SIP Extensions for the eXtensible Service Protocol

Author: Vahid Mosavat
Examinator: Prof. Gerald Q. Maguire
Advisors: Theo G. Kanter
Tom Rindborg
Carl Wilhelm Welin
Date: 30/06/2003

Abstract
The switched telephony network was designed for voice calls. Expansion of data-communication has lead to a wide range of experimentation to create new services.
Theses services take place outside the network. When adding new services we currently encounter problems due to limitations of the simple devices at end points.

Theo Kanter has proposed a new model to remove these limitations; this model is called "Adaptive Personal Mobile Communication". The model consists of several components in the application layer of ISO standard. This model is based on peer to peer connections and the purpose of this model is to move services from within the networks to end point devices and avoid using central servers within the network.

The Session Initiation Protocol (SIP) for establishing multimedia sessions allows us to move the point of integration for multimedia service integration out to the end-points.
This project concerns the implementing of a prototype of this model as an SIP extension along with it evaluation. SIP offer addressing, naming, and localization of resources in this project.

This report presents different design alternatives for XSP as an SIP extension, and the chosen model presents as a result of comparing of these design alternatives.

Preface

I worked at this thesis as a degree project in Ericsson Research, in Stockholm, during the period December 2001- July 2002.

I would like to express my sincere thanks to:
Prof. Gerald Maguire, for his kindness and all useful advices.
Dr. Theo Kanter, for his advices, and helping to starting this project.
Carl Wilhelm Welin, for his advise and all support at Ericsson.
Tom Rindborg, for his advices and all supports at Ericsson.
Bjarne Däcker, for all supports at Ericsson and to be a kindness chef at Ericsson.
Thomas Sjöland, for support with SICStus-prolog.
Erik Forslin, for many interesting discussions.
Hamid Reza Mizani, for the good opponent work.
Roberto Gioacchino Cascella, for interesting discussions.
A special thank to my wife and my family, who has always been supporting me.

Vahid Mosavat 30/06/2003

Table of contents

1. Introduction

The switched telephony network is the most dominate network in the personal communication world. This network was from the beginning designed for voice calls. To set up voice based communication sessions, we utilize a simple user interface at the end points, i.e. telephones.

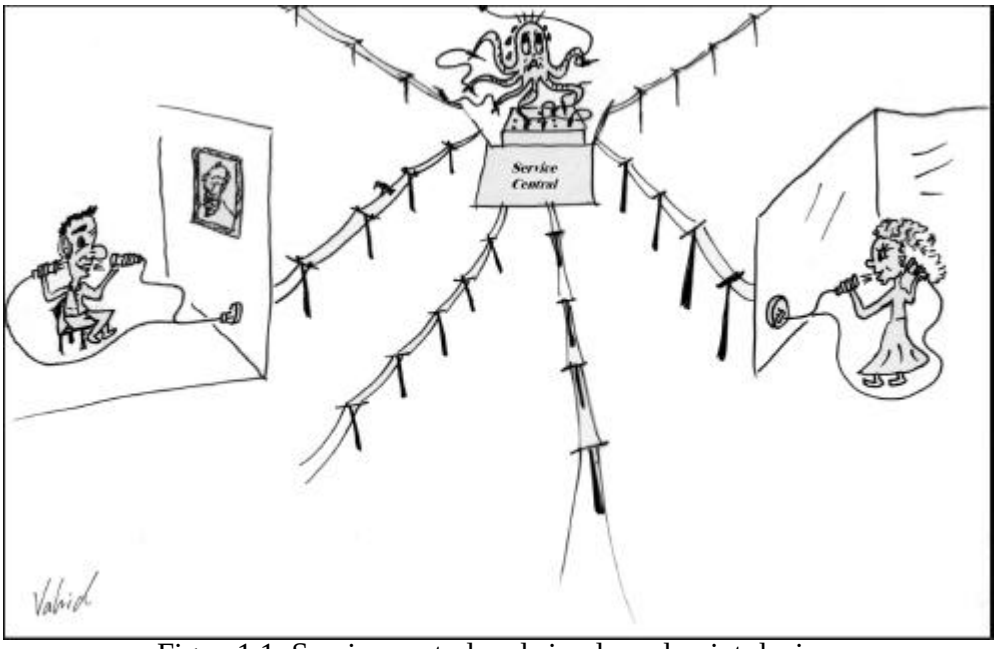


Figure1.1. Service central and simple end point devices

Concurrently the growth and expansion of data-communication and its convergence with telecommunication has lead to a wide range of experimentation to create new services. However, telephones were only designed for setting up and terminating calls and for this reason they are restricted to services of this type.

"Adding service functionality to these networks [switched telephony networks] can only be achieved with a separate entity that monitors events in the communication channels. The events that occur are principally the signaling to set up a call connection and the termination of the call. Telephony services in general, as implemented by these entities, simply speak in terms of call events."[APMC, pg. 1] To avoid this limitation, we may need to move the focus of control all the way from the network to the user interfaces. In this way the computing associated with providing services is removed from the network. In this new model, service functionality and computing are located at the communication devices at the end points and they do not need any computing services **within** the network.

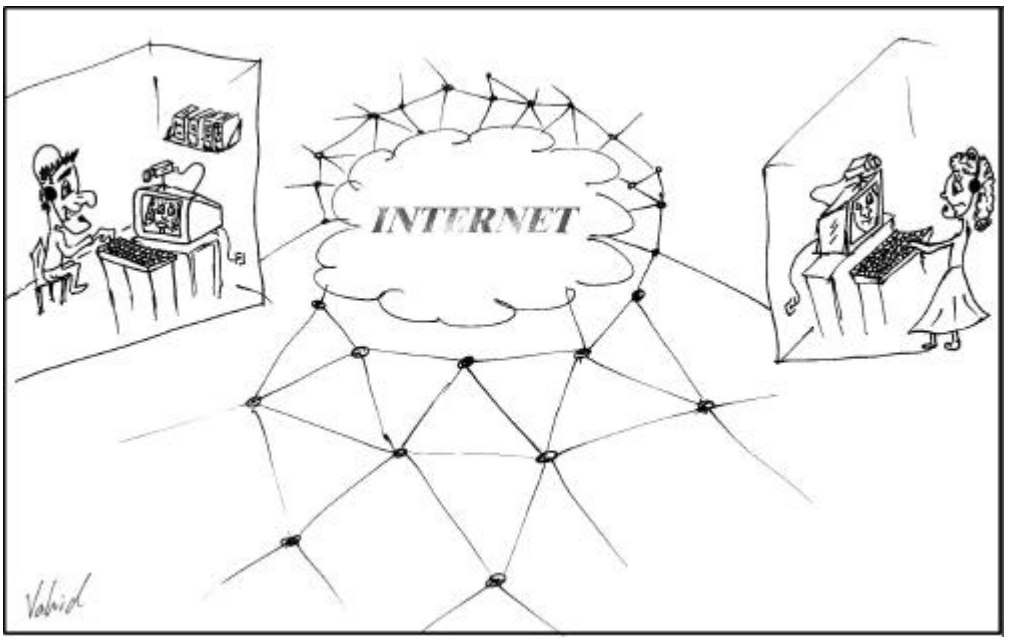


Figure1.2. Powerful end point devices, no computing within the network

This schema removes the limitations that restricted the types of services offered in switched telephony networks. “The enormous growth of new services that we have seen come about on the Internet during this decade were enabled by three properties that switched networks lack:

- End-to-end service transparency
- Multiple services per access link
- Intelligent and capable end devices

By adopting these properties and this new model, we are now able to fully exploit the computing power and the flexibility of user interfaces in the end-points.”[APMC, pg. 2]
Furthermore, this model will make use of the Internet Protocol to achieve its goal, which allows any host to provide a service to any other host. Hence, services should be moved out to end-devices.

1.1. Overview of the model

Theo Kanter has proposed a model to realize the above idea in his dissertation “Adaptive Personal Mobile Communication” [APMC]. This model consists of the following three main components: MSK, ACM, and XSP. This section introduces briefly the functionality of these components:

1.1.1. Mobile Service Knowledge (MSK):

MSK is an XML-based language for describing, managing, using, and exchanging service components in this model. By service components we mean here the description of objects in the communication world. Extensible Markup Language XML is a self describing markup language that is similar to HTML. [XML]

1.1.2. Active Context Memory (ACM):

ACM combines database functionality with some intelligence. The ACM stores MSK messages or descriptions and derives new conclusions based on the aggregated information.

