Alice's) UAC sending an INVITE request towards the caller (Bob) via a SIP proxy (sipproxy.provider.com). The location of this SIP proxy is known to Alice by one of the ways described in paragraph 2.1.7.1. The call setup procedure is shown in Figure 2.2.

1. INVITE

The INVITE message begins the session initiation process. In addition this message is often used to carry an SDP payload for the session negotiation. The message looks like the following:

```
INVITE sip:bob@company.com SIP/2.0
```

```
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
To: <sip:bob@comany.com>
Content-Type: application/sdp
From: “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice” <sip:alice@192.1.0.200;transport=udp>
```

As indicated in the SIP message (*Content-Type* and *Content-Length*) the packet also contains an SDP payload. This looks like the following:

```
v= (Anything - Not used in SIP)
o= (Anything - Not used in SIP)
s= (Anything - Not used in SIP)
c= IN IP4 192.1.0.200
t= (Anything - Not used in SIP)
m= audio 32772 RTP/AVP 0 97 3
a= rtpmap:0 PCMU/8000
```

As shown, the SDP payload contains adequate information for the session such as IP address, port number (32772), protocol (RTP/AVP), codec (PCM μ-law at 8000 samples/s), etc. With this information, Alice's device makes an 'offer' to Bob to use certain media. As we will see, Bob will reply to this offer, accepting it, with the 200 OK response.

2. 100 Trying

This is similar to the 100 Trying message of the previous paragraph, informing Alice that her INVITE request was received successfully and 'understood'; meaning that there were no errors in the message.

3. INVITE

At this point, Alice's SIP proxy must forward her request to the right SIP entity. Locating Bob's proxy server can be done by any of the ways mentioned in the previous paragraph other than the DHCP solution of course. In this example, we assume that the proxy is located via a DNS lookup using the SRV Resource Record (RR) (the DNS messages are not shown in figure Figure 2.2).

This INVITE request will be similar to the 1. INVITE message shown previously, although some header fields will be modified at the proxy's option. Below we give an example message.

```
INVITE sip:bob@company.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.1:5060;branch=8943170feab3209fcd8374.2
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
```

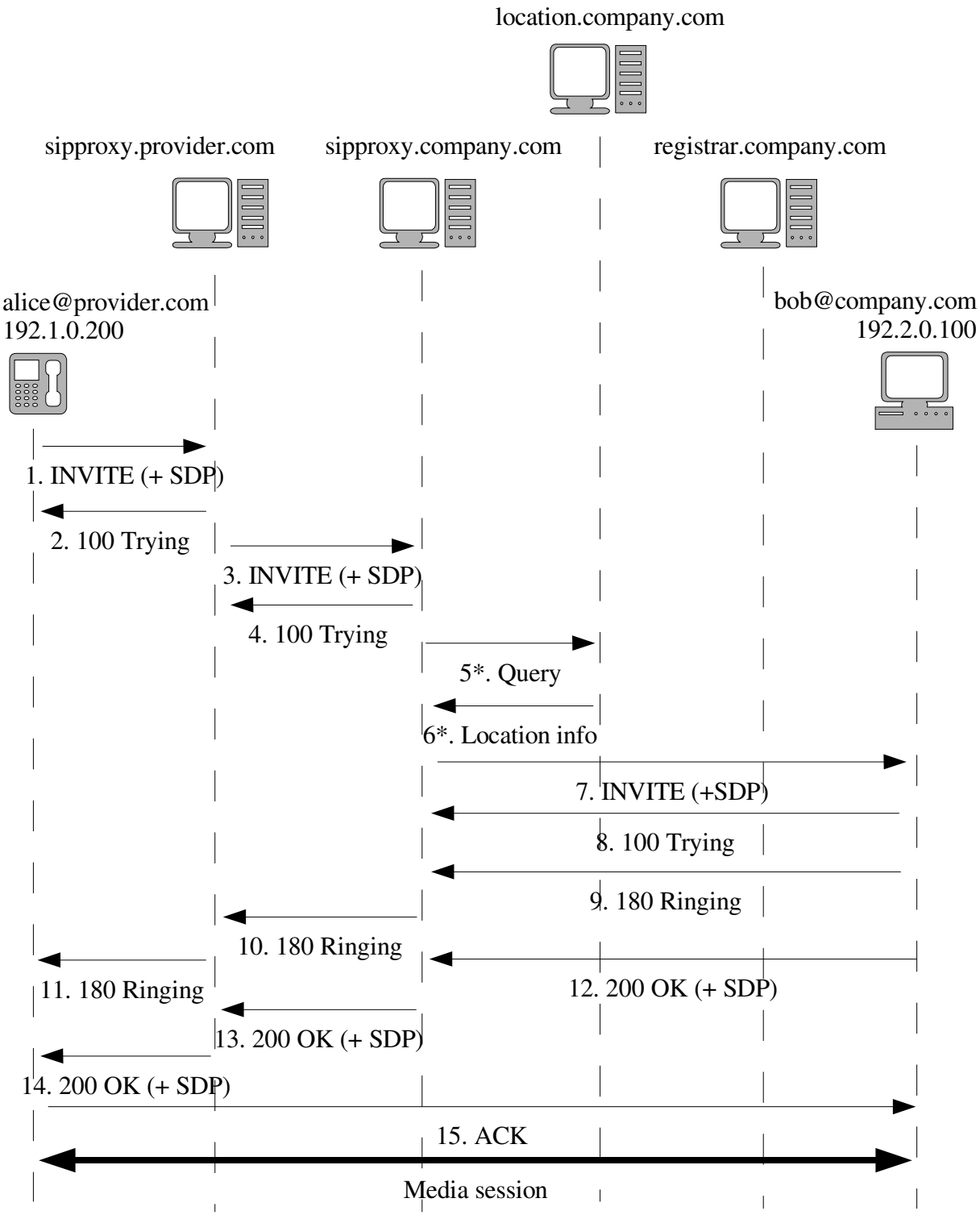

Figure 2.2 - SIP example call initiation

```
Max-forwards: 69
To: <sip:bob@comany.com>
Content-Type: application/sdp
```

```
From: “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice” <sip:alice@provider.com;transport=udp>
```

As we see, the proxy added two lines to the request: one *Via* line with its *own* information and one *Max-Forwards* line to avoid routing loops of the call. The SDP payload is left intact.

4. 100 Trying

Again this is similar to the 100 Trying message of the previous paragraph, with some information added to the message (such as a *Via* header field line). It informs Alice's proxy that its request (3. INVITE) was received successfully.

5. Query location information from the location service.

Bob's proxy queries for Bob's location from the location service. This is not a SIP message.

6. Service location service reply

The location server replies to the proxy's query. Again, note that this is not a SIP message.

7. INVITE

Since Bob had successfully registered earlier, his contact information will be found by the location service. Otherwise, the location server would reply that Bob is not currently connected (even though he has an account) and Bob's proxy would respond to Alice's proxy with a 480 Temporarily Unavailable response. If Bob had no account at all, the location service would not recognize his name and the proxy would reply with a 404 Not Found response. Here we assume that everything was successful, thus Bob's proxy will send the INVITE message to his UAS. Again, the message is similar to the previous INVITE requests, with some parts of the header modified.

8. 100 Trying

Bob's UAS informs his proxy that the INVITE request was received successfully and 'understood'.

9. 180 Ringing

Via this message, Bob's UAS informs the proxy that the softphone was able to produce an alarm to notify Bob that he has an incoming call. The message looks like the following:

```
SIP/2.0 180 Ringing
Via: SIP/2.0/UDP 192.1.0.1:5060;branch=8943170feab3209fcd8374.2
Via: SIP/2.0/UDP 192.1.0.200;rport
To: <sip:bob@company.com>
From:  “Alice” <sip:alice@provider.com>
```

```
Call-ID: 42243331@192.1.0.200
CSeq: 2390 INVITE
Content-Length: 0
```

Note that the *To* and *From* headers in a response are always **identical** to the headers of the corresponding request. This means that the headers in the response do not indicate the direction of the response but the direction of the corresponding request.

10. 180 Ringing

The message is forwarded to Alice's proxy.

11. 180 Ringing

The message is forwarded to Alice's UAC. This informs Alice that Bob was located and notified of her call.

12. 200 OK

Bob picks up the phone (since it is a softphone he probably clicks on a button on the softphone's Graphical User Interface (GUI)). His UAS sends a 200 OK message with a SDP payload attached to it. This message looks like the following:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP 192.1.0.1:5060;branch=8943170feab3209fcd8374.2
Via: SIP/2.0/UDP 192.1.0.200;rport
To: <sip:bob@company.com>
From:  “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
CSeq: 2390 INVITE
Content-Type: application/sdp
Content-Length: 153
```

The SDP payload is similar to the one attached to the INVITE request:

```
v= (Anything - Not used in SIP)
o= (Anything - Not used in SIP)
s= (Anything - Not used in SIP)
c= IN IP4 192.2.0.100
t= (Anything - Not used in SIP)
m= audio 19564 RTP/AVP 0 101
a= rtpmap:0 PCMU/8000
```

With this message Bob's softphone (his UAS) informs Alice's SIP phone (her UAC) of the compatible media sessions which his softphone supports and how it can be accessed (IP address, port number, etc).

13. 200 OK

The 200 OK is forwarded to Alice's proxy.

14. 200 OK

The 200 OK is forwarded to Alice's UAC. In this example, this message informs Alice that Bob picked up the phone. It also finishes the session negotiation with the SDP payload.

15. ACK

This request is sent to inform Bob's UAS that the 200 OK response was received correctly and that the session is ready to start. The ACK request can be sent again via the proxies, or directly since the two endpoints know each other's location. The ACK request is shown below:

```
ACK sip:bob@comapny.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 7123 ACK
To: <sip:bob@company.com>
From: “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
Content-Length: 0
Contact: “Alice” <sip:alice@provider.com;transport=udp>
```

After the ACK request is sent, Alice and Bob begin the media session. The media session is started as a separate session using protocols other than SIP, such as the Real Time Protocol (RTP). Of course, the SIP proxies do **not** take part in this session. Note that unlike other requests, there is no response to the ACK request.

16. BYE

After the session finishes, one of the endpoints (in this example Alice) sends a BYE request, informing Bob's UAS that she wants to terminate the session (see Figure 2.3). Again, the BYE message can either be routed through the proxies or go directly. This request looks like the following:

```
BYE sip:bob@comapny.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 1003 BYE
To: <sip:bob@company.com>
From: “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
Content-Length: 0
```

17. 200 OK

With this 200 OK message, Bob's UAS informs Alice's UAC that the BYE request was successfully received and terminates the session. Note that Bob's UAS could reply first with a

180 Trying response and then with the 200 OK. That could be useful if Bob's UAS required some time to stop the media session for some reason.

In the example described above, SIP messages are routed by the two SIP proxies. In an alternative scenario, one or both of the SIP proxies could act as a redirect server. For example Alice's redirect server would reply with a 3xx response pointing to Bob's proxy and Alice would repeat the request directly to it.

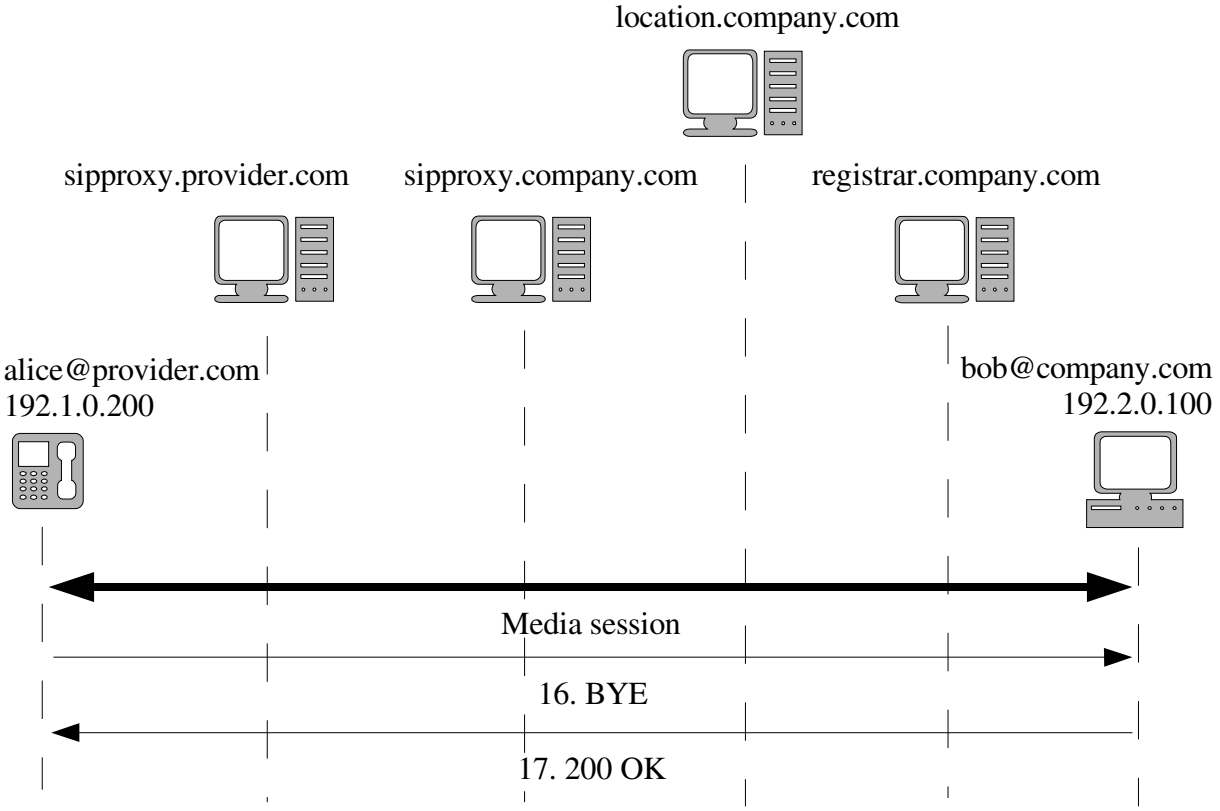

Figure 2.3 - SIP example call termination

2.1.8 SIP Transactions and Dialogs

In order to successfully issue messages and route them correctly, SIP entities need to maintain state concerned their various operations. For this purpose, SIP defines two state machines: transactions and dialogs. These two state machines are very important for the operation of the various SIP entities. In this paragraph we provide a brief description.

As stated in [2], a **SIP transaction** consists of a request along with all responses caused by that request (zero or more provisional responses and one final response). Retransmissions of the same request belong to the initial transaction. In the case of an INVITE transaction the ACK request is considered part of the transaction and not a new one. Transactions involve one client (UAC) and one or more servers[1] (UASs). Therefore, there are two types of SIP transactions: a **client transaction**, which is a state machine maintained at the client's side and a **server**

1. Forking can cause a certain request to have many receivers, and therefore many servers.

transaction, maintained at the server's side. For more details about the state machine of SIP transactions refer to [2].

All SIP entities, except for the stateless proxy, maintain a transaction state machine while sending and receiving messages. This means that messages in SIP are not processed independently, but rather according to the transaction they belong to. A transaction is a relation between two SIP entities that communicate with SIP **directly**. In the example described above, when initiating the session Alice has a transaction with her SIP proxy, which in turn has a transaction with Bob's proxy, and so on.

A **dialog** is another, higher level, state machine that unlike a transaction, is a relation between two endpoints of a SIP session (Alice and Bob) and may last for several transactions (request – response pairs). Not all SIP message exchanges require the maintenance of a dialog. According to the current SIP specification only INVITE requests generate a dialog. Like a transaction, a dialog helps the session endpoints in sending, receiving, and processing SIP messages. Figure 2.4 illustrates the relation between a dialog and a transaction.

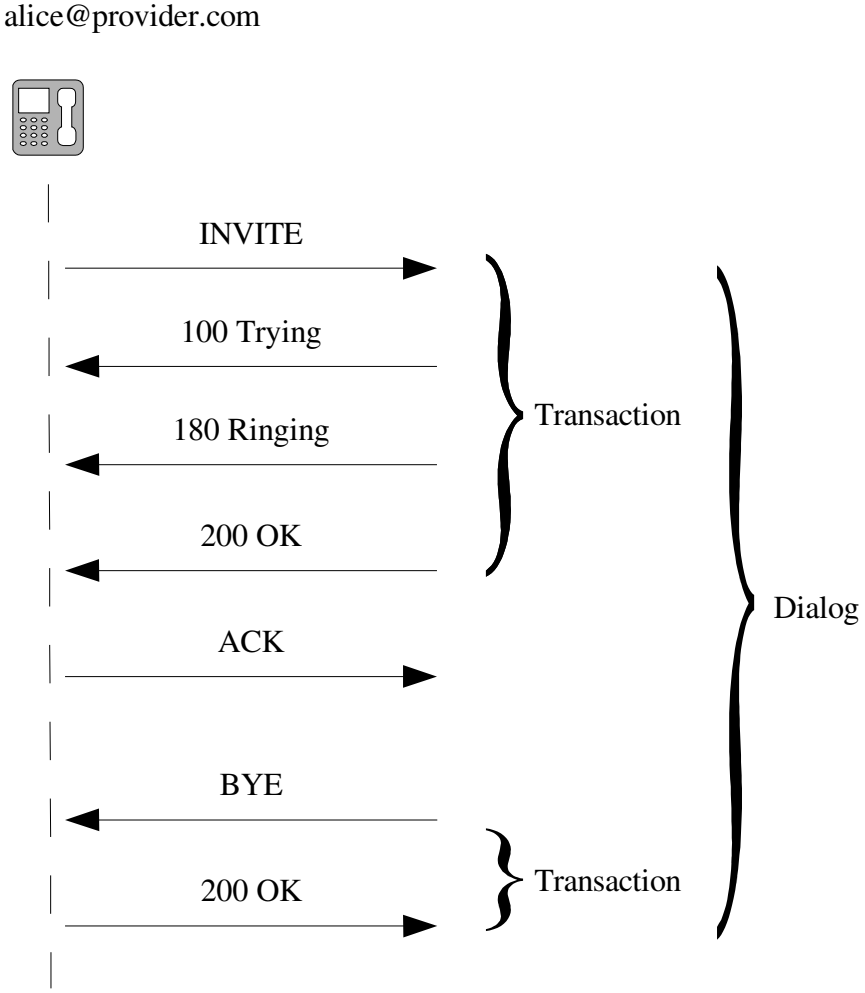

Figure 2.4 - SIP Transactions and Dialogs

2.2 IP Multimedia Subsystem (IMS)

The IP Multimedia Subsystem is an architecture proposed by the 3rd Generation Partnership Project (3GPP) [14] and its main purpose is to offer SIP support for 3GPP specified wireless networks, namely GSM/EDGE Radio Access Networks (GERANs) and UMTS Terrestrial Radio Access Networks (UTRANs). IMS has been designed as an extension to existing mobile operator networks in order to support multimedia services for mobile users. The architecture is described in [15], [16], and [17]. Note that since the architecture is still under intense development, these specifications may change at any time. In this section we will give a brief overview of the subsystem.

Figure 2.5 illustrates a general IMS architecture. The entire system consists of three main parts: the access network, the IMS Core Network (IMS CN), and the Application Server network. According to the 3GPP specifications, the three parts of the architecture **need not** be under the same administration and therefore they can be owned by different operators.

The IMS Core Network is an IP/SIP-enabled network that consists of four main entities: the Proxy Call Session Control Function (P-CSCF), the Serving Call Session Control Function (S-CSCF), the Interrogating Call Session Control Function (I-CSCF), and the Home Subscriber Server (HSS). The CSCFs are all SIP-enabled entities, but they can also understand and use other protocols such as the Common Open Policy Service (COPS) [18] protocol in order to satisfy QoS requirements. Below we give a short description of these four entities.

a) Proxy Call Session Control Function (P-CSCF)

The P-CSCF is the initial interface of the IMS towards its users. It is discovered as soon as a mobile user attaches to the GGSN. The discovery mechanism can be via the DHCP and DNS procedures described in paragraph 2.1.7.1. In addition, (as described in [17]) the P-CSCF can be discovered during a GPRS-Attach, when the mobile terminal establishes a Packet Data Protocol (PDP) context with the GGSN. Regardless of the method used, the terminal finally receives a list of IP addresses of several P-CSCFs, of which one is selected.

The P-CSCF mainly performs the following two functions:

SIP proxy function: The P-CSCF is responsible for routing SIP messages appropriately for home and roaming users. Received SIP requests are routed either towards the S-CSCF or towards other IMSs or generic SIP networks.

Policy Decision Function (PDF): This function involves taking the appropriate actions in order to deliver the required QoS to the user. In this role the P-CSCF does not act as a SIP entity but uses other protocols, such as the Common Open Policy Service (COPS) [18] protocol, in order to transfer QoS policies to the network's GGSN. To perform this function, the P-CSCF must have policy decision privileges on the user's GGSN.

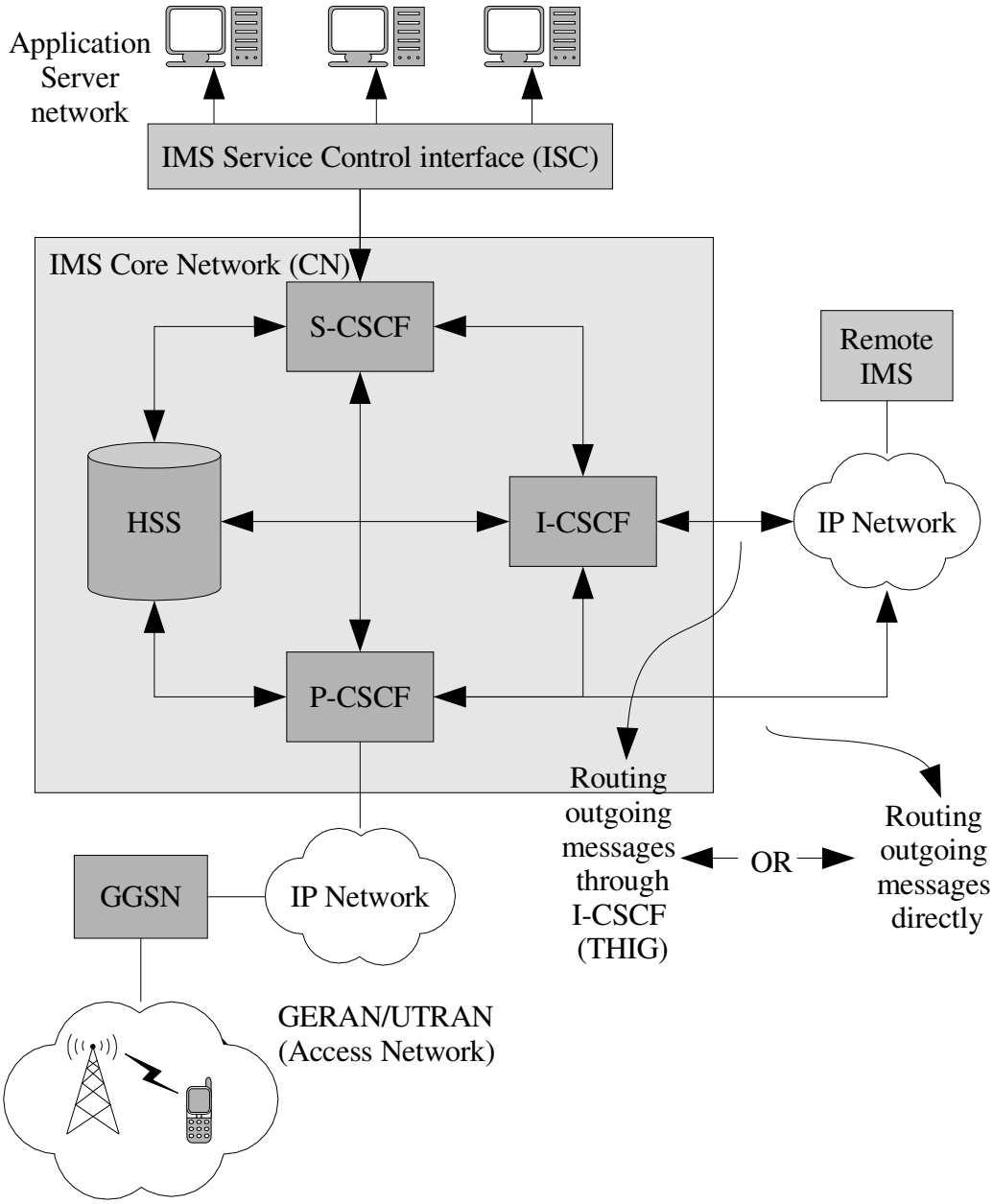

Figure 2.5 - IMS overview

b) Serving Call Session Control Function (S-CSCF)

As soon as a mobile terminal attaches to the GPRS network and discovers the P-CSCF, a S-CSCF must be assigned to it. This assignment is performed according to several criteria including user preferences, user service requirements, location of the user's P-CSCF, and location and capabilities of available S-CSCFs. This means that the assigned S-CSCF does not need to belong to the same IMS as the P-CSCF.

The S-CSCF performs the following functions:

SIP proxy function: The S-CSCF's main function is to locate the user and route messages to and from him/her. This means that the S-CSCF must be able to identify the type of service requested by the entity producing the request and handle it accordingly. It is important to note that the assigned S-CSCF lies always in the path of all calls to and from the user.

Accessing the Application Server Network: The S-CSCF is the interface of the IMS towards the service networks. It accepts SIP requests from users and accesses the services on their behalf, through the IMS Service Control Interface (ISC interface). Since the service network can include a large variety of services, the ISC usually consists of one or more specialized interfaces such as the Open Service Access (OSA) [19], the Customized Application Mobile Enhanced Logic (CAMEL) [20], etc.

Providing interworking with other network types: The S-CSCF is responsible (possibly in conjunction with a specialized gateway) for handling inter-working with other non-IP networks such as the Public Switched Telephony Network (PSTN).

Providing SIP-related services: The S-CSCF is used to provide SIP-related services to the user. Therefore it can act as a SIP registrar, a SIP presence server, and so on.

c) Interrogating Call Session Control Function (I-CSCF)

The I-CSCF is used to handle SIP messages between IMSs. For incoming requests (i. e. heading towards the IMS), the I-CSCF is the entity receiving the request and is responsible for locating the appropriate S-CSCF that can satisfy the request. In many cases, the I-CSCF also handles outgoing requests towards other IMSs. When a P-CSCF decides that a particular request from a user must be handled by another IMS, it can either forward the request directly to the I-CSCF of the remote IMS (as shown in Figure 2.5) or forward it to its local I-CSCF that handles the inter-IMS communication. The latter method is used when an operator wishes to hide the internal IMS architecture from the outside world. In this case the I-CSCF acts as a Topology Hiding Inter-network Gateway (THIG).

d) Home Subscriber Server (HSS)

The HSS is a database server that performs a variety of functions including Authorization, Authentication, and Accounting (AAA). It contains information about users, profiles, passwords, preferences, and so on. It is accessed by all other entities of the IMS when they need user specific information. In addition, the HSS may also be queried by Application Servers.

Figure 2.6 gives an example of how SIP calls made by a user are routed. Note that the S-CSCF is responsible for routing Bob's call towards Alice, towards an Applications Server, or towards another non-IP network, depending on the request it receives. We also assume that IMS operators use I-CSCFs as Topology Hiding Inter-network Gateways (THIGs) in all cases.

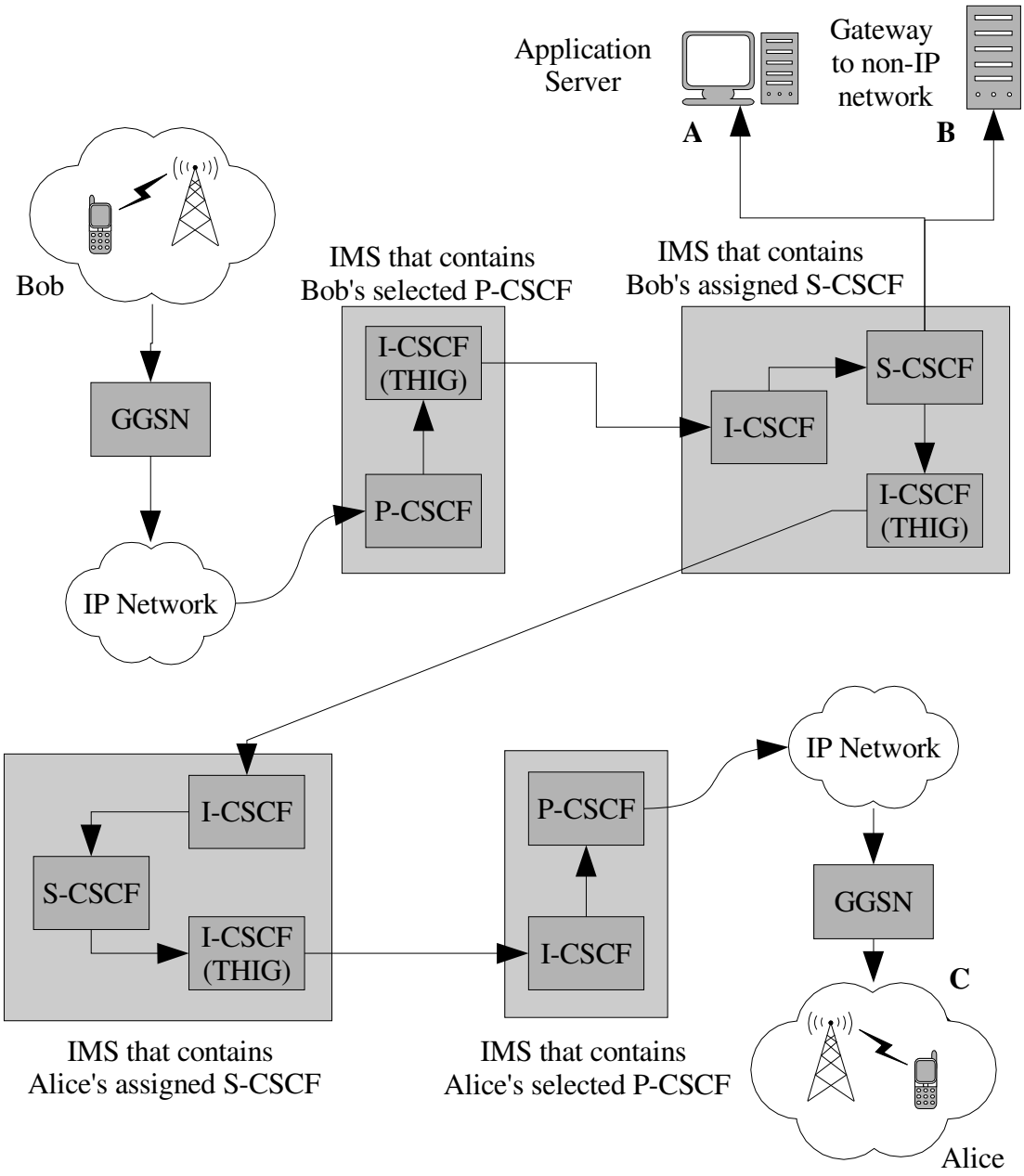

Figure 2.6 - SIP call routing in the IMS.

Bob's assigned S-CSCF routes the call towards (A) an AS, (B) a non-IP network, or (C) Alice (her assigned S-CSCF)

3. SERVICE POLICY MANAGEMENT FOR SIP-BASED SERVICES

This chapter studies the methods and mechanisms that have been proposed for service policy management for SIP-based services. In section 3.1, we introduce a number of terms that will be used in the discussion in the sections that follow. In section 3.2 we describe mechanisms that can be used in all SIP networks and in section 3.3 we survey methods proposed for 3GPP's IP Multimedia Subsystem.

3.1 Policy management concepts

3.1.1 Policy-based and non-policy-based management

System management in general is an important issue especially when dealing with large scale or highly complicated systems. Management can be **policy based** or **non-policy based**.

To give an example, suppose that we have an access control server and ten Network Access Servers (NASs) such as WLAN Access Points, Switches, etc. The system administrator wants to control the access to the NASs so as to optimize the network's performance according to a number of criteria. One way to solve this problem would be to have the NASs 'ask' the control server each time they get a request for a new connection. The reply of the control server depends on the request's contents and the overall state of the system. This scheme is fairly simple. Most of the processing is performed by the control server and the NASs are simple machines that accept or deny access according to the replies they receive from the control server. However, this solution has two major drawbacks. First, the channels to and from the control server will be heavily loaded and second, the request/reply process might take considerable amount of time even if the traffic to the control server is low and this server has adequate processing power.

Another solution to the problem would be the following. Instead of having the NASs make requests to the control server, they could receive **directions on what to do** when they receive requests. These directions would be distributed by another node that is able to keep track of the overall state of the system. This requires that the NASs are smart enough to accept and interpret the directions they receive. If the system's design and operation does not require that directions be distributed too frequently, this solution has several benefits compared to the first one. The traffic towards the control node is lower, the processing load is distributed more evenly, and the system scales better.

The first solution is an example of non-policy based management and the second is an example of policy based management. The directions mentioned in the second solution are the policies.

3.1.2 Generic policy-based management model

Here we introduce the basics of policy management in order to clarify a few important terms. Figure 3.1 shows the basic entities involved in such a system. Note that what is shown is a simplified model. An actual policy management system can be more complicated. The system includes the following entities:

a) Policy repository

The policy repository contains all the policies that are being used by the system. For example, there might be policies that guarantee congestion avoidance (by denying new users), QoS, and so on. A policy is a set of rules or instructions on the course of action taken when a specific event occurs.

b) State data

This entity contains all the necessary data to describe the state of the managed system, including previously installed policies. An example of state data for a layer-2 switch for example can be the number of Virtual Local Area Networks (VLANs) present or the forwarding policy on its interfaces. To remain up to date, state data must be updated as frequently as possible. Since the actual system state can change rather frequently, a critical issue for a system is state data maintenance.

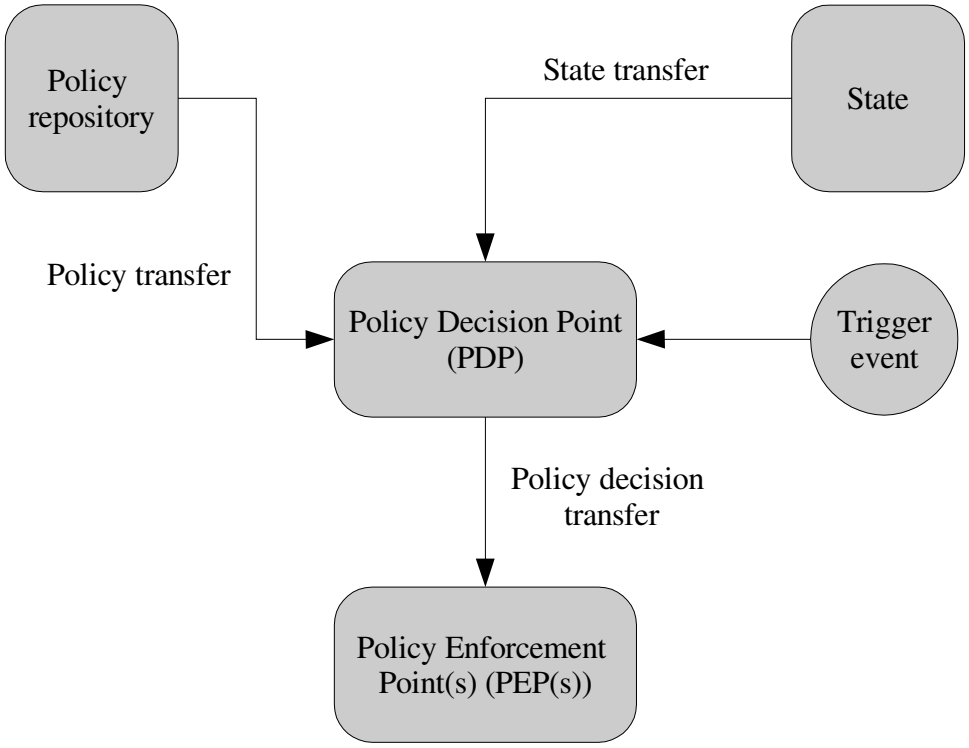

Figure 3.1 - Generic policy management model

c) Trigger event

A trigger event is any event that causes a new policy decision to be made. It can be a request from any of the other entities of the system, a new command from the system administrator or a simple timer that causes new decisions to be made at regular time intervals.

d) Policy Decision Point (PDP)

A policy decision point is where the policy rules, state data and the trigger event are processed in order to produce a policy decision. As shown, a PDP with full functionality must have two types of operation: **monitoring** and **making policy decisions**. Monitoring involves keeping track of the state of the managed entity. Monitoring is of high importance since the PDP must avoid using stale information which can lead to an erroneous decision. On the other hand, in many cases issuing frequent state updates might be impossible or very expensive. Thus, the trade-off while monitoring is keeping state information fresh versus making effective usage of the bandwidth given. Policy decision making is the act of configuring the PEP with a new policy.

In simple cases a policy-based management system will include a single Policy Decision Point. Deploying more PDPs is of course possible but will require an efficient means of synchronization of the PDPs so that stability is guaranteed and we do not have strange phenomena taking place such as when the policy decision of one PDP is canceled by another and vice versa.

e) Policy Enforcement Point(s) (PEP(s))

These entities are responsible for processing policy decisions they receive by taking the appropriate actions.

As stated, the a bove model is generic. Designing a policy manager to perform a specific task involves applying this model and making the appropriate decisions, such as defining the protocols used by the entities in order to communicate and deciding where should each function (PEP, PDP etc) be positioned.

3.1.3 Service policy management

Service policy management regards the provision or usage of a service. Depending on who is acting as the Policy Decision Point we can identify two basic types (see Figure 3.2):

Provider-side service policy management regards the management of service provision to the users. In this case the provider acts as the PDP. Examples are the AAA function performed by a network operator (access control), bandwidth allocation for QoS provision to certain users, and so on.

User-side service policy management regards the management of service reception from a user. This type of management enables a user to control the way that he/she uses a service. Therefore, the user, or a group of users, acts as a PDP. Examples of this type of management

are accept/block caller list, caller hiding, bandwidth control to minimize costs, link or interface selection according to type of communication needed, user preferences, and so on. According to where these policies are applied, we can identify two subtypes for user-side service policy management, one for outgoing and one for incoming traffic.

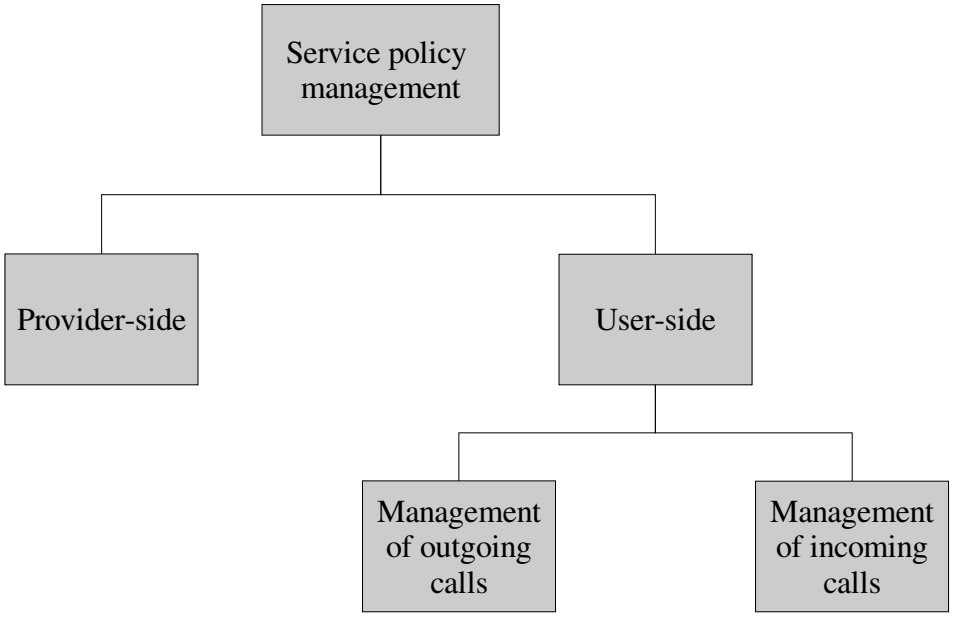

Figure 3.2 - Service policy management categorization

In the following chapters we focus solely on user-side of service policy management. Thus, we will not investigate service policy management from the provider's side (QoS management, provider resource management, etc). Moreover, the services in question are all SIP-based, meaning that they can be accessed via SIP. Thus, from this point on, the term service policy management implies **user-side service policy management of SIP-based services** unless otherwise specified.

3.2 User-side service policy management in generic SIP networks

In this section we present a number of methods that have been proposed for user-side policy management for SIP-based services. We mainly focus on proposals made within the IETF, as RFCs and Internet Drafts.

3.2.1 Call Processing Language (CPL)

IETF has specified a framework for a Call Processing Language in [21]. In addition, there is currently an (expired) Internet Draft ([22]) with a detailed CPL description.

CPL is a language based on the Extensible Markup Language (XML) that allows users of Internet Telephony Services to control the way incoming and outgoing calls are routed. The language is actually used to write scripts with directions on how certain messages (such as SIP INVITE messages) are to be handled. Both SIP User Agents (UAs) and SIP servers can execute CPL scripts. Therefore, a user can upload his/her Cpl script to any SIP entity that will accept it. This user is called the *script owner* and the entity where the CPL script is about to be executed is called the *CPL server.*

When a SIP entity receives or is about to send a message, it checks its store of (previously uploaded) CPL scripts to see if any script owner matches the sender or the receiver of the message. If it finds such a script, it executes it using the SIP message as input. The final outcome of a message process is called an *operation.* The current CPL Internet Draft [22] specifies three *signaling operations* and two *non-signaling operations.* The signaling operations are *proxy, reject,* and *redirect* and the non-signaling operations are *mail* and *log*. What a CPL script basically does is match a message against a set of conditions to decide upon one or more of the aforementioned operations.

Unfortunately, there has been no specified way to upload CPL scripts. Proposals include a web file upload, SIP REGISTER message payloads, using the Simple Network Management Protocol (SNMP)([23] and related RFCs) functions, using the Application Configuration Access Protocol (ACAP) ([24]), using the Lightweight Directory Access Protocol (LDAP) ([25]), and remote file systems such as the Network File System (NFS) ([26]).

As an example, a policy that can be implemented with CPL is call forwarding. A user can upload a script to forward incoming calls to his/her office phone during working hours, and to his home phone during non-working hours. CPL can also handle cases when a call goes unanswered. Therefore, the user in our example can forward the call to his mobile phone if a call to his home phone goes unanswered.

Figure 3.3 illustrates a typical scenario where CPL is used for service policy management. The SIP User Agent application includes the SIP UA component for handling SIP communication, the media interfaces, and the service manager. The service manager collects information from the device (e.g. time, battery charge, etc.), the interfaces, and the application user profile and preferences. Using this information it is able to construct CPL scripts and upload them to the server. In management terms, the service manager acts as the PDP, the device information base (s) and the media interfaces act as the system state, the user profile and preferences constitute the policy repository, and the CPL/SIP server acts as the PEP.

As we can see, CPL only provides the **decision making** function of the PDP, offering a way to express the policy rules. CPL is able to enforce policies both for incoming and outgoing calls as discussed in paragraph 3.1.3. The monitoring function of the PDP must be performed by other means. In most cases (as in Figure 3.3) the PDP is part of the UA application and therefore monitoring is performed within the application itself. If a user uses only one UA application at a time, this poses no issue. However, if more than one UA applications are used by one user, then there might be problems that have to do with PDP synchronization. For example, consider a user that has a mobile phone with a service manager that at some point issues a policy to forward all incoming calls to the home telephone. The home telephone on the other hand has issued a policy that forwards all calls to the mobile phone. While this error can be detected at the CPL server, the UA applications are (still) not “aware” of each other. Another example is

when one UA application issues a policy that another UA application must be used while the latter is down for some reason. The lack of a central service monitor can be a crucial problem in highly volatile environments when links are lost and recovered frequently, where services are only available for a short period of time, etc.

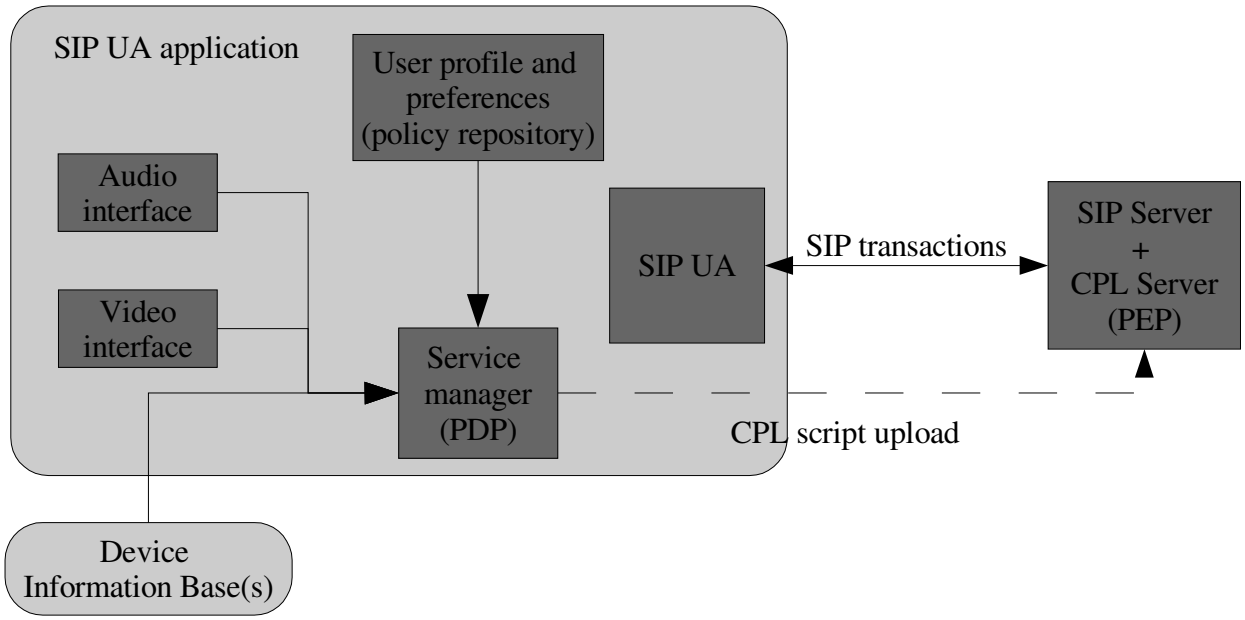

Figure 3.3 - Service policy management using CPL

Another disadvantage of CPL is that it does not support making decisions based on the media type of the requested service. A user, for example, might wish to forward a phone call to his/her phone and a fax to his fax machine. The proxy should therefore make the decision according to the SDP payload contained in the initial INVITE. Unfortunately, this is not supported by CPL at the moment.

3.2.2 Other methods

Apart from CPL, there have been a few other alternatives for user-side service policy management, focused on how a user can affect processing of his/her outgoing calls. Rosenberg, Schulzrinne, and Kyzivat [27] propose a means for the caller to specify preferences on how his/her call should be routed or otherwise handled by intermediate entities. These preferences are expressed with three SIP headers (specified in [27]): *Accept-Contact*, *Reject-Contact*, and *Request-Disposition*. Schulzrinne and Polk propose a way so that the caller can indicate the priority by which his/her call must be handled by intermediate entities [28]. The caller expresses this preference with the *Resource-Priority* header whereas an intermediate entity can express the priority preferences it accepts with the *Accept-Resource-Priority* header. This method is intended to be used in emergency circumstances.

As we can see, these proposed alternative methods mostly address service policy management for outgoing calls, unlike CPL, which supports incoming **and** outgoing call handling.

3.3 User-side service policy management in the IMS

The mechanisms for user-side service policy management in generic SIP networks described above are also applicable to the IMS, since it is, to a large extent, a SIP-based architecture. Unfortunately, 3GPP has not specified standard ways to facilitate these mechanisms. However, CPL support may be offered by an IMS operator just like any other service. The most appropriate PEP in this case is the S-CSCF (possibly in conjunction with a specialized CPL Application Server, as proposed in [29]), since it is responsible for providing SIP related services to the user and always lies in the path of every SIP call to and from the user, as we mentioned in the previous chapter.

In Figure 3.4 we show how a CPL script can be uploaded to the S-CSCF so that policies issued by the user can be enforced by the S-CSCF for calls to and from the user. In this example the S-CSCF uses a CPL AS that handles the CPL execution. Note that the messages between the S-CSCF and the CPL AS are **not** SIP messages.

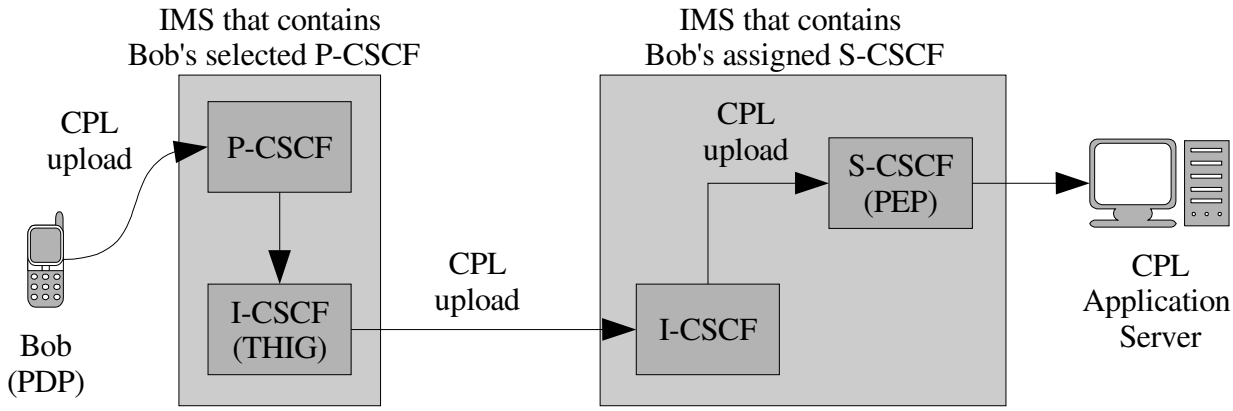

Figure 3.4 - CPL script upload to the S-CSCF

A common characteristic that all these management mechanisms have is that the policy is always enforced by an intermediate entity meaning that **the PEP always resides within a provider's network**. Placing the PEP at the provider's side is at first glance reasonable and providers intend to offer this PEP functionality as a separate service (in many cases as a value-added service). However, this architecture has a number of problems. First, many providers are often reluctant to give their users what they consider increased freedom to manipulate the service they receive. CPL is designed in such a way so that advanced and complicated policies cannot be expressed (e.g. loops are not allowed) so that the processing a CPL script would require is within acceptable limits. Second, there might be certain interoperability problems when different providers offer their users different service policy management facilities. In the next chapters we will investigate both options, i. e. placing the PEP inside and outside the provider's network, in order to analyze the benefits of each solution.

4. PROPOSED ARCHITECTURES FOR USER-SIDE SERVICE POLICY MANAGEMENT IN SIP NETWORKS

In the previous chapter we investigated possibilities for user-side service policy management. In this chapter we propose new architectures that aim to offer the user as much control as possible over the services he/she uses. To achieve this, we propose a new SIP entity, called the SIP Service Manager (SSM) with the task of monitoring the service network and issuing policies.

4.1 Motivation and goals

In the previous chapter we investigated the possibilities for user-side service policy management. From this discussion, one can identify a number of problems that these mechanisms show. We summarize these problems as:

- Although CPL is a very useful tool for policy enforcement, it remains quite limited. No policy enforcement according to media type or other service attributes is supported. CPL takes into consideration *only* the information contained in the SIP header of a packet. This problem could be overcome by extending the functionality of CPL. Such extensions, however, do not yet exist.

- Even when CPL is used, the question of which entity (one of the user's UAs or another node in the network) will upload the script to the provider (i.e. the entity acting as the PDP) remains unanswered. Since a CPL script involves policies not only about one, but possibly all services controlled by the user, this PDP must, in some way be able to check the status of all the services that the user controls (monitoring). Such an entity does not yet exist.

- Monitoring of the set of services controlled by the user can have other benefits in the case of a wireless mobile user. Many of these mobile devices include a number of sensors and interfaces that can provide useful information to the system's PDP, thus providing context information. A smart PDP can aggregate and combine this information in order to present it to the user in some way in order to implement smart policies.

- There are cases when a user needs to have a **central contact representative** to a SIP network that can optionally hide the actual services that he/she controls, making them appear in a uniform, simple way to the outside world and possibly as a single service. Thus, all SIP traffic will be routed through this central representative. This may also be desired in cases where additional security is needed. For example, suppose that Alice wishes to use an online radio client, configured with the URI

sipradio32@alicescompany.com to access Bob's online radio. What if Bob's radio, for security reasons, rejects all requests from UAs that have unknown URIs? Instead, Alice could route her request through another entity that would make her appear as alice@provider.com so that her request can be accepted. This entity may also contain Alice's authentication data (passwords and certificates) that she might be unwilling to use from devices that are not entirely controlled by her (e.g. public and company telephones).

In this thesis we propose an alternative way for user-side service policy management that addresses the problems described above. Our proposal focuses on giving the user more control, especially in handling incoming calls. More specifically, the desired solution must include the following features:

a) Extended control and full management functionality

Unlike existing mechanisms, the architecture should have full management system functionality. This includes a PDP with monitoring and policy enforcement functions. The PEP must be included and, as we will see, this might reside within the provider's network depending on how the system is used. The architecture must facilitate the enforcing of policies according to combined data that is gathered from the service network.

b) Ability to operate in mobile, highly volatile environments

As discussed in our introduction, users may be using several services (i.e. several User Agent applications) at the same time. Moreover, we presume that some of these services are not being used continuously, but might only be used for short periods of time. Such examples are a short game session, a telephone in a public telephone booth that a user uses for a short period of time, a fax machine in the company's office that is available for only ten minutes, etc.

c) Switching SIP providers (roaming)

In a fully mobile environment, a user can move across networks and across network operators as well. However, this may introduce complications, as different providers can have different service agreements with their customers and therefore offer different service policy management facilities. A service policy manager must be able to address these difficulties by making few assumptions of what the providers offer to the user.

d) Performance

Obviously, the system must make efficient use of the available communications resources (bandwidth), processing power and memory.

e) Security

Security is a major factor in the design of all contemporary communications systems. Therefore the architecture must support various kinds of security, such as information integrity, end-point authentication, and so on.

It should be noted that our proposed solution does **not** replace the existing mechanisms. On the

contrary, our aim is to extend existing mechanisms. Therefore, as we will see in the following sections, the proposed system is able to use these mechanisms as required.

4.2 Proposed design

4.2.1 Overview

The core of our proposed management system is a new SIP entity that we call the SIP Service Manager (SSM). The SSM is an intermediate entity that resides between the service network and the SIP backbone (see Figure 4.1) and combines proxy, UA, registrar, and redirect functionality, depending on the way it is used. Since this is a user-side management system, it is the user that controls and configures the SSM either directly or remotely.

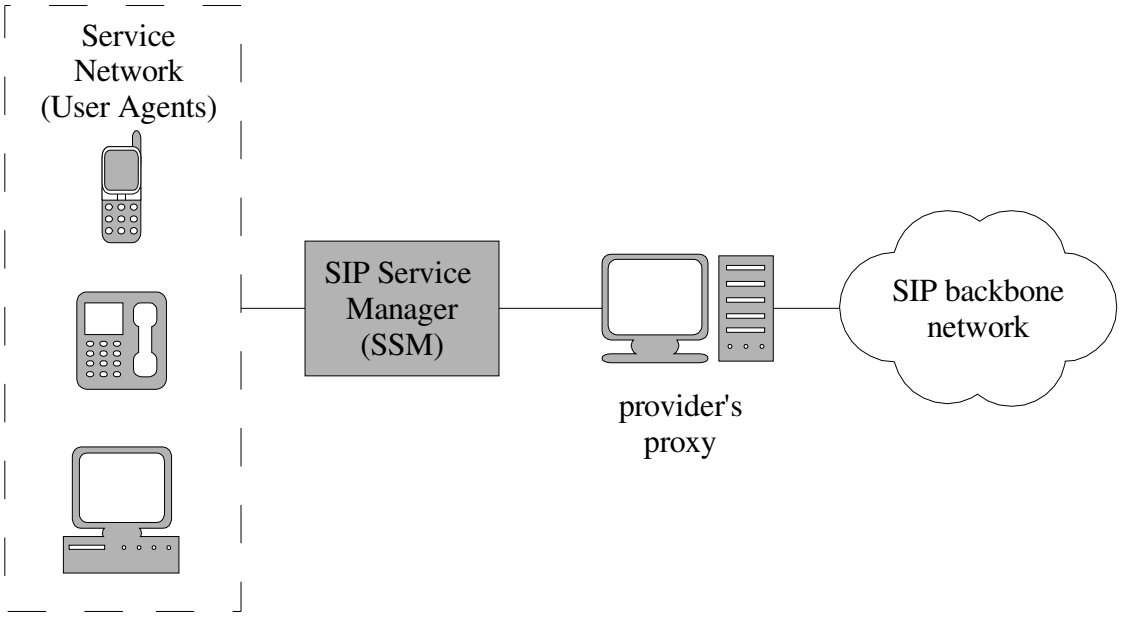

Figure 4.1 - SIP Service Manager overview

Before monitoring and managing a service, the SIP Service Manager must, obviously, be aware of it. This can be done by manual configuration by the user or by automatic service discovery. This thesis **does not** study service discovery mechanisms. Readers interested in the subject may refer to [30]. After the SSM is informed of the service's existence, it must **associate** with it. In this document we define two types of association: soft and hard.

A **soft association** simply involves discovering and monitoring the service. No configuration is made and no special action is expected from the service other than responding properly to the SSM's monitoring probes. A soft association may fail if the service refuses or fails to respond to the SSM's probes.

A **hard association** involves forcing SIP configuration (outbound proxy, URI, possible headers to add in messages, etc.) on the service in order to make it behave in a desired manner. The service may be expected to take special actions on its own, depending on the type of configuration. A hard association also includes monitoring the service. In this thesis we will not study means of performing a hard association. Possible ways to achieve this are by using the Trivial File Transfer Protocol (TFTP) [31] and manual configuration. In addition, a hard association can be performed as part of the discovery process.

The purpose of the SSM is to monitor the service network, present the status of the network to the user, and issue policies to the system's Policy Enforcement Point (PEP). Depending on where the PEP resides we can identify two types of operations for the SSM: one in which the PEP resides within the provider's network and one in which the SSM performs the PEP function itself. These two types of operation are presented in the next two paragraphs.

4.2.2 SIP Service Manager without PEP functionality

This type of SIP Service Manager (SSM) operation is the simplest of the two. A block diagram is given in Figure 4.2. As shown, the SIP Service Manager in this case consists of the following components:

Registration Client: The Registration Client is a UAC, which is responsible for **registering** with the SIP provider providing location information so that the user can be reached from the

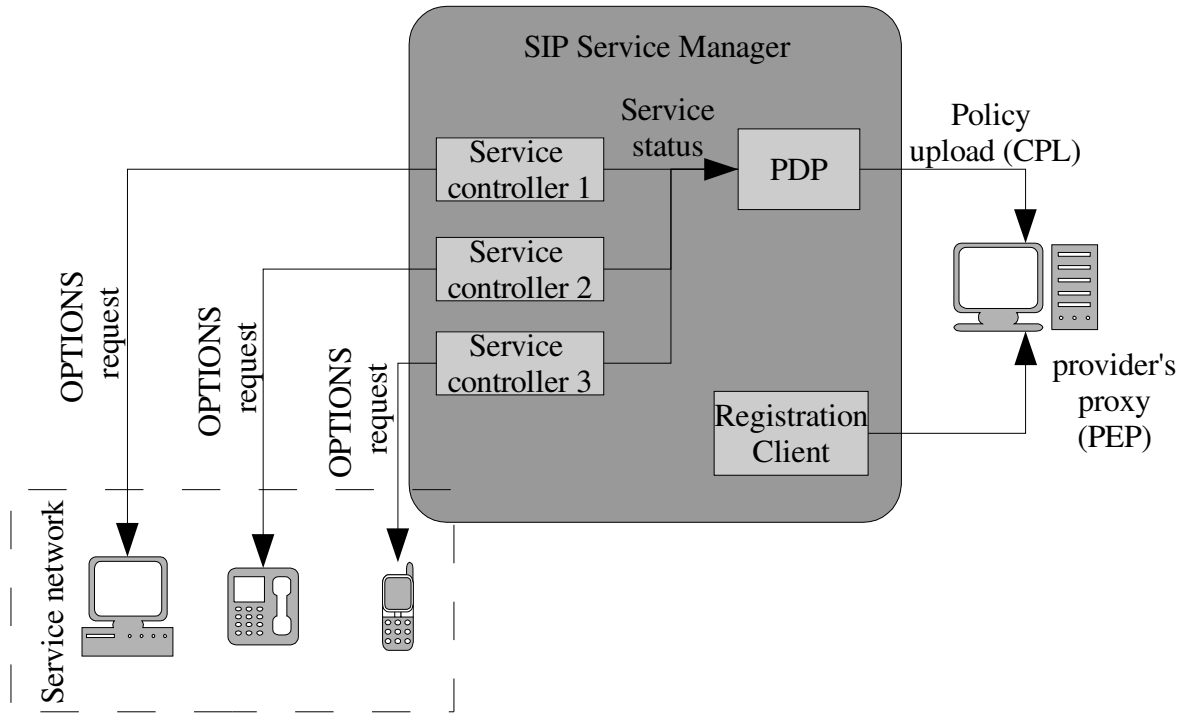

Figure 4.2 - SIP Service Manager without PEP functionality

SIP backbone. The UAC is configured with the user's **Address-Of-Record (AOR)**.

Service Controllers: The Service Controllers in this case are responsible for monitoring the service network. Each Service Controller is responsible for monitoring a single service and report the service's state to the PDP. Monitoring is performed by sending OPTIONS requests to the service (see also paragraph 4.2.4).

Policy Decision Point: The PDP component collects all information from the Service Controllers in order to determine the service network state. Following that, it issues appropriate policies to the provider's proxy, which in this case acts as the PEP of the management system. Policies in this case are expressed using the Call Processing Language. This assumes of course that the provider supports CPL script uploading. A straightforward way to upload the CPL script is attaching it to a REGISTER request made by the Registration Client. Alternatively, any of the methods mentioned in paragraph 3.2.1 can be used.

The major benefit of this type of SSM operation is that it is very simple and lightweight. Indeed, the SSM without PEP functionality includes only a number of simple SIP UACs plus the PDP component, whose complexity depends on the complexity of the policies that are applied. A UAS may optionally be added so that the SSM can process a number of SIP methods and respond to them appropriately. Below we summarize the advantages of this architecture.

- As we mentioned above, this type of SSM is simple and has low demands on memory and processing power. To justify this claim we can consider that this architecture includes only simple UACs (and optionally a UAS), whereas contemporary SIP softphones that run on PDAs and laptops include UACs, UASs, voice codecs and possibly other components that make these UA applications much more demanding than the SSM proposed in this section.

- The association between the SSM and the managed services can be soft. This means that the SSM can avoid forcing configurations on the UA application which might be desirable in cases where there are security or ownership issues.

- The SSM is the only entity in the management system that acts as the PDP. Therefore, the risk of issuing conflicting policies (as we described in paragraph 3.2.1) by different PDPs is eliminated.

- Unlike other service policy management architectures, the PDP in our proposal is aware of the status of the service network. Furthermore, it is able to collect other useful information (see paragraph 4.2.4) from the services and combine them to issue smarter and more effective policies.

Although the architecture described in this paragraph allows for advanced service policy management compared to existing solutions it does have two limitations. First, the architecture relies on CPL with its known limitations. There might also be cases where providers will be reluctant to allow the use of future CPL extensions. Second, the service network is open to the outside world. Services can be accessed by the SIP backbone without the participation of the provider's proxy. While the security effect of this can be limited to a certain extent, having an open topology might be unacceptable. This also means that the architecture does not solve the

authentication problem discussed in section 4.1.

4.2.3 SIP Service Manager with PEP functionality

To address the problems that we mentioned in the previous paragraph we proceed with our investigation by adding PEP functionality to the SSM. This means that the SSM in this case is not only capable of monitoring the network and issuing policies (PDP function), but also enforcing these policies by actively routing and manipulating the content of SIP messages. As we will see, this grants the user full control over the services he/she controls, at the cost of increased complexity of the system.

Before describing this alternative architecture, we need to specify two new terms that we use for convenience purposes: the public service and the private service.

A **public service** is a UA application that is configured with a Globally Routable UA URI (GRUU) either statically or dynamically within an already existing session, as described in [32]. Since a public service can access (and can be accessed by) the SIP backbone directly, it does not rely on the SSM for registration or for forwarding messages. SSM's association with a public service can be soft or hard.

There are various cases where a service may not have a GRUU, either deliberately or because there are no GRUUs available for some reason. These UAs are configured with a non-globally routable (private) URI, which is known only to the SSM. We will refer to these services as **private services**. A private service uses the SSM for registration and for proxying messages. The association between the SSM and the private service is always a hard association.

Figure 4.3 shows a block diagram of the SSM with PEP functionality. We observe a number of components that have been added, which we describe below:

Registrar: The registrar component acts the same way as described in section 2.1. It is responsible for processing REGISTER requests from the service network and storing the location information received in the Location Service.

Location Service: The Location Service is a database that stores location information. Its entries are added or deleted only by the Registrar (write access). The stored information is used by the Service Controllers in order to locate the services (i.e. get the IP address) they control (read access). The Location Service may reside on the same network node as the SSM or it might by a remote database depending on the implementation.

Service Controllers: The Service Controllers in this case have proxy and redirect functionality. They are responsible for forwarding requests to and from their associated service or redirecting calls to their associated service when needed. A Service Controller is also able to modify SIP messages (adding, removing, and modifying headers) when needed. These functions are in addition to the monitoring function described in the previous paragraph.

Service Controllers must be able to perform stateful proxying and therefore they need to

maintain transaction states for all the requests they handle. As messages are routed through the Service Controller back and forth, pairs of server and client transactions are created. These pairs together form a map, which is called the transaction map. The Service Controller consults this transaction map to determine the peer SIP transaction a certain message needs to be routed to (see Figure 4.4).

Policy Enforcement Point: The Policy Enforcement Point (now residing within the SSM) is responsible for applying the policy issued by the PDP. Practically, the PEP chooses which of the Service Controllers will handle an incoming request.

As mentioned, the SSM with PEP functionality is capable of routing messages to and from a service. We will now describe how routing of outgoing (from the service network to the SIP backbone) and incoming (from the SIP backbone to the service network) calls is performed.

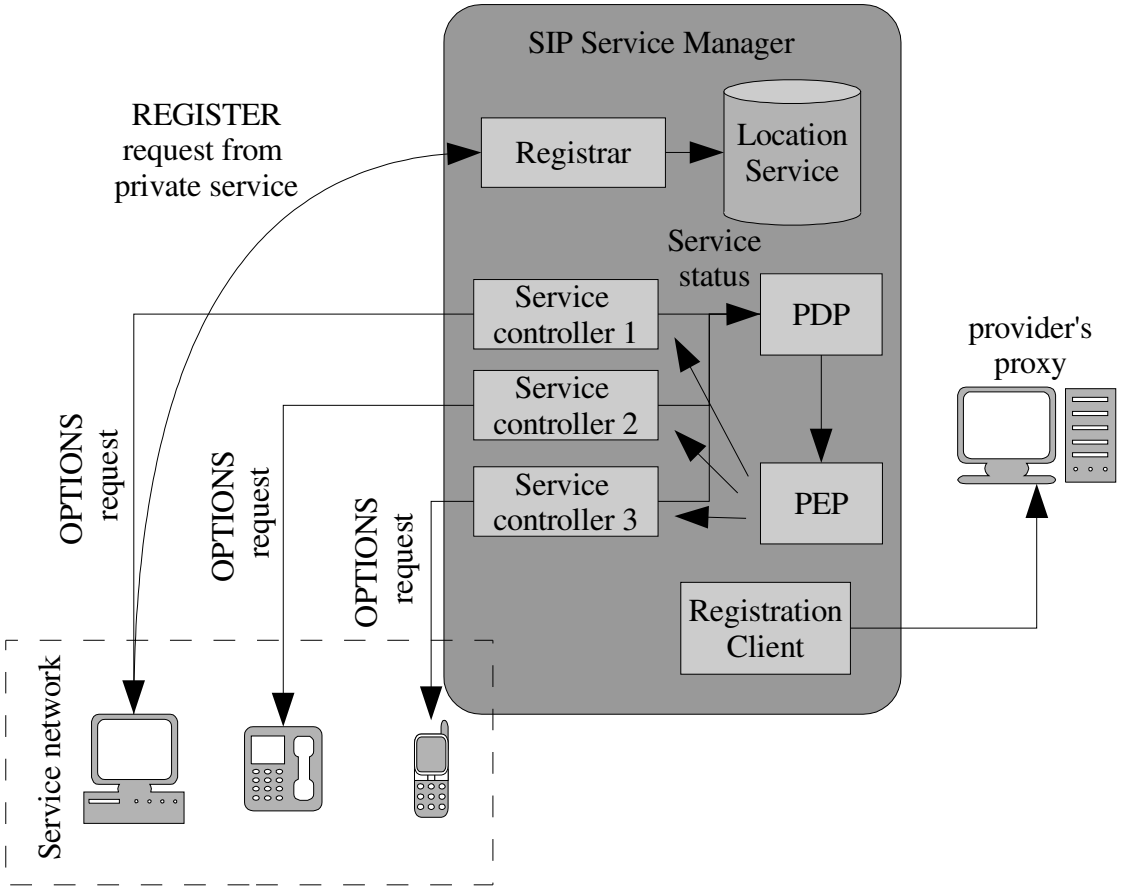

Figure 4.3 - SIP Service Manager with PEP functionality

An **outgoing call** occurs whenever the user initiates a session from one of the services associated with the SSM. We can identify two cases: (a) the user initiates a session from a public service and (b) the user initiates a session using a private service.

As shown in Figure 4.5, when a public service makes an outgoing request, the request is routed directly to the SIP backbone (via the same or some other SIP provider). The SIP Service

Manager is not involved in this case, **unless** the user forces the request to be routed via the SSM using the *Route* header field. This may be desirable if the user wishes to use some of the SSM's functionality.

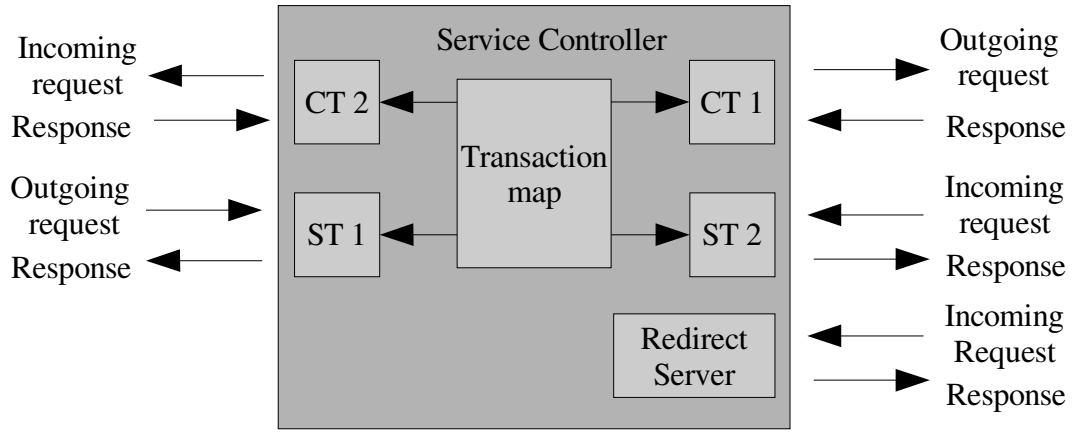

Figure 4.4 - Service Controller components

When the request is made from a private service, it is routed through the SSM and more specifically, through the associated Service Controller, which is responsible for modifying and forwarding the request appropriately according to the outgoing call policy. In practice, the outgoing policy determines the SIP proxy that will be the request's next hop and the set of headers that need to be added to or removed from the request. Note that since the private service

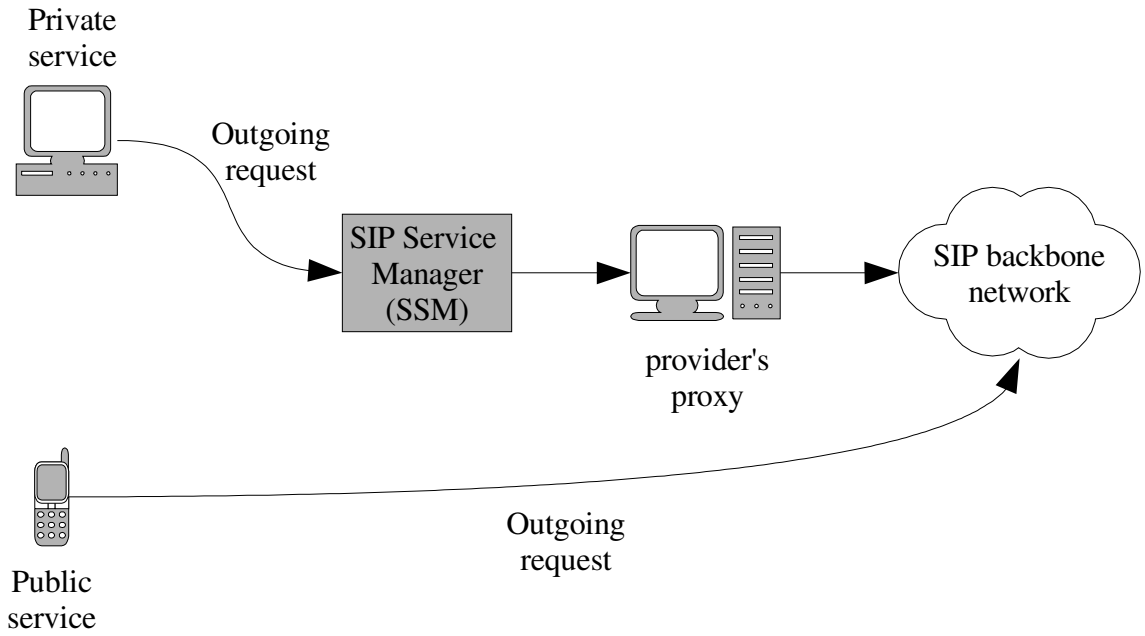

Figure 4.5 - Outgoing message routing

does not use a Globally Routable UA URI (GRUU), modifying the request involves changing all headers that include the non-GRUU so that they contain the user's well known URI, which is the Address-Of-Record. This modification makes the request appear as if it was made from the SSM itself. In this case, the SSM performs a type of multiplexing that resembles the function of a Network Address Translator (NAT).

For example, suppose that Alice uses a private service configured with the non-GRUU: sipphone@alicesnetwork with her SSM and sends an INVITE request to initiate a session. The INVITE message received by the SSM would look like the following:

```
INVITE sip:bob@company.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
To: <sip:bob@company.com>
Content-Type: application/sdp
From: “Alice's SIP phone” <sip:sipphone@alicesnetwork>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice's SIP phone” <sip:sipphone@192.1.0.200;transport=udp>
<SDP payload not shown>
```

The SSM would have to change the *From* header so that the request becomes like the following:

```
INVITE sip:bob@company.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
To: <sip:bob@company.com>
Content-Type: application/sdp
From: “Alice” <sip:alice@provider.com>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice's SIP phone” <sip:sipphone@192.1.0.200;transport=udp>
<SDP payload not shown>
```

The display names might, of course, differ. The Service Controller, since it acts as a proxy in this case will also manipulate the *Via* header to include the SSM's location so that requests will be routed the same way backwards in order to perform the appropriate de-multiplexing. Note that the *Contact* header is not modified since it contains location information that is needed to establish a session.

Responses to outgoing requests are routed in a similar manner. Responses to outgoing requests made by public services pass through the SSM only if the *Route* header field was included in the request and routing them towards the public service is simple, since no de-multiplexing is needed. For private services the Service Controller de-multiplexes the request by changing URIs the same way as in requests when needed. In both cases the Service Controller uses the transaction map to route messages to each direction.

Routing of incoming requests is shown in Figure 4.6. Since the SSM is configured with the user's Address-Of-Record, all requests targeted to that user end up at the SSM. This is one of the benefits of the system, because the user may control as many services as he/she needs without having to inform the caller of their existence. This means that a caller will use the same AOR when he/she tries to access the callee's phone or fax, without having to know the specific URIs of these services. The decision on where to route the request is left up to the user controlling the SSM. However, there might be cases where this type of routing is not required and the caller wishes to contact the callee explicitly at a specific service. This poses no difficulty since, for public services, the callee can be contacted directly from the SIP backbone provided that the caller knows the URI of the public service (dashed arrow in Figure 4.6).

At this point, the SSM, having received the incoming request, must decide upon the appropriate service (or services in the case of forking) that will handle the request, in this way applying the policy for incoming calls. Policy application is discussed in more detail later in this chapter.

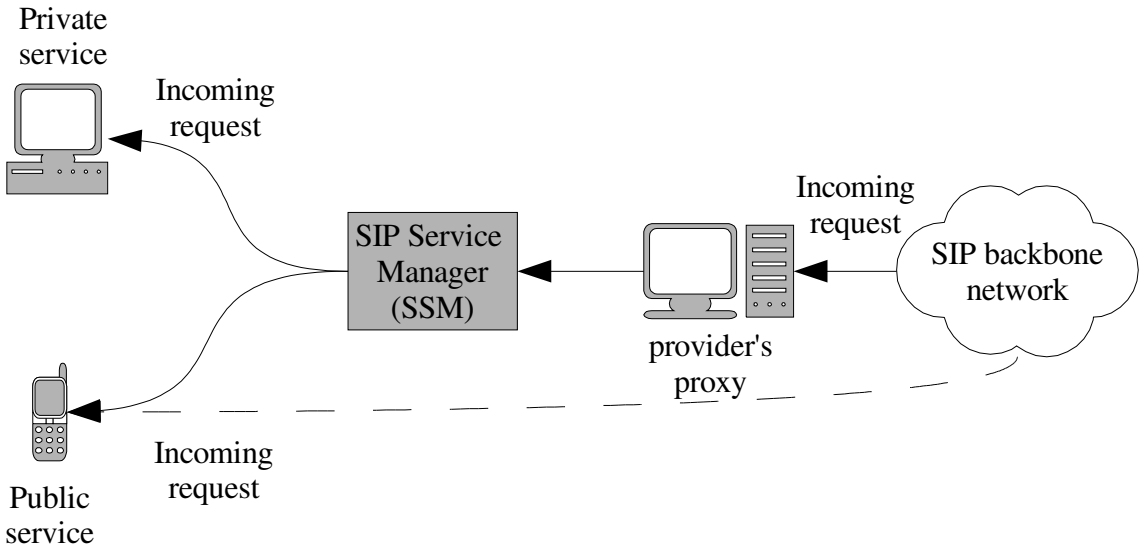

Figure 4.6 - Incoming call routing

After deciding the set of services that will handle the request, the PEP passes it to the corresponding Service Controllers. When a Service Controller receives an incoming request it has two options. If the associated service is public, the Service Controller can choose to either proxy or respond with a *302 Moved Temporarily* response, which will redirect the call to the public service (redirect server function), or forward the request itself (proxy function). In the case of a private service, the Service Controller can only forward the request consulting the Location Service of the SSM.

As in the outgoing call case, the Service Controller needs to modify a request appropriately when forwarding it. However, there is no need for multiplexing of incoming calls. All a Service controller has to do is just change the Request-URI of the request in order to match the URI of its associated service. The *To* header does not need to be changed as it represents the “logical” recipient of the request, who may not match the actual recipient. For more details on this the reader should refer to [2].

For example, suppose that Alice calls Bob, who uses his SSM to manage incoming calls. Alice only knows Bob's AOR, i. e. bob@company.com. The incoming INVITE request received by the SSM looks like the following:

```
INVITE sip:bob@company.com SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
To: <sip:bob@company.com>
Content-Type: application/sdp
From: “Alice's SIP phone” <sip:sipphone@alicesnetwork>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice's SIP phone” <sip:sipphone@192.1.0.200;transport=udp>
<SDP payload not shown>
```

After the PEP decides on where to forward the request, the associated Service Controller queries the Location Service about the IP address of the service. Provided that it gets a positive reply it forwards the request to the service changing the Request-URI as shown below:

```
INVITE sip:bob@192.10.0.150 SIP/2.0
Via: SIP/2.0/UDP 192.1.0.200;rport
CSeq: 2390 INVITE
To: <sip:bob@company.com>
Content-Type: application/sdp
From: “Alice's SIP phone” <sip:sipphone@alicesnetwork>
Call-ID: 42243331@192.1.0.200
Content-Length: 153
Contact: “Alice's SIP phone” <sip:sipphone@192.1.0.200;transport=udp>
<SDP payload not shown>
```

As we can see, positioning the system's PEP within the SIP Service Manager introduces considerable complexity to its function. The gain of course is that the user is now in full control of how his/her associated services can be accessed and used. This type of a SSM is essentially a standalone SIP provider of its own with the difference that it relies on another provider to connect to the SIP backbone. The architecture's benefits are listed below:

- This solution includes all the benefits that the SSM without the PEP functionality has, though, as we mentioned, it is not as simple. If the user wishes to avoid routing of certain calls he can still configure the SSM to issue policies via CPL.

- The system proposed makes minimal assumptions about the provider. No special services are required from the provider other than simple SIP routing. This means that the system is totally independent of the provider from a policy management perspective. Of course, the user still relies on the SIP backbone to handle his/her outgoing calls as desired.

- Each user is represented in the SIP backbone by a single, well-known URI (the AOR), independent of the UA application he uses, provided that the UA application is used as a private service of course. Thus, we can say that everything within the private service network is seen as a single UA from the outside world.

This type of SIP Service Manager operation, although powerful, has a number of drawbacks. Apart from its relative complexity, the SSM in this case is a single point of failure. If the SSM is down for some reason, the private services lose SIP connectivity to the backbone. In the first architecture we described, this is not the case as all services remain accessible (although the user will need a quick solution to cancel an annoying policy from his/her provider's proxy). For this reason, unless there is some type of system redundancy scheme (e.g. another SSM takes over when the other is down), positioning the SSM with PEP functionality on unreliable nodes (i.e. nodes that are likely to go down or lose connectivity) should be avoided.

4.2.4 Monitoring services

Monitoring in both SIP Service Manager types is performed via SIP using the OPTIONS request sent by each of the Service Controllers to the associated service (see Figures and). As mentioned, an OPTIONS request must produce the same response as an INVITE request. Therefore, the response to the OPTIONS request should give information about:

- The operational status of the service (up, down, busy, authentication requirements, and so on).

- The methods that can be processed by the UA through the *Allow* header.

- The media type of service. This information is contained in the SDP payload that is attached to the response.

In addition, in [33] there is a proposal that a UA can convey information about its capabilities through the *Contact* header field. It is also noted that the UA should include this information when responding to an OPTIONS request. However, [33] is still an Internet Draft and this function is not supported (as far as we know) from User Agents yet. Furthermore, similar extensions maybe proposed so that the UA can convey even more information such as context information which might be very useful for implementing advanced, context-aware services. In the latter case the UA might include a Context Data eXchange Protocol (CDXP) ([34]) payload in a response to the OPTIONS request.

Using the information mentioned above, the Service Controller can determine the status of the associated service. A Service Controller may probe the service at regular intervals or after a request made by the SSM, when a new message is received from the SIP backbone. The latter mechanism, although it makes better utilization of the available bandwidth, might considerably decrease the speed by which a routing decision is made by the SSM. Especially in the case when the service is not operating, the Service Controller will have to wait for the OPTIONS request to time out before it determines that the service is down, a procedure which in normal cases takes about 15 – 20 seconds. On the other hand, monitoring a service at regular intervals

obviously causes extra traffic to and from the SSM. A typical uncompressed OPTIONS request is 535 bytes whereas a 200 OK response is around 700 bytes (depending on the headers used and the additional payloads). Therefore, if an SSM monitors one service with OPTIONS requests every 10 seconds, the total traffic generated on a daily basis is around 10 MB, which is unacceptable if the cost of the link is based on the traffic generated (i. e. the user is charged per byte). For this reason, care must be taken when the SSM monitors services using such links. Monitoring in these cases must be performed at larger polling intervals or it must be disabled completely, and the user may provide status information to the Service Controller manually (i.e. by configuration).

4.3 Usage scenarios

In this section, we investigate ways in which the two SSM types described previously in this chapter can be used, so that we give a more practical approach to our design.

4.3.1 SSM without PEP on a portable device

As we saw, the SSM without PEP is a very simple and lightweight SIP UA. Therefore, it is suitable to use it on a portable device. Furthermore, the SSM in this case is not a single point of failure: if the portable device is down, it does not cause harm to the service network and services can still be accessed and used.

For example, suppose that Bob, running a SSM on his PDA arrives at his company. Using a discovery mechanism, he discovers two fax machines that are SIP enabled and receives this information on his PDA. However, by the time they are discovered he observes (using the monitoring functionality of the SSM) that they are both busy and therefore not currently usable. Later his SSM reports that they are available and uploads a CPL script to Bob's proxy directing fax calls to the company's fax machines. In general, our user can upload policies according to the status of the network reported to him by the SSM.

4.3.2 SSM with PEP on a non-portable device

As we mentioned previously, the SSM with PEP functionality can be viewed as a standalone SIP provider. Its increased policy management capability can be desirable by users that have high demands on policy management such as routing per media type, which is not currently supported by CPL. In these cases a user can use this type of SSM in the same way that simple answering machines are used today. The SSM can be positioned at home and be connected to the wired network, directing calls according to the user's commands. This scenario will also require that the SSM can be managed remotely, for example via a Web-based interface.

Furthermore, this type of SSM can be even more interesting for groups of users, such as companies. A company may have numerous SIP User Agents such as phones, videophones, fax machines, User Agents that run on cameras, instant messengers, etc. Delegating routing or

management of all these devices to the SIP provider might not be desirable or efficient. Therefore, a locally located system such as the SSM is more suitable. In this case the SSM's operation resembles that of a Private Branch Exchange (PBX).

5. IMPLEMENTATION AND RESULTS

In this chapter we present our implementation of the SIP Service Manager (SSM) with Policy Enforcement Point (PEP) functionality. The main goal of our implementation is to prove that such architectures can be realized and to identify the practical problems that arise when implementing these architectures.

5.1 Goals of the implementation

So far we have taken a theoretical approach to user-side service policy management. However, in practice there can be numerous interesting issues that might arise such as timer manipulation, calculation of SIP transaction IDs, header modification, and so on. For this reason, we proceeded with our research by implementing parts of the architectures that we discussed in the previous chapter. In our implementation we focused on the second of the proposed architectures, the SIP Service Manager with Policy Enforcement Point functionality. The reason for our focus on is this architecture is that it includes features that require practical study, especially the SIP URI multiplexing and de-multiplexing function described in paragraph 4.2.3.

The main goal of the implementation is to prove that the architecture described above can exist and operate in practice (i.e., a proof of concept). Moreover, the implementation aims to identify practical issues and problems that need to be taken into consideration when implementing such architectures.

On the other hand, parts that were not of major importance for our study were excluded from our implementation. These parts include system security and service discovery. Although performance is also not a part of our main focus, we attempt to estimate the resource requirements, especially in memory and processing power, of such a system.

5.2 Methods and tools

We chose to implement the SIP Service Manager with PEP functionality using the Java programming language and more specifically the Java 2 Standard Edition (J2SE). The reason for this choice is that Java provides a means to quickly develop programs that work and it suits situations where performance is not the highest priority. Furthermore, the platform portability feature of the language makes porting the code to various platforms easy. Finally, there is an open source Application Programming Interface (API) for SIP, developed by the National Institute of Standards and Technology (NIST) [35], that was available for free and written in Java. This API is well documented and easy to use, while it provides ways for low level SIP message manipulation and forwarding.

To perform our tests, we used various tools such as SIP soft phones, hardware phones, and also provider-side entities such as registrars and proxies. Our tests were performed using *KPhone* (versions 3.14 and 4.01), *X-lite* (version 2.0) and *linphone* (version0.12.0) User Agents as well

as the a Cisco SIP phone 7960 with SIP software installed (firmware version P0S30200). *KPhone* is an open-source SIP User Agent for the *K Desktop Environment* (*KDE*) of the *Linux* operating system developed by *Wirlab* [36] and acts as a telephone and videophone. *X-lite* is a free SIP soft phone for *Windows*, *Windows CE,* and *MacOS* operating systems and it is developed by *Xten* [37]. Finally, *linphone* is another open source SIP soft phone for the GNOME desktop environment developed by Simon Morlat, Aymeric Moizard, and Sharath K. Udupa and is available at [38].

For the provider side functions (registrars, proxies, etc.) we used *VOCAL*, which is an open source SIP server system that includes proxies and a registrar developed by *Vovida* [39] and a very simple registrar-proxy-presence server, developed as open source software by NIST [35].

5.3 Software design

Figure 5.1 illustrates the structure of the Java code. As we can see, many of the SIP Service Manager components that we described in the previous chapter, such as the SIP Registrar and the Service Controller are also Java objects. Note that .java files listed in italics are interfaces and not objects. Some of these parts belonged to NIST's registrar-proxy-presence server and were added to our implementation with a few modifications. One of the features that we focused our attention on was code reusability: each of the important components can be used independently or in other applications without modifying the code they contain. In this section we will provide an overview of the code for details the reader can refer to the HTML-based documentation or the code itself.

***sip* package:** This is the main package of our implementation. The only object contained here is the *RegistrationClient*, a simple User Agent Client (UAC) that is responsible for registering with a registrar.

***applications* package:** This package contains Java *main* objects that are used to start and test the corresponding component. Each of these objects is used to start and initialize the SIP stack and the corresponding component. The most complex of these is of course the *SSMTestApp* which is used to start the SIP Service Manager, which in turn initializes the needed components.

***location* package:** This package contains all objects related to the location functions of SIP applications. The *LocationServiceClient* is an interface that is used by applications to access the location service. In our implementation, the location service is a simple object, the *LocalLocationService,* that runs locally, within the SSM. A *Binding* is one of the entries of the *LocalLocationService* and maps URIs to a number of contact addresses, that contain the IP address of the resource. Finally, the *LocalLocationService* is able to maintain a number of *LocalLocationServiceListener* objects in order to notify them for special events.

***registrar* package:** The *registrar* package contains a single object, the *Registrar*. This object is capable of processing REGISTER requests and uses a *LocationServiceClient* to store location information to a location service.

***ssm* package:** The *ssm* package contains strictly SSM related objects. A *SIPService* contains