1.1.3. Extensible Service Protocol (XSP):

XSP "allows users to ad hoc join in (or create) an infrastructure and automatically negotiate services with minimal user intervention and a minimum of shared service knowledge. No central server or control points are needed, as we use event routing to disseminate Mobile Service Knowledge. XSP is chiefly intended for negotiation of services, propagation of events, and coordination of MSK between the agents [by agent we mean here any software process which is able to exchange and share information with another instance of same agent]. XSP enables agents to discover & register with each other, provide (implicit) subscription to events, and exchange service profiles." [APMC, pg. 9] By the functionalities of ACM, MSK, and XSP, users can exchange knowledge, reach new conclusions based upon this knowledgebase and finally take action (do something) depending on these conclusions, an example of such an action is to send the conclusion to another user who can make use of this knowledge. By adding learning to the ACM we can have an affect on the communication via interaction with others.
In according with [APMC, pg.66] "a personal agent that runs on the (mobile) device uses co-located or integrated functionality: a SIP User Agent, the eXtensible Service Protocol and the Active Context Memory."

SIP (Session Initiation Protocol) is a network protocol defined at the application layer of the ISO's OSI model. The functionality of SIP is to setup and terminate a communication session, notice that SIP does not actually define the "media session", thus it is not necessary for SIP to know details about the media session. SIP makes use of another protocol for describing sessions; this protocol is the SDP (Session Description Protocol). SIP is not required to use SDP; in fact SIP is free to use any protocol that is able to describe a session. Therefore setting up, modifying, and terminating of communication sessions between the agents on the end-points will to be accomplished by SIP. In accordance with this model, SIP offers addressing, naming, localization of resources and carrying messages between agents.

The above was a concise description of the model, there are other components to this model, but they are not directly related to this thesis. Later in this paper we will give more detailed descriptions of SIP, XSP, MSK, and ACM.

1.2. Purpose of this project

The focus of this project was to implement and evaluate XSP as a SIP extension. The SIP User Agents will be co-located with a logically separate agent. The functionality of this agent is to take care of XSP messages, infer and manage of MSK messages and store them to its ACM. A complete implementation of XSP mechanism will not occur within this project, because of the size of this model. In addition there is not yet a complete description of the exact behavior of the agents.

1.2.1. Goals of this project

The goal of this thesis is to examine the most suitable direction for XSP and to clarify what "suitable" means in terms of explicitly stating the criteria. Subgoals are to implement and evaluate a simple prototype of the XSP mechanism, including straightforward inferring of Mobile Service Knowledge in the agent. An important boundary condition for the experiment is the absence of a priori and shared service knowledge between the parties involved.

1.3. Protocol stack

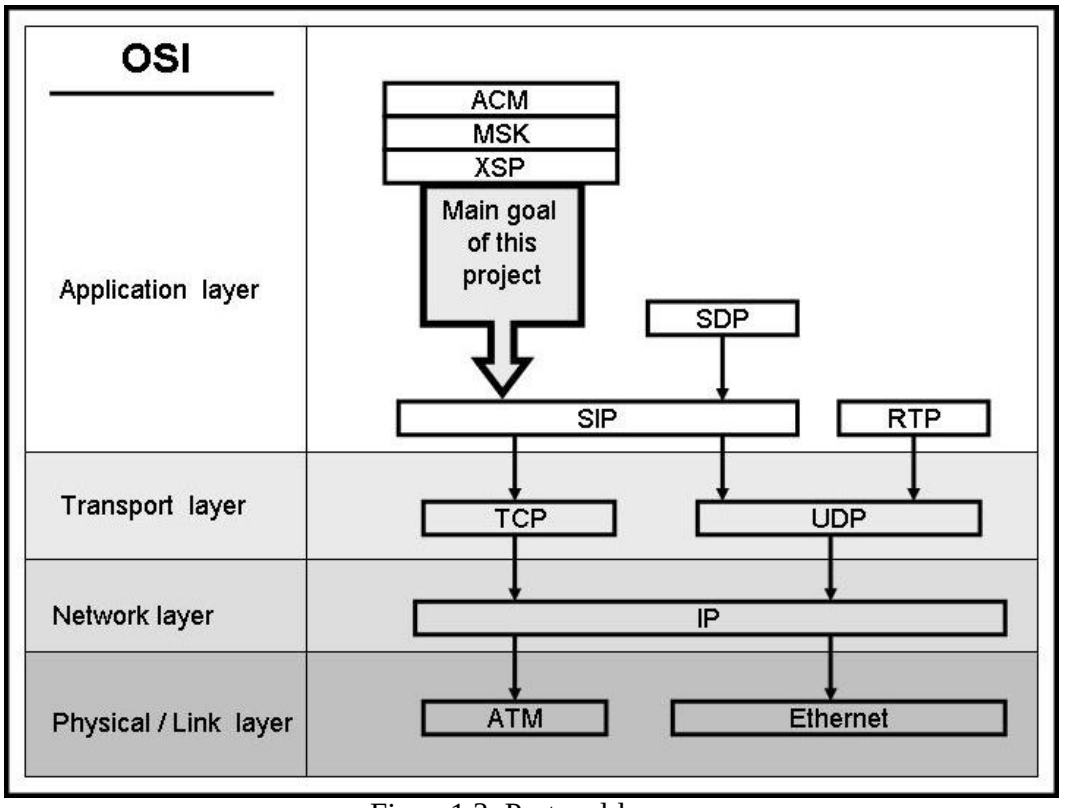


Figure1.3. Protocol layers

1.3.1. Transport layer

This layer includes several alternatives, including: User Datagram Protocol (UDP) and Transmission Control Protocol (TCP). SIP can use UDP, TCP, or some combination of them.

TCP

This protocol guaranties successful receipt of segments by using sequence numbers and acknowledgments. In addition TCP has a mechanism for repeating transmissions based on starting a timer each time a sender transmits a segment, if the timer expires before the segment is acknowledged the segment will be resent again.

UDP

This protocol doesn't guaranty receipt of packets. Hence most of complexity of TCP, such as sequence numbers and acknowledgments is skipped in this protocol. UDP only detect corrupted datagram based on the checksum.

Why use UDP when TCP is reliable?

UDP is faster than TCP, because UDP don't use sequence numbers and acknowledgments and therefore a lost packet will not be resent, this saves a lot of time and bandwidth. Hence UDP is most suitable for multimedia communication which values timely delivery above delivery of all packets. In multimedia communication the loss of some packets often does not mater as minimizing delay is more important factor.

1.3.2. Application Layer

The most relevant layer in this project is the application layer. This layer includes signaling protocols such as SIP, XSP, SDP (Session Description Protocol), FTP (File Transport Protocol), RTP (Real-time Transport Protocol), etc.

2. Background

In this section we will present several SIP components, and more details concerning XSP and MSK. Two scenarios make it easy to understand, both SIP's mechanism and how this model makes use of SIP. Following this is an introduction to the implemented components and the components related to this project.

2.1. SIP

This section is a concise description of SIP, based on the three following sources:
RFC 2543 Session Initiation Protocol [RFC 2543], Understanding Session Initiation Protocol [USIP], and finally The Session Initiation Protocol (SIP): A Key Component for Internet Telephony. [AKCIT]

2.1.1. History

SIP has its origins in late 1996 as a component of the "Mbone" set of utilities and protocols. The Mbone, or Multicast backbone, was an experimental multicast network overlaid on top of the public Internet. It was used for distribution of multimedia content, including talks and seminars and Internet Engineering Task Force (IETF) meetings. One of its main aims was creating a mechanism for inviting users to participate in multimedia session on the Internet. Thus the session initiation protocol (SIP) was born.

2.1.2. What does SIP do?

As the name implies, the session initiation protocol (SIP) is used to initiate communication sessions between users. Of course terminating and modification of sessions are a also part of SIP's functionality. SIP actually doesn't define what a "session" is. The description of session is often handled by another protocol: Session Description Protocol (SDP) defined by IETF.

Most of SIP is about the initiation part of the protocol, since this is really the most difficult aspect. "Initiating a session" requires determining where the user is to be contacted, i.e., where they are actually residing at a particular moment. Suppose that a user has a PC at work, a PC at home, and an IP desk phone in the lab, or a mobile device with him irrespective of physical location. A call for that user might need to ring all phones at once. Furthermore, the user might be mobile; one day at work, and the next day visiting a university. To find the user is a difficult problem which involves some type of registration service located somewhere on the network. However, this problem will be especially difficult when we talk about mobility, i.e., when the user has a mobile device with him and may be moving.

2.1.3. How does SIP work?

SIP is based on the request-response paradigm. To initiate a session, the caller (known as the User Agent Client, or UAC) sends a request (called an INVITE), addressed to the person the caller wants to talk to. In SIP, addresses are URLs. A URL (Uniform Resource Locator) is the address of a file or resource accessible on the Internet. SIP defines a URL format that is very similar to the popular "mailto" URL. If the user's e-mail address is uzr@host.se, his SIP URL would be sip:uzr@host.se.

The INVITE message is not sent directly to the called party, but rather to an entity known as a proxy server. The proxy server is responsible for routing and delivering messages to the called party. The called party then sends a response, accepting or rejecting the invitation, which is forwarded back through the same set of proxies, in reverse order.

Consider the following scenario, in which I will introduce some components defined in SIP (Figure2.1). Suppose user Vahid with SIP URL sip:vahid@kth.se would like to place a voice call to user Carl with SIP URL sip:carl@ericsson.se.

The SIP UAC which is running at Vahid's computer sends a SIP INVITE to the SIP proxy kth.se (1). This message is forwarded to the redirect server ericsson.se (2), the redirect server looks up this URL in its database and asks the proxy kth.se to directly contact the uab.ericsson.se(3). Now kth.se will send the INVITE messge from step 1 with the new incoming proxy URL sip:carl@uab.ericsson.se to uab.ericsson.se(4). Uab.ericsson.se looks up Carl in its SIP-register database and finds Carl is at bacardi.uab.ericsson.se. Notice that SIP defines another request, REGISTER which is used to inform a proxy of a user's current address binding.Here we assume that Carl sent a REGISTER request to uab.ericsson.se earlier. Uab.ericsson.se forwards the INVITE message to the final location of Carl (bacardi.uab.ericsson.se), this final URL looks like sip:carl@bacardi.uab.ericsson.se(5). Finally the INVITE reaches the User Agent Server of Carl that sends a response in answer to Vahid's invite request.

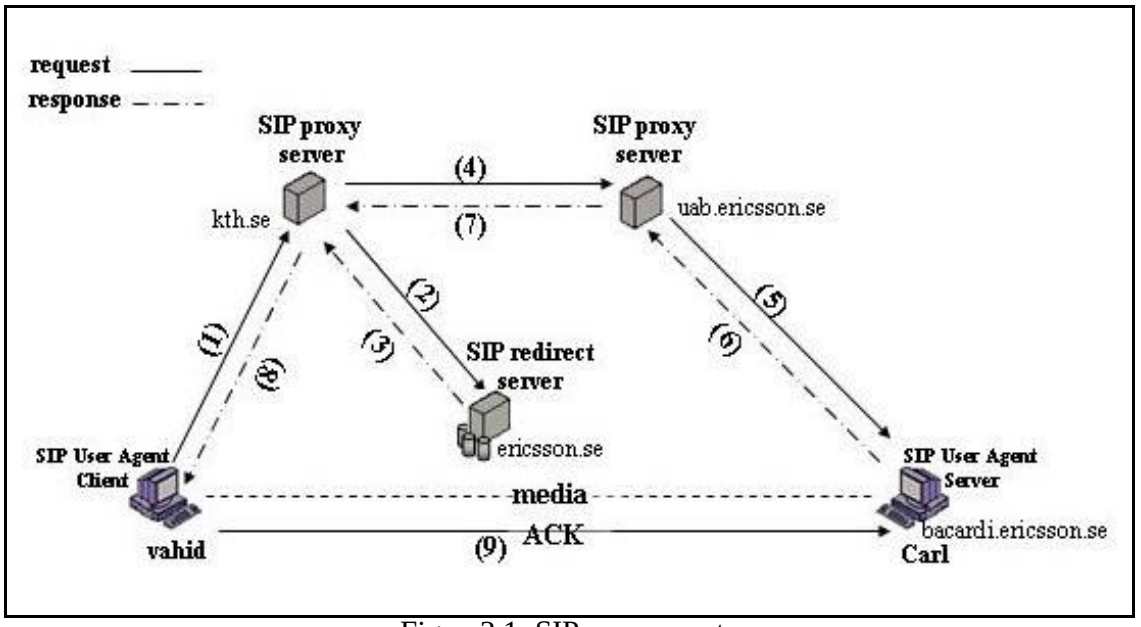


Figure2.1. SIP components

This response can be accepted, rejected, redirected, busy, can let an answering machine record the voice, and so on. (However, the case of an answering machine requires sending an accept response). We choose the case of acceptance here. The acceptance response is forwarded back through the proxies to the original caller Vahid (6, 7, 8). Finally before setting up the communication session an acknowledgement is sent directly from Vahid´s host to Carl's host (9), and at this point the voice session can start.

Notice that the above scenario is only a brief version of sending and receiving SIP messages between agents, I have tried to include only the most interested parts of these

messages, later in this paper I will describe in more detail these messages and their functionality.

The above scenario introduced the SIP proxy server, the SIP redirect server, the SIP User Agent Client, and finally the SIP User Agent Server and the functionality of each; additionally there are other components defined in RFC 2543, such as the SIP Location server and a SIP Registrar.

Proxy server	An application program which actually acts as both a server and a client. The proxy receives a request, if necessary, rewrites the request before forwarding (as was shown in the above scenario.)
Redirect server	A redirect server receives requests from clients and sends zero or more new addresses in response to the request sent by the client.
User agent (UA)	Consist of two parts: User agent client and user agent server.
User agent client (UAC)	The client part of SIP user agent. All SIP requests are initiated by this part of SIP UA.
User agent server (UAS)	This is another part of the UA. When a request is received by the UA, the UAS is this part which processes the request and sends a response back to the client, who initiated the request.

2.1.4. Main advantage

Services: Internet telephony began with the premise that it was cheaper than normal phone calls. Users were willing to tolerate degraded quality or reduced functionality for lower cost. However, the cost differentials are rapidly disappearing. To continue to exist, Internet telephony must find another reason to exist. The answer is new services. Some of the most exciting applications have already reached killer status on the Internet, though not (yet) in the form of multimedia services. Consider integrating multimedia communications, such as voice, with web, e-mail, buddy lists, instant messaging, and online games. Whole new sets of features, services, and applications become conceivable. SIP is ideally suited here. It's use of URLs, its support for MIME like carriage of arbitrary content (SIP can carry images, MP3s, etc), and its use of e-mail routing mechanisms, means that it can integrate well with these other applications.

Extensibility: History has taught Internet engineers that protocols get extended and used in ways they never intended (e-mail and HTTP are both excellent examples of this). So, they've learned to design in support for extensibility from the outset. SIP has numerous

mechanisms to support extensions. However, it does not require everyone to implement these extensions. Facilities are provided that allow two parties to determine the common set of capabilities, so that session initiation can always be completed, no matter what.

Flexibility: SIP is not a complete system for Internet telephony. It does not dictate architecture, usage patterns, or deployment scenario. It does not mandate how many servers there are, how they are connected, or where they reside. This leaves operators tremendous flexibility in how the protocol is used and deployed. One way to think of it is that SIP is a like LEGO block; operators can piece together a complete solution by combining it with other LEGO blocks, and putting them together in the way that they believe is best.

2.1.5. SIP service provisioning

Beyond SIP´s ability to create new services, these services can be created by service providers, enterprise administrators, and IT departments, or even directly by the end users. By opening up innovation to the public at large, all sorts of new services and features can be developed, potentially creating entire new markets. In overcoming the highly centralized and carrier-controlled model of telephony, and in putting tools for service creation in so many hands, SIP has the potential to deliver what the PSTN's (public switched telephone network) Intelligent Network only promised.