```
.
`-- se
    `-- kth
        `-- javax
            `-- sip
                |-- RegistrationClient.java
                |-- applications
                |   |-- RegisterClientTestApp.java
                |   |-- RegistrarTestApp.java
                |   |-- SSMScalabilityTestApp.java
                |   |-- SSMTestApp.java
                |   `-- ServiceControllerTestApp.java
                |-- location
                |   |-- Binding.java
                |   |-- BindingEmptyException.java
                |   |-- ExpiresTimerTooSmallException.java
                |   |-- LocalLocationService.java
                |   |-- LocalLocationServiceClient.java
                |   |-- LocalLocationServiceListener.java
                |   |-- LocationServiceClient.java
                |   `-- NoSuchURIException.java
                |-- registrar
                |   `-- Registrar.java
                |-- ssm
                |   |-- CouldNotDetermineIntendedDirectionException.java
                |   |-- CouldNotLocateServiceException.java
                |   |-- SIPService.java
                |   |-- SSM.java
                |   |-- SSMConfiguration.java
                |   |-- ServiceController.java
                |   `-- policy
                |       |-- Policy.java
                |       `-- SelectLastPolicy.java
                |-- test
                |   |-- DummyStartApp.java
                |   `-- DummyUAS.java
                `-- util
                    |-- HopImpl.java
                    |-- ProxyUtilities.java
                    |-- RequestForwarder.java
                    |-- RequestValidator.java
                    |-- ResponseForwarder.java
                    |-- SimpleRouter.java
                    `-- TransactionMap.java
```

Figure 5.1 - Java code structure

information about a service associated with the SSM. The *ServiceController* is responsible for monitoring and forwarding messages to its associated service as described in the previous chapter. Finally, the *SSM* object is the core object of a SIP Service Manager. It is responsible for receiving messages from the stack and passing them to the appropriate component.

***test* package:** The *test* package is used only for testing. It consists of the *DummyUAS* object which is a simple UAS that replies with a *200 OK* response to every request it receives. The *DummyStartApp* is a main object that starts up a *DummyUAS*. The use of these objects is explained in paragraph 5.4.3.

***policy* package:** This package contains incoming call policy related objects. When a new incoming request is received, the SSM activates an object that implements the *Policy* interface in order to decide which of the *ServiceController* objects (and eventually which of the associated services) will handle the request. For our implementation, we also implemented the *SelectLastPolicy* object, which is a very simple policy that selects the most recently added *ServiceController* to handle the new incoming request. More policies can be implemented in a similar manner, in this way forming a policy repository, as described in section 3.1.

***util* package:** The package contains various utility objects that are used by other objects for specific tasks, such as message forwarding and validation. Large parts of the code contained in these objects follows the coding of similar objects of the proxy developed by NIST. However, the objects we provide here were designed so that they can be as reusable as possible, without making assumptions about the object that uses them.

5.4 Testing and results

It should be noted that performing tests on such SIP architectures is not always as easy or straightforward as it may seem. There are various problems that frequently occur, having to do with how each developer and implementor interprets the SIP specification ([2] and related RFCs). For example, we mentioned that the Service Controllers perform monitoring using the OPTIONS method, which allows them to retrieve information about the media supported by the UA application. However, not all UAs respond the same way. For example, *X-lite* version 2.0 does not include a SDP payload with the *200 OK* message while *KPhone* does and therefore the media type can be detected. Other UAs do not even support the OPTIONS method (though the SIP specification states that the OPTIONS method is mandatory). Another example is the Cisco phone we used, when configuring the SIP proxy with a Fully Qualified Domain Name (for example sip.ourlab.edu), instead of registering with the proxy as phone@sip.ourlab.edu it first resolves the FQDN using DNS and then it tries to register as phone@192.0.0.10, which may cause some problems to the proxy when routing messages.

On the other hand, since the protocol is still under development from the IETF, there are various extensions being released frequently as new Internet Drafts (such as [33]). Some UAs and servers are quick to support them, but others are not. Finally, free SIP tools such as the ones we mentioned above have not been heavily tested or widely used and therefore software malfunctions (internal errors, crushes) are frequent. It would have been possible to try to detect and correct these errors, since many of them are open source. However, this is a time-consuming process, and outside the scope of our study. However, we believe that as the Session Initiation Protocol becomes more mature and widely used, these problems, both interoperability and software errors, will be resolved.

5.4.1 Multiplexing test

One of the major features of the SIP Service Manager is its multiplexing functionality, i.e. the ability to receive messages from the service network and changing appropriate headers to make them appear that they are generated from the SSM itself and, most importantly, identifying the responses and forwarding them back to the appropriate service. To perform this test, we use the simple setup shown in Figure 5.2. Note that the connections shown imply SIP connectivity. All the nodes are of course connected to each other at the IP level.

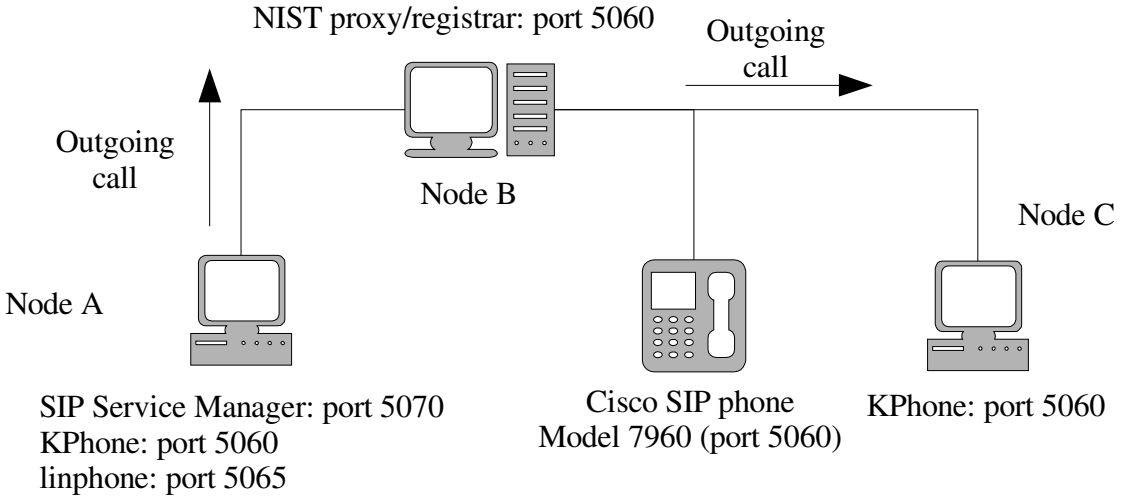

Figure 5.2 - Multiplexing test

This setup shows how two User Agent applications can run on the same machine (listening on different ports of course) and use a SIP Service Manager to connect to the SIP backbone with a single Address-Of-Record.

In this setup the UAs running on Node A are private services associated with the SIP Service Manager that runs on the same node. Therefore they register with the SSM on startup, and rely on it to forward the SIP messages. We perform the test by calling the Cisco IP phone and the *KPhone* on node C simultaneously. We observe that the SSM is capable of multiplexing their requests and de-multiplexing the responses.

Problems arise when forwarding BYE and ACK requests. In the SIP example we gave in section 2.1, the end-nodes sent these requests directly. Many UAs (including the ones we use in our setup) often use their proxy to forward these requests, something which is indeed allowed by the SIP specification. However, since it is not certain if the proxies will receive these requests or not (as they are not part of the corresponding INVITE transaction), the SSM has to create a new transaction when it receives them. The SSM must make sure that these requests are routed towards the right destination, which is the same as the destination as that of the INVITE message that created the call. We found that there are two ways to solve the problem: either the SSM must maintain a dialog state for the peer to peer communication, or forward the BYE or ACK request using the *Route* header that must be included in these requests. In our implementation we chose the second way, because it was simpler.

5.4.2 Policy enforcement test

The other main function of the SIP Service Manager is to apply policies to incoming calls and route them accordingly. In order to test the policy enforcement functionality we have implemented a very simple policy that always selects the more recently added associated service; provided that it is up (we can determine that using the monitoring function). The idea behind this policy is that the SSM assumes that the user will be closer to the more recently added service and will be able to answer the call. The setup for this test is shown in Figure 5.3.

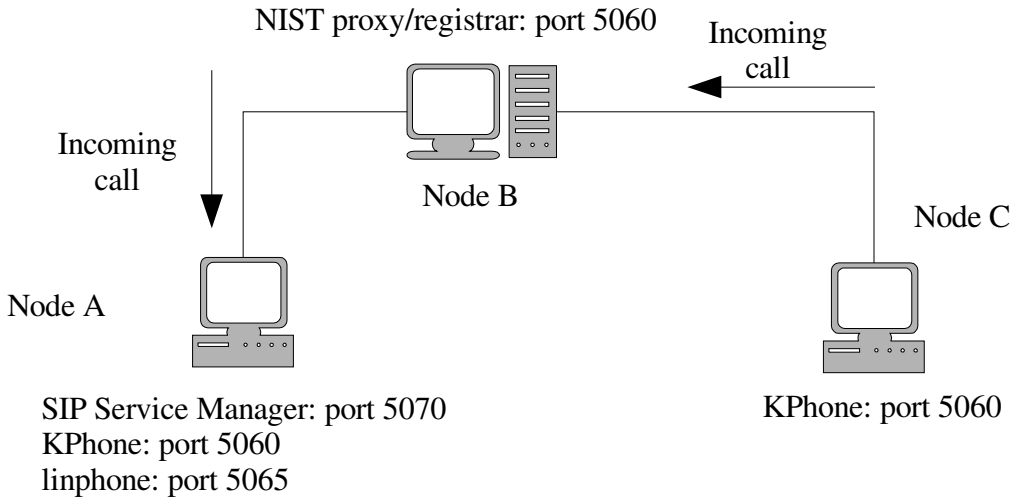

Figure 5.3 - Policy enforcement test

As we initiate the call from Node C we observe that the SSM always chooses the more recently added service. Of course more complex policies can be implemented simply by programming new objects that implement the *Policy* interface as we mentioned in the previous section.

5.4.3 Scalability tests

In paragraph 4.2.3 we mentioned that the SIP Service Manager with PEP functionality can be used by groups of people in companies or homes. In this case the number of managed User Agents can grow considerably. Although the performance of the application, since it is implementation dependent, is not our main focus, we will try to estimate the system's scalability in terms of memory by adding Service Controllers and observing their memory usage.

To do this, we use several *DummyUAS* objects running on one machine and listening on different ports. Then, we use our SSM to monitor these services sending OPTIONS requests at an interval of 10 seconds. Our results on memory usage are shown in Figure 5.4. The small squares represent our measurements while the line is a spline interpolation of those measurements. We observe that memory usage increases proportional to the number of services, something that we expected. We can also see that this increase is not a straight line, but fluctuates. This is because the Service Controllers contain timer threads that send the

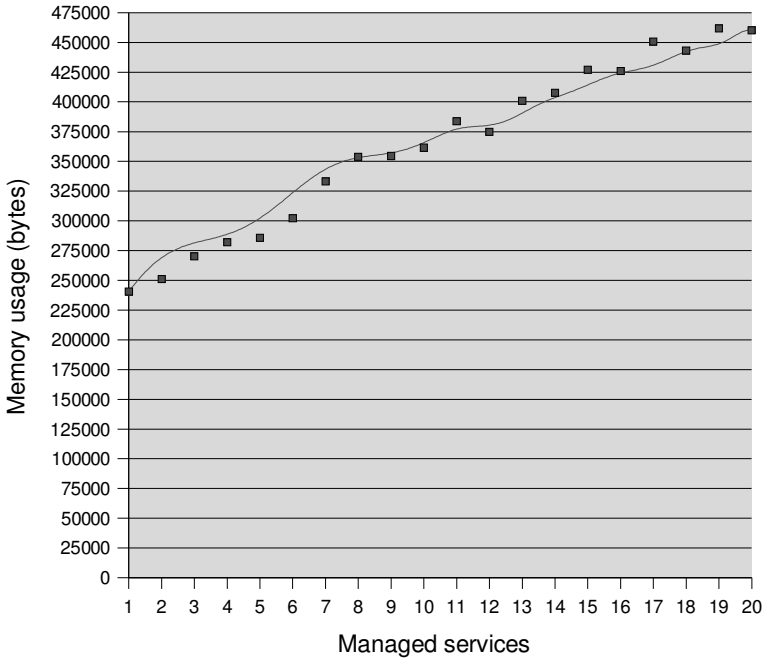

Figure 5.4 - Memory usage against the number of managed services

OPTIONS requests. If it happens that there are many Service Controllers transmitting or processing responses at the time of the measurement, then the memory usage is higher. However, we can observe that usage increases at a rate of 10.5 to 11 Kbytes per service. If the SSM is being used by a single user with 20 services, then it would require 470 Kbytes. If the SSM is used in a company of 500 employees with an average of 20 User Agents per user it would require 23500 Kbytes (22 MB). We can see that the memory demands are relatively low.

However, the proposed system can have scalability problems that are related to the processing power of the machine that the SSM is running on. To estimate this limit we will perform another test, measuring the number of OPTIONS requests and OK responses are sent and received by the SSM, as the number of monitored services increases. Note that the results of this test highly depend on the processing power of the machine the SSM is running on. For this reason we performed two tests using two different machines.

For the first test, we used a PC with Intel Pentium 4 at 1.7 GHz, and RAM memory of 256 MB. The operating system we used is Linux (kernel version 2.4) with Java Runtime Environment version 1.4.2_3, from Sun Microsystems. The monitoring interval in this test was again 10 seconds. Our measurements are shown in Figure 5.5. We observe that after the limit of 600 services the performance degrades dramatically. Due to thread starvation, the Service Controllers were not able to send the appropriate number of requests and the number of responses that are being processed by them is even less. Note that the CPU usage after that point was constantly 100 %.

For our second test we used another PC with Intel Pentium 4 at 2.8 GHz, and RAM memory of 3530 MB. As we see, the performance of the second machine is about 60% better than that of

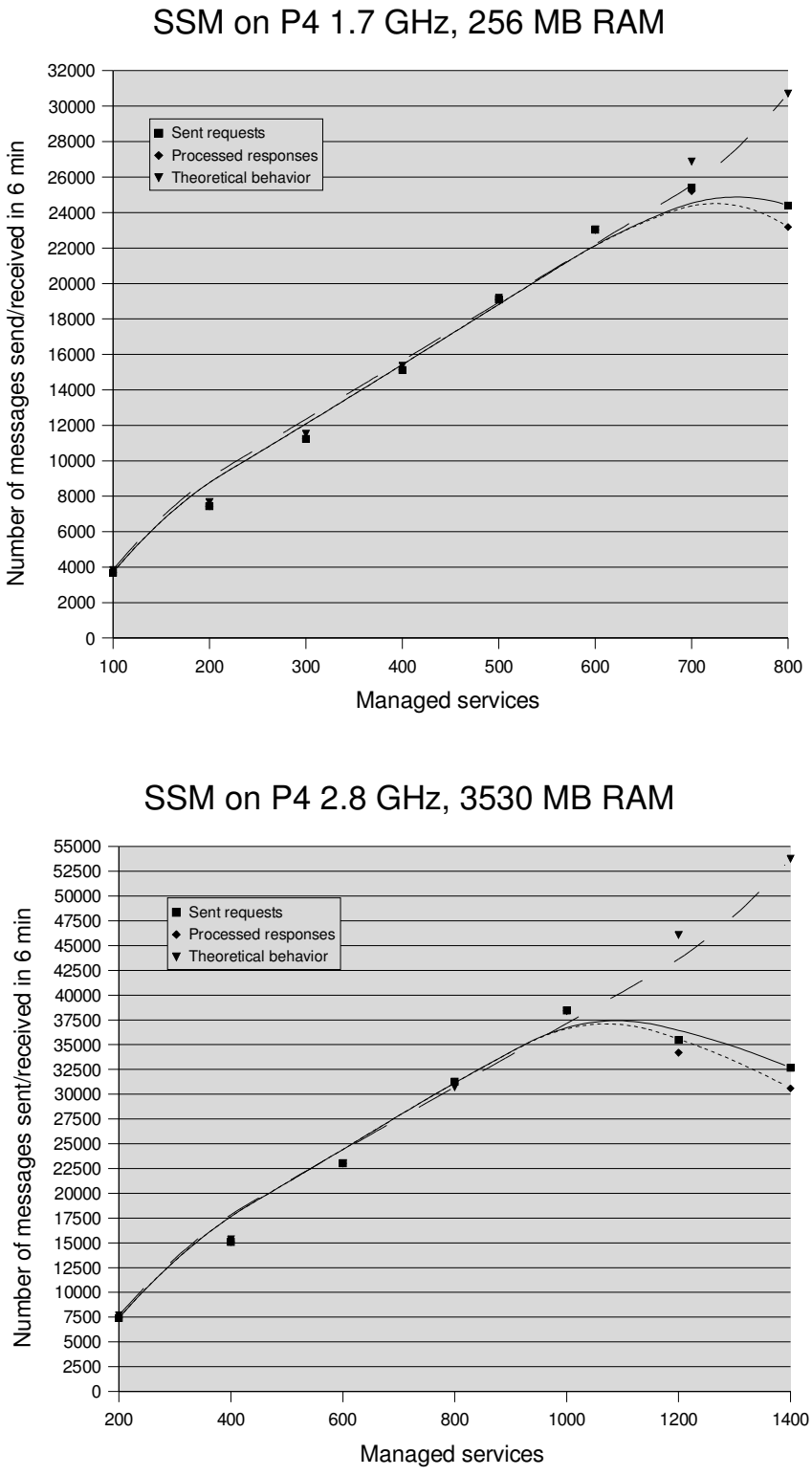

Figure 5.5 - Messages sent and received against the number of managed services

the first one, which is an expected result: The ratio of the managed services is approximately equal to the the ratio of the processor speed:

$$\frac{1000}{600} \approx \frac{2.8}{1.7}$$

From the scalability tests we conclude that the memory demands of the SSM we implemented are low, hence the system scales approximately linearly with processing power. This means that as the service network grows, the demands on processing will grow as well, and the machine where the SSM is running on will be short of resources. This is a typical behavior in management systems that consist of a single central management node that is polling the managed entities. For this reason, in the case where the SSM is used by groups of people (i. e. in companies) a hierarchical architecture will be required, where an SSM monitors other SSMs, which in turn monitor the services.

6. CONCLUSIONS AND FUTURE WORK

In this thesis we investigated means for user-side service policy management in SIP networks. We observed that advanced architectures are possible at the cost of complexity. In this last chapter we summarize our findings and suggest directions for future work.

6.1 Findings

Our initial assumption in this document was that the future user will need to use various types of services simultaneously, therefore, various User Agents will constitute the user's personal service network. The motivation for this thesis was to find ways so that this personal service network can be managed efficiently in order to improve the user's communication options and overall control of the system.

We proposed two architectures. Both of these architectures rely on a central entity, which we named SIP Service Manager (SSM). The main task of the SSM in both cases was to monitor the service network and issuing policies according to its status. In the first architecture, the system partially relies on the provider, using one of its proxies as a Policy Enforcement Point (PEP) and uploading policies to it, which are expressed using the Call Processing Language ([21] and [22]). In the second one the management system is standalone, as the SSM acts also as a PEP. The advantage of the first proposal is that it is simple and lightweight. However, the system in this case is bound to the limits of CPL for its policies and assumes that the provider supports CPL script uploads. The second architecture does not have these limitations. It also allows User Agents that do not have a Globally Routable UA URI (GRUU) (we referred to these as private services) to connect to the SIP backbone. However, the system in this case is more complex and the SSM in this case is a single point of failure since, if it malfunctions, private services cannot access the SIP backbone.

To prove our concept, we implemented our second and more complicated proposal. Based on this implementation, we found that architectures such as this can exist and operate in practice. In addition, the implementation substantially helped evaluate our proposed design. It also helped detect possible problems that arose, especially when the SSM acts as a proxy. Finally, our scalability tests showed that although the memory demands of the SSM are low the demands on the processing power can be quite high when service networks are large.

6.2 Future work

In paragraph 4.2.4 we mentioned that the monitors, using the OPTIONS request are able to detect the status of the managed service (i.e. if it is operating or not) along with a number of other attributes such as methods allowed, media types and CODECS supported (the latter information is contained in the SDP payload), and so on. We also mentioned that the SSM could take advantage of protocols that are able to express more complicated data, such as context data, and enforce policies that are context-aware rather than media-aware. In this case the SSM will

collect all context data and issue policies accordingly. To transport the data, managed User Agents can use the Context Data Exchange Protocol (CDXP) proposed in , either separately or as SIP payload like SDP. Therefore, future work can involve context-aware service policy management, where the PDP of the management system will be able to issue policies according to context data. Moreover, in the case where more advanced management is desired, the system could use specialized management protocols such as the Simple Network Management Protocol (SNMP) ([23] and related RFCs). In that case, management information will be structured as Management Information Bases (MIBs). However, It should be noted that the introduction of more protocols will increase the complexity of the system, as in that case both the SSM and all managed UAs must support them.