What kind of services or applications could be enabled by SIP? Besides the traditional call-forwarding, follow-me, and do-not-disturb, SIP has the potential for enabling a whole new class of services that integrate multimedia with web, e-mail, instant messaging and presence. One of he value that the Internet brings to Internet telephony is the suite of existing applications that can be merged with voice and video communications.

2.2. MSK

MSK is a machine interpretable description of the world as observed by a virtual object, or user. Thus, MSK fulfills a dual purpose of on one-hand describing the (observed) objects in the world, the relationships of the observed entities in it, including the entity that carries this MSK and on the other hand constituting the behavior of this entity in combination with the reasoning in the Active Context Memory (ACM).

The specific purpose of MSK is to capture knowledge and thus empower user's agents (and thus the users themselves) to make intelligent decisions regarding use of local communication resources, modes of user operation, as well as interaction with services belonging to other entities.

Kanter has purposed using Resource Description Format (RDF) for MSK [APMC]. The Resource Description Framework (RDF) is a general framework for how to describe any Internet resource such as a web site and its content. "The format of Mobile Service Knowledge (MSK) can thus be provided in a short form Resource Description Format

(RDF) and internally adheres to the logical notation of the object oriented programming in SICStus Prolog. Objects, i.e., the entities, can reference specifications of remote entities and relations."[APMC, pg.80] However, for simplicity we don't further examine RDF in this project and only use SICStus prolog as a database and an inference engine.

2.3. XSP

XSP is a language for agents, this is similar to KQML (Knowledge Query and Manipulation Language). KQML is a language and protocol for exchanging information and knowledge.[KQML] It is part of a larger effort, the ARPA Knowledge Sharing Effort which aims to develop techniques and methodology for building large-scale knowledge bases which are sharable and reusable. KQML can be used as a language for interaction between agents. This language is often used in an intelligent system or between two or more intelligent systems to reach a cooperative problem solving.

According to [APMC] XSP is simple, but more powerful than KQML; because XSP has functionality for inheriting of capabilities and accumulating new service knowledge depending on prior knowledge or service behavior. However, I consider these functionalities belong to ACM and that it is more correct to say "the model is more powerful". XSP sends messages between agents on a peer-to-peer basis.

2.4. How these components work together

We begin this section with a very simple scenario. All devices in this scenario are equipped with a SIP UA and all SIP UAs are co-located with an XSP mechanism. In addition, we assume that the discovery phase has already been completed, so that all agents know about the XSP capability of the other agents.

There are four users, we call them **A**, **B**, **C**, and **D**. (Figures 2.2. and 2.3.)
User **A** has information **Im**, user **B** has information **In,** and user **C** has **Io** in their respective databases. **I*x*** can be information about resources, where ***x*** is a resource, an agent, a file, a service, or relation between two objects.

In addition **A** has two following rules:

if **In** and **Io,** then **Ip**.
if **Ip** and **Im,** then **Iq**.

These rules have been read from an init file when the SIP client started.

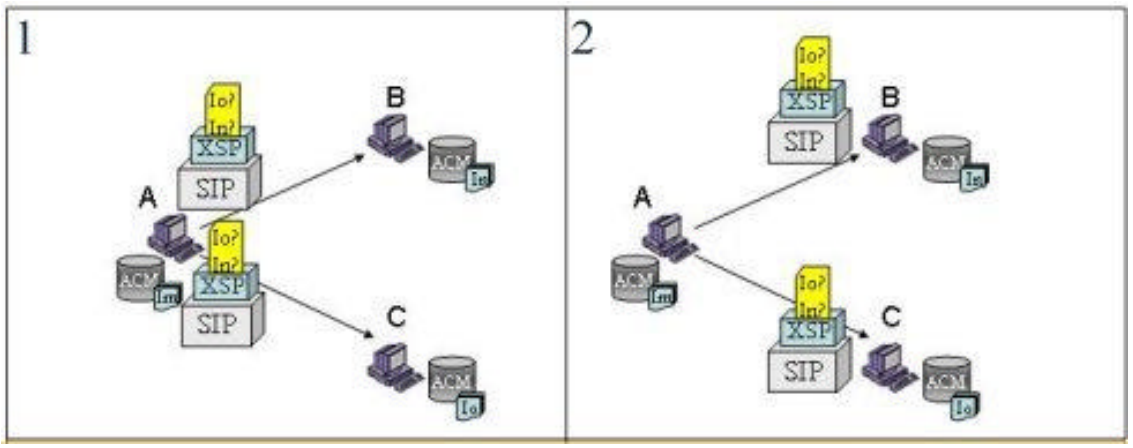


Figure 2.2 Simple scenario of this model

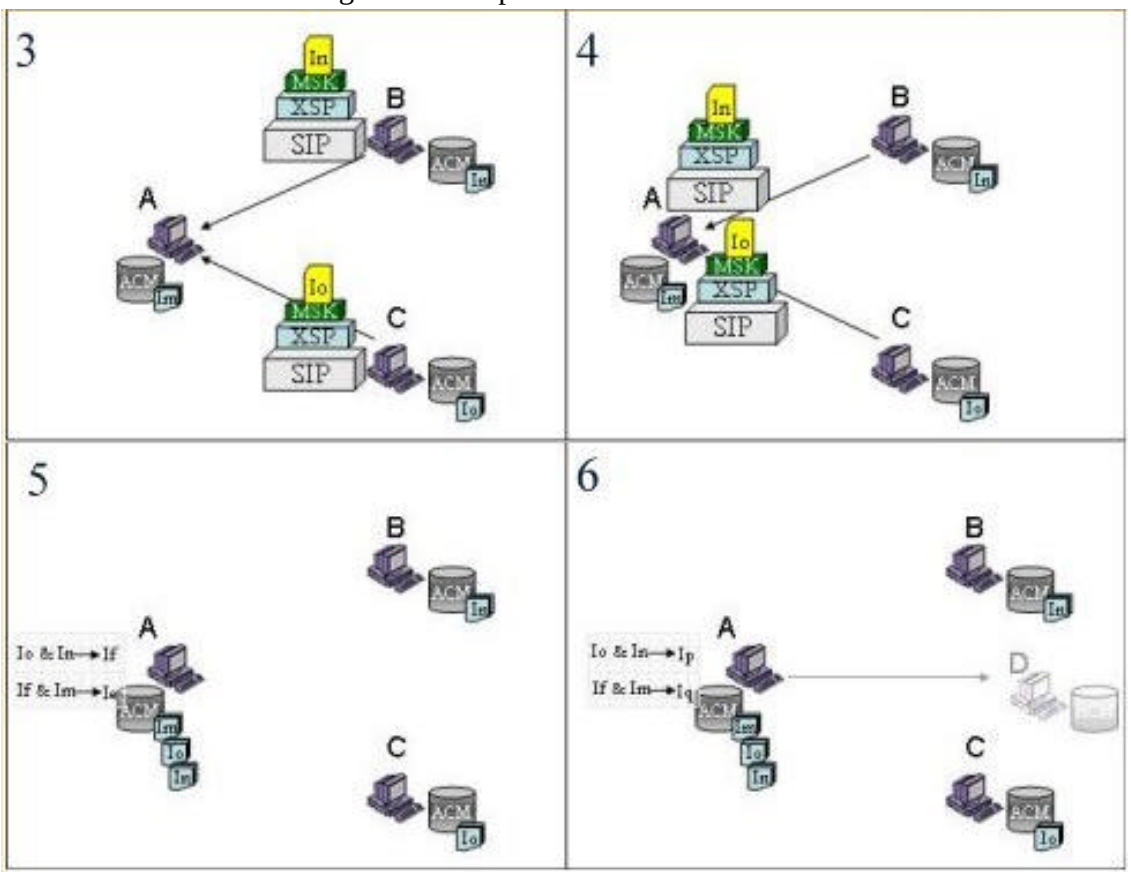


Figure2.3. Simple scenario of this model

Agent **A** asks agent **C** for any information about resources **n** and **o**. **C** sends **Io** to **A**, at this point **A** stores **Io** in its ACM. **A** asks agent **B** same question, this results in **B** sending

In to **A**. Notice that **A** asks the question concurrently, this means **A** does not wait for a response from **C** before it asks the question of **B**. Now **A** has enough information to conclude **Ip** (rule 1). Now, another agent **D** might ask **A** for any information about **q**, thus **A** is capable to send **Iq** to **D**.

Although this scenario was very simple, by replacing or adding more complicated rules we may handle very complex situations.

Which roles do XSP and MSK have in this scenario? Since all of **Ix** are MSK because they are descriptions of some resource. XSP is the instruction or information by which agents understand what they must do with the MSK part of message, so that storing of **Io** and **In** is a result of an XSP message. XSP messages in this scenario can be passed by using SIP and in this project we will investigate which SIP-extension is most suitable given the aims of XSP.

For simplicity we suppose that the ACM is the brain of the agent, and MSK is the structure which this agent uses to describe and understand a description of its environment and the objects in the system. XSP is the language which the agents speak, and finally SIP provides the medium that transports the language to other agents. Figure 2.4 shows the above.

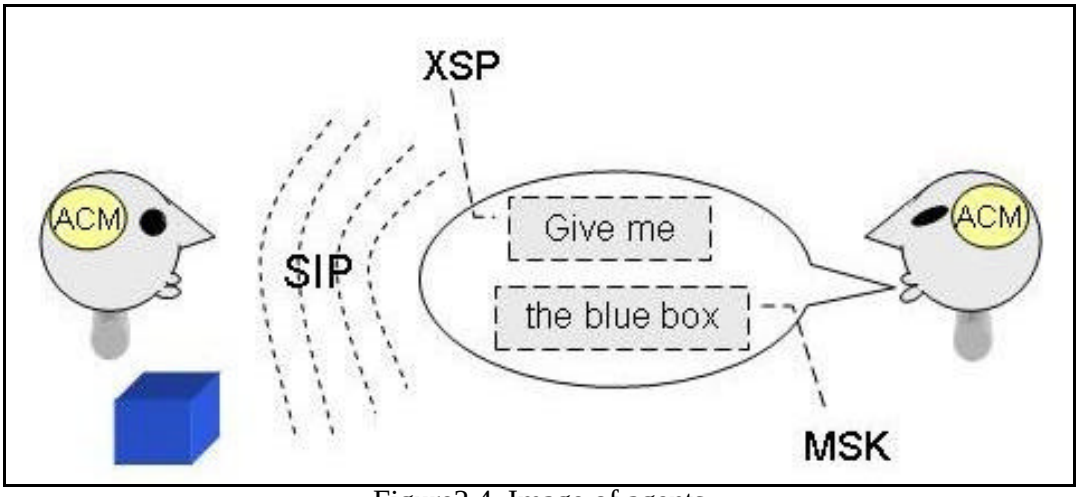


Figure2.4. Image of agents

2.5 How have others used SIP extensions?

Because the SIP-UA I used was implemented in java by Carl Wilhelm Welin, this project was also done in java. However, other SIP components may be implemented in other programming languages even those without an interface to Java. However, in this project we do not require other SIP-components.

The SIP-UA currently only uses UDP, as the TCP part was not yet implemented. In addition to SIP's standard methods Carl has implemented other methods such as SIP INFO, simple presence (SUBSCRIBE-NOTIFY), and finally a SIP extension for instant messaging. The purpose of the SIP INFO method is to carrying control information during a session. [RFC2976] The purpose of the simple presence functionality is to use SIP for subscription and notification of user presence, here "presence" means willingness and ability to communicate with other users on the network. [D-ISP] The aim of instant messaging is to transfer messages between users in near real-time. [D-ISEIM]
Later in this project we will compare the above methods and discus which of them is most suitable given the aims of XSP.

The other major component of this project is SICStus-prolog [SICS], we make use of this language to store MSK messages and reason about the stored information. Because MSK describes objects and the relation(s) between objects, we use Prolog. In addition, we make use of Jasper [SICSJ] an interface between SICStus-prolog and java, which makes it easy to using Prolog functionality from java.

The finally component we have made use of in this project is jaxb [JAXB] (java architecture for XML binding). We use it to parse XSP messages, which are in XML form. Jaxb generates java object from XML and regenerates XML from java objects.

3. Method

In this section we study SIP architectural model and guidelines for SIP extensions. We will look into how SIP is constructed and how a SIP extension could be designed and what makes a good SIP extension. In addition we will discuss the behavior of XSP with respect to this.

Generally there are several ways to accomplish the aims of XSP: XSP may be a separate protocol, XSP may be implemented as a SIP extension, or XSP could use some of SIP's functionality whiteout being a SIP extension. The aim of this project is to implement XSP as an SIP extension, but it is important to study how the behavior of XSP conforms to the rules for a SIP extension.

3.1. Guidelines for SIP extensions

This section summarizes of the internet draft "Guidelines for Author of Extensions to SIP" [GAES]. The Session Initiation Protocol (SIP) is a flexible protocol. This flexibility makes it easy to design new extensions to SIP and therefore it is important to be careful that extensions are well defined and strictly follow the rules for extensions. To determine whether a SIP extension is an appropriate solution to our problem, we will examine XSP with regard to two broad criteria: first the fitting of behavior of XSP into the general provisions of SIP's solution space and secondly how this solution conforms to the general SIP architectural model in other ways.

3.1.1. SIP's Solution Space and XSP

The SIP protocol is for initiating, modifying, and terminating interactive media sessions. This process involves discovery of entities and services that can be communicated with, wherever they may be located, so that a description of the session can be delivered to the user.

When we design an extension of SIP, we must consider the following:
The primary purpose of SIP is a rendezvous function, to allow a request initiator to deliver a message to a recipient wherever they can be. Whereas, the primary purpose of XSP is negotiation of services between clients. Keeping SIP independent of the sessions, it initiates and terminates is fundamental. XSP is not dependent on a session. SIP is not a session management protocol or a conference control protocol, and thereby a SIP extension should not lead to management or control of sessions. SIP will not affect the particulars of the communications with the media session: Media transport, virtual microphone passing, and feedback on session quality are some example of aspects not to be supported. SIP is not a resource reservation protocol for sessions. (This is fundamentally because SIP is separate from the underlying session it establishes, and secondly the path of SIP messages is completely independent from the path that session packets may take. The path independence refers to paths within a provider's network, and the set of providers itself. For example, it is perfectly reasonable for a SIP message to traverse a completely different set of autonomous systems than the audio in a session SIP establishes.) SIP is not a general purpose transfer protocol. It is not meant to send large amounts of data unrelated to SIP's operation. The register step is one of XSP messages which involves exchange of "agent profiles", these profiles are not allowed to be a large amount of data, in addition these profiles are actually not related to SIP operations. SIP is a poor control protocol. It is not meant to be used for example to change a configuration parameter or to be more exactly to support "auto-configuration of resources" which is proposed in Theo Kanter's dissertation. The above describe some of unsuitableness of XSP as a SIP extension.

3.1.2 SIP Architectural Model

In terms of to the general SIP-architectural model, SIP can be extended in following ways:

1. New methods
2. New headers
3. New body types
4. New parameters for existing headers can be defined

In this section we will consider the influence of XSP on SIP and if XSP could be an extension of SIP. SIP is a request / response protocol. Many of the rules of operation in

SIP are based on general processing of requests and responses.Table **3.1** shows some of SIP standard methods divided in request and responses:

Table 3.1

SIP Request Messages	**SIP Response Messages**	
INVITE REGISTER BYE ACK OPTIONS INFO SUBSCRIBE NOTIFY …	**Informational**	
	Trying Ringing Success 200 OK …	100 180 200 …
	Client Error	
	Bad Request Unauthorized …	400 401 …
	Server Error	
	Server Internal Error Not Implemented …	500 501 …
	Global Error	
	Busy Everywhere Decline …	600 603 ….

Here we describe some of the primary architectural assumptions which underlay SIP. Extensions that violate these assumptions should be examined more carefully to determine their appropriateness for SIP. In this section we will study XSP and its violation of following assumptions:

Session independence:
Any extensions to SIP must consider the application of SIP to a variety of different session types. XSP is not a protocol which depends on a specific session type so there is no violation for session types.