In addition, a natural continuation of the proposals we made in this thesis will be to combine the operation of the SSM with that of a service discovery method. The service discoverer will discover the available services and the SSM can handle their management.

In paragraph 4.2.4, we discussed the additional traffic generated by the monitoring function of the SSM. Further research can investigate possible **adaptive** monitoring schemes so that the generated traffic is reduced, while keeping the status information of the service network from growing stale.

In the beginning of chapter 3.3, we mentioned that security was an important issue when designing service policy management systems. However, in both our design and implementation we did not discuss security implications of such systems. Research in this regard should involve AAA, SDP message protection, protection from Denial-of-Service attacks, and so on.

The SSM can be viewed as a User Agent (and therefore as a service) of its own. Thus, it might be convenient if this special type of UA can be managed by other SSMs. The monitored SSM can in this case aggregate the information of its own service network and report it to the SSM that monitors it.

Moreover, in our first proposal for a SIP Service Manager, we claimed that CPL was too limited. Indeed, CPL currently is only able to process SIP and H.323 **headers**. In order to expand its functionality, there must be proposals that take other protocols into consideration so that more complicated policies can be expressed and enforced. Using an extended CPL, one could implement our first proposal of the SSM that uses CPL, and compare it with the second.

Finally, since processing demands can grow fast in large service networks, future research may include a study on hierarchical architectures of SSMs, so that processing is distributed more evenly.

REFERENCES

[1] C. Perkins, "IP Mobility Support for IPv4", IETF, RFC 3220, August 2002.
[2] J.Rosenborg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M.Handley and E. Schooler, "SIP: Session Initiation Protocol", IETF, RFC 3261, June 2002.
[3] T. Berners-Lee, R. Fielding and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", IETF, RFC 2396, August 1998.
[4] R. Sparks, "The Session Initiation Protocol (SIP) Refer Method", IETF, RFC 3515, April 2003.
[5] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema and D. Gurle, "Session Initiation Protocol (SIP) Extension for Instant Messaging", IETF, RFC 3428, December 2002.
[6] J. Rosenberg, "The Session Initiation Protocol (SIP) UPDATE Method", IETF, RFC 3311, September 2002.
[7] S. Donovan, "The SIP INFO Method", IETF, RFC 2976, October 2000.
[8] J.Rosenberg and H. Schulzrinne, "Reliability of Provisional Responses in the Session Initiation Protocol (SIP)", IETF, RFC 3262, June 2002.
[9] A. B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification", IETF, RFC 3265, June 2002.
[10] M. Handley and V. Jacobson, "SDP: Session Description Protocol", IETF, RFC 2327, April 1998.
[11] H. Schulzrinne, "Dynamic Host Configuration Protocol (DHCP-for-IPv4) Option for Session Initiation Protocol (SIP) Servers", IETF, RFC 3361, August 2002.
[12] H. Schulzrinne, "Dynamic Host Configuration Protocol (DHCPv6) Options for Session Initiation Protocol (SIP) Servers", IETF, RFC 3319, July 2003.
[13] A. Gulbrandsen, P. Vixie and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", IETF, RFC 2782, February 2000.
[14] 3GPP home page, URL: http://www.3gpp.org accessed February 2004.
[15] 3GPP, "Service requirements for the Internet Protocol (IP) multimedia core network subsystem; Stage 1", TS 22.228 v6.5.0, January 2004.
[16] 3GPP, "IP Multimedia Subsystem (IMS); Stage 2", TS 23.228 v5.11.0, December 2003.
[17] 3GPP, "Internet Protocol (IP) multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3", TS 24.229 v5.7.0, December 2003.
[18] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan and A. Sastry, "The COPS (Common Open Policy Service) Protocol", IETF, RFC 2748, January 2000.
[19] 3GPP, "Virtual Home Environment/Open Service Access", TS 23.127 v6.0.0, December 2002.
[20] 3GPP, "Customised Applications for Mobile network Enhanced Logic (CAMEL) Phase 4", TS 23.078 v6.0.0, January 2004.
[21] J. Lennox and H. Schulzrinne, "Call Processing Language Framework and Requirements", IETF, RFC 2824, May 2000, expires February 2004.
[22] J. Lennox, X. Wu and H. Schulzrinne, "CPL: A Language for User Control of Internet Telephony Services", IETF, Internet Draft , August 2003, expired.
[23] D. Harrington, R. Presuhn and B. Wijnen, "An Architecture for Describing Simple Network Management Protocol (SNMP) Ma", IETF, RFC 3411, December 2002.
[24] C. Newman and J. G. Myers, "ACAP -- Application Configuration Access Protocol", IETF, RFC 2244, November 1997.
[25] M. Wahl, T. Howes and S. Kille, "Lightweight Directory Access Protocol (v3)", IETF,

RFC 2251, December 1997.
[26] S. Shepler, B. Callaghan, D. Robinson, R. Thurlow, C. Beame, M. Eisler and D. Noveck, "Network File System (NFS) version 4 Protocol", IETF, RFC 3530, April 2003.
[27] J. Rosenberg, H. Schulzrinne and P. Kyzivat, "Caller Preferences for the Session Initiation Protocol (SIP)", IETF, Internet Draft , October 22 2003, expires April 21 2004.
[28] H. Schulzrinne and J. Polk, "Communications Resource Priority for the Session Initiation Protocol (SIP)", IETF, Internet Draft , February 15 2004, expires August 15 2004.
[29] P. Asprino, A. Fresa, N. Gaito and M. Longo, "A layered architecture to manage complex multimedia services", Proceedings of the 15th International Conference on Software Engineering and Knowledge Engineering San Francisco Bay, USA, June 2003.
[30] R. Cascella, "Reconfigurable Application Networks through Peer Discovery and Handovers", Royal Institute of Technology (KTH), Master of Science Thesis , June 2003.
[31] K. Sollins, "The TFTP Protocol (Revision 2)", IETF, RFC 1350, July 1992.
[32] J. Rosenberg, "Obtaining and Using Globally Routable User Agent (UA) URIs (GRUU) in the Session Initiation Protocol (SIP)", IETF, Internet Draft , February 15 2004, expires August 15 2004.
[33] J. Rosenberg, H. Schulzrinne and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", IETF, Internet Draft , December 24 2003, expires June 23 2004.
[34] A. Wennlund, "Context-aware Wearable Device for Reconfigurable Application Networks", Royal Institute of Technology (KTH), Master of Science Thesis , April 29 2003.
[35] Projet IP telephony / VOIP, URL: http://snad.ncsl.nist.gov/proj/iptel/ accessed March 4 2004.
[36] WIRLAB - Network Research Center, URL: http://www.wirlab.net/ accessed March 4 2004.
[37] Xten.com, URL: http://www.xten.com/ accessed March 4 2004.
[38] linphone.org, URL: http://www.linphone.org/ accessed March 7 2004.
[39] Vovida.org, URL: http://www.vovida.org/ accessed March 4 2004.

APPENDICES

Appendix A – Abbreviations

AOR	Address-Of-Record
API	Application Programming Interface
AVP	Attribute Value Pair
CDXP	Context Data eXchange Protocol
CEPT	Conference for European Posts and Telecommunications
CODEC	Coder Decoder
COPS	Common Open Policy Service
CSCF	Call Session Control Function
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
EDGE	Enhanced Data Rates for GSM Evolution
FQDN	Fully Qualified Domain Name
GERAN	GSM/EDGE Radio Access Network
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
GRUU	Globally Routable UA URI
GSM	Global System for Mobile communications
GUI	Graphical User Interface
HSS	Home Subscriber Server
HTTP	Hyper Text Transfer Protocol
I-CSCF	Interrogating CSCF
IEEE	Institute of Electrical and Electronics Engineers
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISC	IMS Service Control
ISDN	Integrated Services Digital Network
ISUP	ISDN User Part
J2SE	Java 2 Standard Edition
KDE	K Desktop Environment
LAN	Local Area Network
MIB	Management Information Base
MIME	Multipurpose Internet Mail Extensions
NAT	Network Address Translator
NIST	National Institute of Standards and Technology
P-CSCF	Proxy CSCF
PBX	Private Branch Exchange
PC	Personal Computer

PCM	Pulse Code Modulation
PDA	Personal Digital Assistant
PDF	Policy Decision Function
PDP	Policy Decision Point, Packet Data Protocol
PSTN	Public Switched Telephony Network
QoS	Quality of Service
RAM	Random Access Memory
RFC	Request For Comments
RMI	Remote Method Invocation
RR	Resource Record
RTP	Real-Time Protocol
S-CSCF	Serving CSCF
SDP	Session Description Protocol
SDPng	SDP next generation
SIP	Session Invitation Protocol
SRV	DNS Resource Record used to locate services
SS7	Signaling System No. 7
SSM	SIP Service Manager
TFTP	Trivial File Transfer Protocol
THIG	Topology Hiding Inter-network Gateway
TLS	Transport Layer Security
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UMTS	Universal Mobile Telecommunications System
URI	Uniform Resource Identifier, Uniform Resource Indicator
UTRAN	UMTS Terrestrial Radio Access Network
WAP	Wireless Access Protocol
WLAN	Wireless LAN
XML	eXtensible Markup Language

Appendix B – List of Figures

Appendix C – SIP Headers

Header Field Name	*Description*
Accept	Specifies acceptable media types.
Accept-Encoding	Specifies acceptable content codings.
Accept-Language	Specifies preferred language for reason phrases, session descriptions or status responses.
Alert-Info	In INVITE requests, it specifies ring tone. In 180 Ringing responses, it specifies ringback tone.
Allow	Specifies which methods are supported by the sender.
Allow-Events	A list of *event packages* supported by the SIP entity.
Authentication-Info	The field is included in a 2xx Success response to indicate that the corresponding request was successfully authenticated using HTTP Digest.
Authorization	Specifies authentication credentials.
Call-ID	A string in the form of <any_string>@<host> that uniquely identifies an invitation or the registrations of a particular client.
Call-Info	Provides additional information on the caller (in requests) or on the callee (in responses).
Contact	Provides a set of URIs that indicate where a specific resource can be accessed.
Content-Disposition	Specifies how the message body or a message body part is to be interpreted by the receiver. This field extends the Content-Type header field
Content-Encoding	Indicates content codings applied to the message body.
Content-Language	Specifies natural language of intended audience.
Content-Length	Size of message body in decimal number of octets.
Content-Type	Indicates the media-type of the message body.
CSeq	A 32-bit unsigned integer sequence number along with a request method name used to distinguish a new request from a retransmission of an old request.
Date	Date and time.
Error-Info	Provides a pointer to additional information about the error status response
Event	Specifies an *event package* in SUBSCRIBE and NOTIFY requests and responses to such requests.
Expires	Indicates time after which a message or its content expires

Header Field Name	*Description*
From	A “display name” along with a URI indicating the initiator of the message.
In-Reply-To	A list of *Call-IDs* specifying the calls that the current call references or returns.
Max-Forwards	The maximum number of times a message can be forwarded by SIP proxies.
MIME-version	Specifies Multipurpose Internet Mail Extension (MIME) version
Min-Expires	Minimum refresh interval supported for soft state elements maintained by the server (such as a registration entry).
Organization	Conveys the name of the organization the sender of the message belongs to.
Path	Used in REGISTER requests to indicate a set of intermediate entities that need to be traversed in order to send requests to the registered UA.
Priority	Indicates the urgency of the request as perceived by the client.
Privacy	Specifies the type of privacy needed for a specific transaction.
Proxy-Authenticate	Part of a “407 Proxy Authentication Required” response that poses the authentication challenge to a client.
Proxy-Authorization	Client credentials used to authenticate a client to a proxy.
Proxy-Require	Specifies proxy related features that must be supported by a proxy.
Record-Route	This field is inserted by proxies in a request to force future requests of the same dialog to be routed through the proxy.
RAck	Included in PRACK requests to specify the provisional response that is being acknowledged.
Reason	In requests it indicates the reason of the request. In responses it is used to encapsulate the final status code to requests that required forking.
Record-Route	The Record-Route header field is inserted by proxies in a request to force future requests in the dialog to be routed through the proxy.
Reply-To	Specifies a URI at which a user accepts replies.
Require	This field is used by a UAC to inform a UAS about options that the UAC expects the UAS to support in order to process a request.
Retry-After	This field may be included in some 4xx, 5xx and 6xx responses. It specifies a decimal integer specifying the number of seconds the called party will be available again.
Route	Forces routing of a request through the listed set of proxies.
RSeq	Sequence number used in provisional responses (1xx) that need reliable transport. It identifies different provisional responses.
Security-Client	List of security mechanisms a SIP client is able to use.

Header Field Name	*Description*
Security-Server	List of security mechanisms a server can use.
Security-Verify	Part of SIP requests after a security agreement has been made. It equals the contents of the received *Security-Server* header field.
Server	Information about the UAS used to handle a request.
Service-Route	Added to responses by registrars to indicate the outbound proxy (and possible next hops) that the registered UA should use to access the registrar's domain.
Subject	Summary indicating the nature of a call.
Subscription-State	Used in NOTIFY requests to describe the state of a previously made subscription. Its values are “active”, “pending” and “terminated” along with other parameters.
Supported	Enumerates all supported extensions by the SIP entity.
Timestamp	Time at which a UAC sent a request to a UAS.
To	Destination (recipient) of a request.
Unsupported	Enumerates all unsupported extensions by the SIP entity
User-Agent	Information about the UAC software used.
Via	Contains the path that has been followed so far by a request. It indicates the path that should be followed in responses.
Warning	Used to provide additional information in a response.
WWW-Authenticate	Contains an authentication challenge.
Refer-To	Used in REFER requests. Contains a URL for reference.

Appendix D – 3GPP related SIP headers

Header Field Name	*Description*
P-Access-Network-Info	Provides information about the layer 2/layer 3 access network that connects the UA to the SIP backbone (e. g. an IMS).
P-Asserted-Identity	Added by proxies to indicate that the sender of the request is either trusted or that he/she has been successfully authenticated with this (usually SIP) URI.
P-Associated-URI	Used in 200 OK responses to a REGISTER request. Its value contains a list of URIs associated with the registered AOR.
P-Called-Party-ID	Added by a proxy to indicate the intended AOR that a certain request (usually an INVITE request) is targeted to.
P-Charging-Function-Addresses	Contains addresses of nodes that are able to accept charging information.
P-Charging-Vector	Used to convey charging related information, such as the globally unique IMS charging identifier (ICID) value.
P-Media-Authorization	Contains one or more media authorization tokens which are to be included in subsequent resource reservations for the media flows associated with the session.
P-Preferred-Identity	Identity (URI) that a UA intends to be authenticated as. This identity will be the the value of the P-Asserted-Identity header (see above) after the US is successfully authenticated.
P-Visited-Network-ID	Contains the identity of the UA's “visited” network, which is a network that the UA connects to but has no direct business relationship with it.

IMIT/LCN 2004-04

www.kth.se