SIP and Session Path Independence:
Extensions that only work under some assumption of overlap are not generally applicable to operation of SIP and should be scrutinize carefully.

Multi-provider and Multi-hop: SIP assumes that its messages will traverse the Internet. That is, SIP can work through multiple networks administered by different providers. It is

also assumed that SIP messages may traverse many hops (where each hop is a proxy). Extensions should not work only under the assumption of a single hop or single provider. There by the proposed discovery method in section 7 [APMC:104] “In order to find XSP enabled entities, multicast requests are sent on a well-known XSP-port.”

Usually multicast requests will not traverse outside the local network the request initiates from if the neighbor network is not configured special for this aim. Therefore multicasting is not a good idea to find other XSP enabled entities. However if XSP is used in a local area network this problem will not occur, but it will not work in the global Internet.

Proxies can ignore bodies
XSP is defined so that proxies don’t need to understand XSP methods.

Proxies don't need to understand the method
Extensions must not define new methods which must be understood by proxies. In our case proxies do **not** have to understanding XSP messages.

Heterogeneity is the norm
Extensions should not be defined which will only function if all devices support the extension. This will **not** affect XSP, because XSP only works with XSP capable SIP **user agents**.

4. Analysis
In this section we will determine the most suitable method for using SIP for XSP.
First we introduce the SIP architecture and SIP Messages, followed by a description of which messages are most suitable to use for XSP.

4.1 SIP standard messages
SIP messages are either requests or responses. Some of these were presented in table 3.1. SIP messages start with a start-line, one or more headers and eventually a message-body. The UAC part of a UA sends requests and receives responses, while the UAS part sends responses and receives requests.

4.1.1. Requests
A start-line in a request names a request-line that consists of Method, Request-URI and Version.

Method: there are 6 standard methods, REGISTER, INVITE, ACK, CANCEL, BYE, and OPTIONS; an extension of SIP could define new methods.
Request-URI: indicates the service or the user that this message is constructed for.
Version: indicates the SIP version in use.

A short description of each of these SIP methods follows:

INVITE, REGISTER, BYE, ACK, CANCEL, and OPTIONS methods are the original six methods. SUBSCRIBE and NOTIFY [RFC3265], INFO [RFC2976], and PRACK [RFC3262] are additional methods.

INVITE: used by user agents to establish media sessions to other user agents. Notice that the session is defines by another protocol such as RTP and not by SIP.

REGISTER: used by a user agent to notify a SIP network of its current IP address and the URLs for which it would like to receive calls.

BYE: used by user agents to terminate an established media session.

ACK: used to acknowledge INVITE requests.

CANCEL: used to terminate pending search and call attempts.

OPTIONS: used to query a user agent or server about its capabilities and discover its current availability.

INFO: used by a user agent to send call signaling information to another user agent with which it has an established media session.

PRACK: used to acknowledge receipt of reliably transported provisional responses (1XX).

SUBSCRIBE and **NOTIFY** will be introduced later, but their purpose is to indicate interest in the presence of a user.

4.1.2. Responses

A start-line here is called for a status-line which consists of Version, status-code, and information text. The following classes of response are defined in SIP:
1XX: Informational messages like trying, ringing, and call are being forwarded
200: Success OK
3XX: Redirection: moved permanently, moved temporarily, use proxy
4XX: Client error: bad request, unauthorized, payment required
5XX: Server error
6XX: Global error

I want to remind the reader that SIP messages are sent between agents to establishing a media session, but the final goal of XSP is not to establish a media session, actually the purpose of XSP is to exchange information between agents, and from captured information each agent decides some action. However, the action can lead to setting up or changing the characteristic of a media session.

Therefore to use INVITE, REGISTER, BYE, ACK, CANCEL, OPTION and MESSAGE will not be suitable for this aim. INFO is unsuitable too because the INFO method sends application layer information, generally related to an existing session, i.e. when XSP communication begins it is not certain that there has been a media session established. Unfortunately if some media session has been established it is not certain the XSP messages are related to the session.

4.1.3. Headers

A header in SIP carries information. Some of headers belong only to requests and some of them only to responses and they are called request header and response header.

The following headers concern extensions: require, supported, proxy-require, and unsupported headers.

Supported: The UAC part of extension of SIP uses this header. The UAC must use this header in a request to inform other SIP UA about extensions this Client supports.

Require: The UAC part of extension of SIP uses this header. The UAC uses this header to require another SIP UA to support some extension.

Proxy-require: it uses as a requirement to proxies that the request will be forwarded by. These headers are reserved for standard-track RFC extensions, thus XSP is not allowed to uses these headers, until XSP is a standard-track RFC.

4.1.4. Body

The body can be used by extensions, generally a SIP client must be prepared to receive messages with a body. However the client need not understand what to do with a given body. Naturally the functionality may be reduced when the client does not understand the body. Both requests and responses are able to have a body.

Message body type and length

When a body is present, there are two headers that will be interesting. The first one is Content-length and the other is Content-type. The Content-length informs the receiver of the message about the length of body in bytes and the Content-type informs them about the format of the body.

A message body can be encoded, in this case the header Content-encoding must be present in the message and in otherwise encoding must be avoided.

4.2. Registrar
A registrar is a special SIP UA server that accepts REGISTER requests, and places the information it receives in those requests into the location service for the domain it handles.

4.3. Location service
A location service is used by a SIP redirect or proxy server to obtain information about a callees possible location(s). It is database and is sometimes referred to as a location server. The contents of the database can be populated in many ways, including being written by registrars.

4.4. Dialog
A dialog represents a peer-to-peer SIP relationship between two user agents that persists for some time. The dialog facilitates sequencing of messages between the user agents and proper routing of requests between both of them. UAC and UAS procedures depend strongly on the method used by a request; this may depend on whether the request or response is inside or outside of a dialog.
An example of a request sent outside of a dialog is OPTION. The INVITE method is the only method that could result in establishing a dialog among other SIP standard methods. However, an INVITE method could be sent both within and outside a dialog.

4.5. Proxy
A proxy is a component that forwards a SIP request to UAS and a SIP response to UAC. There are stateless and state full proxies. A stateless proxy forwards requests and response without remembering transaction state, but a state full proxy remembers the transaction state of each incoming request and each response sent is result of a managed incoming request.

4.6. SIP and Multicast
Unicast and multicast routing are two ways to transport a packet over Internet. Unicast is the most usual way to send a packet to a destination today. When we use multicast we send a packet to group of destinations. Broadcasting (i.e. multicasting to all) is a way to send packets within a LAN to all nodes within that LAN. Multicasting is possible in SIP. Using multicast is very simple and flexible. SIP uses usually multicast for two aims, the first is discovery which we will describe more in section 4.8 and the second is multicast session invitation, here multicasting is used when a user starts a conference call by sending only one INVITE request to a multicast address. For example an INVITE request could be sent to all nodes in domain nada.kth.se by sending a simple INVITE with sip:*@nada.kth.se as a request URI. However we know that not all of these users are in a single multicast domain.

4.7. OPTION request

This type of request asks the SIP UA or proxy server about their ability with respect to type of methods, content-type, codec, extensions, and so on.
This type of request seems to be useful for XSP in order to find other SIP client's which are XSP capable, thus extending the discovery mechanism in SIP.

4.8 Discovery in SIP

SIP's discovery mechanism makes it easy for a caller to find a callee client. Discovery uses the three following SIP components:

- Registrar
- Proxy
- Location service

A SIP user agent wants to inform a proxy about its localization, the proxy needs this information to route incoming requests and responses to the current location of the agent. A Registrar server is a server, which accepts REGISTER requests and registers the location of the client who sent the REGISTER request to a location service. The location service itself is not defined by SIP. Location service is an abstract name for a service which has the following two characteristics:

- A Registrar must be able to write and read from the location service.
- A proxy server must be able to read from location service.
- The location service can be a database, a file, or a web page on the Internet.

The registering mechanism is:
The SIP user agent sends a REGISTER request to the Registrar server then the Registrar registers the location of the user with the location service. From that moment on the proxy will be able to route all requests and responses, which belong to the client to the correct location. When a proxy receives a message, the proxy looks for the receiver's SIP URI (given in the message) in the location server and when the proxy learns the location of the receiver, then the message will be sent to the receiver. The SIP implementation I have access to does not contains any proxy, Registrar, or location service therefore this mechanism could not be used in this project.

How does the SIP UA know the location of Registrar server to send REGISTER requests to?
That can be done by three following ways:

- Manual configuration: Local administrator could configure the address of Registrar in to SIP UA with installation moment.
- Using address–of–record: The UA uses the host part of the address-of-record as the Request-URI and address the request there, using the normal SIP server

location mechanisms. For example, the UA for the user “sip:vahid@nada.kth.se” addresses the REGISTER request to “sip:nada.kth.se”.

- Multicasting: SIP UA could send a multicast message to the sip.mcast.net (224.0.1.75) to which all SIP UA, listen.

4.9. General User Agent Behavior

In this section we will discuss rules for UAC and UAS behavior for generating and processing of requests and responses outside of a dialog.

4.9.1. UAC Behavior

This section discusses generating a request and its format.

4.9.1.1. Generating the Request

A valid SIP request is generated by a UAC contains at a minimum, the following mandatory header fields: To, From, Cseq, Call-ID, Max-Forwards, and Via.

Request-URI

The initial Request-URI of the message is set to the value of the URI in the To field. When a pre-existing route set is present, the procedures for populating the Request-URI and Route header field as detailed in Section 12.2.1.1 must of [RFC 3261] be followed (even though there is no dialog), using the desired Request-URI as the remote target URI.

To

The To header field specifies the desired "logical" recipient of the request, or the address-of-record of the user or resource that is the target of this request. The To header field contains a SIP URI. A request outside of a dialog doesn’t contain a To tag; since the tag in the To field of a request identifies the peer of the dialog and no dialog is established, no tag is present.

The following is an example of a valid To header field:

To: Theo <sip:Theo@ericsson.com>

From

The From header field indicates the logical identity of the generator of the request, possibly the user's address-of-record. Like the To header field, it contains a URI. It is used by SIP elements to determine which processing rules to apply to a request (for example, automatic call rejection).

As such, it is very important that the From URI not contain IP addresses or the FQDN of the host on which the UA is running, since these are not logical names.

The From field always contains a new "tag" parameter, chosen by the UAC.

Example:

From: Theo <sip:Theo@ericsson.com> ;tag=314p

Call-ID

The Call-ID header field is a unique identifier to join a series of messages in same group. It is the same for all requests and responses sent by either UA in a dialog. It is the same in each registration from a UA. So all request and response within the same dialog have the same Call-ID.

Example:

Call-ID: f81d4fae-7dec-11d0-a765-00a0c91e6bf6@foo.bar.com

CSeq

The CSeq header field serves as a way to identify and order transactions. It consists of a sequence number and a method. The method matches that of the request. For non-REGISTER requests outside of a dialog, the sequence number value is arbitrary.

Example:

CSeq: 4711 INVITE

Max-Forwards

The Max-Forwards header field is used to limit the number of hops a request can transit on the way to its destination. The value of this header is an integer that is decremented by one at each hop. If this value reaches 0 before the request reaches its destination, it will be rejected with a 483 (Too Many Hops) response.

A UAC always insert a Max-Forwards header field into each request it originates with a value that usually is 70.

This header was not defined in RFC 2543, therefore the SIP implementation used in this project don't use this header.

Via

The Via header field indicates the transport path used for the transaction and identifies the location where the response is to be sent. A Via header field value is added only after the transport path that will be used to reach the next hop has been selected.

When the UAC creates a request, it always inserts a Via into that request. The Via header field value **always** contains a branch parameter. This parameter is used to identify the transaction created by that request. The branch parameter value is unique across space and time for all requests sent by the UA.

The branch ID uses of amount of others proxies to determine the incoming request is sent by an implementation of RFC 2543 or RFC 3261. If the value of the branch ID begins with the characters “z9hG4bK”, then other SIP components who receives this request will understands that the request is sent by an application that implements RFC 3261.

Contact
The Contact header field provides a SIP URI that can be used to contact that specific instance of the UA for subsequent requests. The Contact header field is mandatory in each request and each request that can result in the establishment of a dialog contains exactly one Contact header.

Supported and Require
Support and require use of SIP extensions define in standards-track RFCs informs other components of the supported methods and required methods to make subsequent communications possible. The Supported, Require, and Unsupported headers are not allowed to be used in non standards-track RFCs like XSP. This is to prevent servers from insisting that clients implement non-standard, vendor-defined features in order to receive service.

Additional Message Components and Body
After including the mandatory headers described above, it is time to include optional and specific header fields depending on the method to be used.

The body field is the absolutely last element in a request and it is optional for all SIP requests and responses. The body usually carries information about the session that will be established later in a communication sequence.

4.9.1.2. Sending the Request
When a request is generated and it is ready to send. The request is passed to the transaction layer and after that to the transport layer of SIP. Transaction and transport layers use a queue to process requests and responses.

4.9.1.3. Processing Responses
Responses are first processed by the transport layer and then passed up to the transaction layer. The transaction layer performs its processing and then passes the response up to the above layer which calls for transaction user (TU). The majority of response processing is method specific.

4.9.2. UAS Behavior

When a request outside of a dialog is processed by a UAS, there is a set of processing rules that are followed, independent of the method. Note that request processing is atomic. If a request is accepted, all state changes associated with it are performed. If it is rejected, no state changes will be performed.

Processing of requests will be done in following steps. The initial authentication step is skipped here.

4.9.2.1. Method Inspection

If the method is not supported, then the following response is generated: 405 (Method Not Allowed). This response includes an Allow header that lists all supported methods of this server.

4.9.2.2. Header Inspection

All headers in a request that are not understood by the server will be ignored and no error response will be generated. Processing of the request will continue with all understood methods.

4.9.2.3. To and Request-URI

The To header normally identifies the original recipient of the request. In some circumstance, such as call forwarding the To header may not be the same SIP URI as the server that processed the request. Behavior of UAS when the To header is not the UAS URI is determined by the local policy of the UAS. However for interoperability it is recommended that the server accept the request. A 403 (Forbidden) response is generated when the UAS decides to reject the request. The Request-URI identifies the UAS that is to process the request. A 416 (Unsupported URI Scheme) response is generated when the Request-URI use a scheme that is not supported by the UAS. A 404 (Not Found) response is generated when the UAS doesn't accept the request for the address given in Request-URI.

4.9.2.4. Merged Requests

If the To header of the received request has no tags then the UAS checks the request against ongoing transactions. A 482 (Loop Detected) response is generated if the From tag, Call-ID, and CSeq exactly match those associated with an ongoing transaction, but the request does not match that transaction the UAS.

4.9.2.5. Require

As we described earlier the Require header is used only with of SIP extensions defined in standards-track RFCs. Therefore behavior of UAS is not interesting for XSP as long as XSP is none standards-track RFC.

4.9.2.6. Content Processing

A 415 (Unsupported Media Type) response is generated when the UAS doesn't understand the body of the message. Note that the body is optional only when no Content-type, Content-language, or Content-Encoding is present in the message. This response contains an Accept header that lists all type of bodies it understands. If the request contained content encodings not understood by the UAS, the response contains an Accept-Encoding header field listing the encodings understood by the UAS. If the request contained content with languages not understood by the UAS, the response contains an Accept-Language header field indicating the languages understood by the UAS.

4.9.2.7. Applying Extensions

A UAS that wishes to apply some extension when generating the response does not do so unless support for that extension is indicated in the Supported header field in the request. If the desired extension is not supported, the server relies only on baseline SIP and any other extensions supported by the client. In rare circumstances, where the server cannot process the request without the extension, the server sends a 421 (Extension Required) response. This response indicates that the proper response cannot be generated without support for a specific extension. The needed extension(s) will be included in a Require header field in the response. This behavior will generally break interoperability and therefore is not recommend. Any extensions applied to a non-421 response are listed in a Require header field included in the response. The server will not apply extensions not listed in the Supported header field in the request. As a result of this, the Require header field in a response will only ever contain option tags defined in standards-track RFCs.

4.9.2.8. Processing the Request

Assuming all of the checks in the previous subsections are passed, the UAS processing becomes method-specific.

4.9.2.9. Generating the Response

When a UAS wishes to construct a response to a request, it follows the general procedures detailed in the following subsections. Additional behaviors specific to the response code in question, which are not detailed in this section, may also be required. Once all

procedures associated with the creation of a response have been completed, the UAS hands the response back to the transaction server from which it received the request.

4.9.2.10. Sending a Provisional Response

Generally a UAS doesn't issue a provisional response for a non-INVITE request. Instead of a provisional response a final response to a non-INVITE request is sent as soon as possible.

4.9.2.11. Headers and Tags

The From field of the response equals the From header field of the request. The Call-ID header field of the response equals the Call-ID header field of the request. The CSeq header field of the response equals the CSeq field of the request. The Via header field values in the response equals the Via header field values in the request and **must** maintain the same ordering.

If a request contained a To tag in the request, then the To header field in the response equals that of the request. However, if the To header field in the request did not contain a tag, the URI in the To header field in the response must equal the URI in the To header field; additionally, the UAS must add a tag to the To header field in the response (with the exception of the 100 (Trying) response, in which a tag **may** be present). This serves to identify the UAS that is responding, possibly resulting in a component of a dialog ID.

The same tag will be used for all responses to a request, both final and provisional.

4.10. Firewalls and NAT

Firewalls pose a difficult challenge to SIP sessions. When we used TCP it is not so difficult to configure the firewall to pass SIP messages. But this does not help the media path, however, which uses UDP and will be blocked by most firewalls. A firewall needs to understand SIP, be able to parse a request, extract the IP address and port numbers from the content of body (often SDP), and finally open up pin hole in the firewall to allow traffic to pass and close these holes when media sessions are finished.

Network address translations (NAT) cause more serious problems for SIP. NAT is used sometimes for security purposes, to conserve or hide IP addresses and the LAN structure behind the NAT. NAT is often used on routers and firewalls that provide connection of a LAN to the Internet.

As we have seen in section 4.9.1.1 SIP responses are routed by using Via headers. When a request is forwarded outside the intranet by the NAT, the UDP and IP packet will be rewritten with a temporary global Internet address (often the IP of the machine running the NAT). The NAT will keep track of the binding between the local address and the global address so that incoming packets can be routed to the correct local address. Note that the IP address in the SIP message (both headers and body) will never be touched by the NAT,

so the IP address given in Via headers will not be translated and the server will use that non-unique address to send responses, and if another machine has the same address then the machine might receive the response to a request which it had not sent. This problem will directly affect XSP. To solve this problem a proxy or UAS that receives the request needs to detect if the request was sent through a NAT or not. This can be determined by comparing the IP address of the sender with the address given in Via header. If the address is not the same there is probably a NAT unless the packet is corrupted or the packet was sent by a hacker. If a NAT has been used, the UAS does not use the address given in Via to send its response, but rather it uses the IP address of the IP packet instead.

4.11. Reliability

SIP has a reliability mechanism defined, which allows the use of unreliable transport layer protocols like UDP. This mechanism will not used for TCP, which implicitly has a reliability mechanism. This reliability mechanism uses: retransmission timers, increasing CSeq numbers, and finally positive acknowledgement.

4.12. XSP as a SIP extension

In this section we will examine SIP when the focus is transport of messages between clients. We will present the methods, headers and tags that can be useful for XSP.
The first step for an XSP agent is discovering on the Internet for the others XSP agents to start sending subscription and receiving notifies.

Multicasting is described in section 4.6, Multicasting in SIP can be used in XSP's discovery mechanism, and we have discussed that in more details in section 4.8.

The second step is to send a subscription to another XSP agent when the client has awareness about location of the other server who will receive the subscription.

The discovery mechanism will be successful when all online clients has learned location of a new client and the client has learned location of all already online clients. In this section I will present four different ways for XSP discovery. In some of them we will just use SIP's methods and with other we will purpose some extension by defining new methods and headers and finally we will discuss advantages and disadvantages of each proposal.

As we described earlier in section 4.9.1.1 XSP is not allowed to use: Support, require, proxy-require, and unsupported.

Hear is the headline of my four proposals for discovering of other XSP agents:

- Using SIP registration mechanism
- Using OPTIONS method with multicast
- Using a central server on the public Internet

- Built-in discovery

4.12.1. Using SIP registration mechanism

In this proposal we will use SIP's registry mechanism to register a client to a location server and get an address-of -records list over all other users that have already make a registration from the location server.

Since the address-of-record list consists of both simple SIP UA and all SIP UAS which are collocated by XSP, the client who receives the list needs to remove from the list all simple SIP UAs and save only address-of-records to the XSP agents.

Earlier in section 2.1 we described how registration will be done, here we will describe more exactly the behavior of a UAC, proxy, and Registrar and their which limitations and recommend which solution is suitable.

4.12.1.1 Overview

The first step for an XSP agent is to register the location of this agent. The registration is exactly like a SIP registration, which was described in section 4.12 and we don't need make any extension. Now the client is reachable for other clients who have the address-of-record of this client. In the worst case when no client has any information about this client, this client may want to learn location of all other registered clients. The Registrar has read and write access to the location server and a proxy server has read access to it so the client needs to send a request either to a proxy server or to a Registrar server and ask for a list of address-of-records of all registered users. The question is who is best to send this request to and which type of method should we use? We look first at the case of asking the proxy, as we mentioned earlier a proxy doesn't need to understand the method of a message. If we define a new method that must be understood by proxies, then we don't follow the extension guideline described in section 3.1. Thus we should send our request to Registrar server and asks him for the list. Since the registrar server is a UAS which understands the Register method, the request must be a Register request. A register request is able to add, remove and update bindings. We will now study the headers of a Register request and examine how to best to use the Register method.

4.12.1.2. Constructing of Register requests

As for all other requests constructed by a UAC, the register request must have the following headers To, From, CSeq, Call-ID, Max-Forwards, and Via. A Register request does not need, but can have a Contact header.

REGISTER: as mentioned the semantics of Register is removing, adding and updating the location of SIP UA. Note that registration can be done by a third party.

To: The To header field contains the address of record whose registration is to be created, queried, or modified. The To header field and the Request-URI field typically differ, as the former contains a user name. This address-of-record must be a SIP URI.

From: The From header field contains the address-of-record of the client responsible for the registration.

Call-ID: It is a global unique identify for a call. All registrations from a UAC use the same Call-ID header field value for registrations sent to a particular registrar. If the same client were to use different Call-ID values, a registrar could not detect whether a delayed REGISTER request might have arrived out of order.

Max-Forwards: this limits the number of hops before the destination.

CSeq: The CSeq (Command sequence) value guarantees proper ordering of REGISTER requests. A UA increments the CSeq value by one for each REGISTER request with the same Call-ID.

Via: This header is the path to the destination and indicates the path that a response must take to return.

The following is an example of a minimal REGISTER method:

```
REGISTER sip:registrar.nada.kth.se SIP/2.0
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: Vahid <sip:vahid@nada.kth.se>
From: Vahid <sip:vahid@nada.kth.se>;tag=456248
Call-ID: 843817637684230@998sdasdh09
CSeq: 1826 REGISTER
```

4.12.1.3. Constructing of 200 OK response

When a registrar receives the above REGISTER request it first validates the message. If the validation is successful the registrar does the appropriate operation.
Notice no contact header is present in this method this means the aim of this registration request is not really registration but rather obtaining a list of registered locations.
The 200 response to this request will contain a list of all location of the user with SIP URI vahid@nada.kth.se. Note a 200 OK must always contain a Contact header. An example of 200 OK follows:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKnashds7
;received=130.237.226.215
To: Vahid <sip:vahid@nada.kth.se>;tag=2493k59kd
From: Vahid <sip:vahid@nada.kth.se>;tag=456248
```

```
Call-ID: 8438176376842 30@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:vahid@130.237.226.215>
Expires: 7200
```

After introducing the REGISTER method and its response we want to try to construct a REGISTER method which returns the location of all registered users within a 200 OK response.We usually use a SIP URI in the To header to inform registrar which location we are interesting of, there is a special SIP URI which means all users in the domain:

Sip:*@domain.countrycode

That means any-users at domain **domainname.countrycode**
So the whole request will look like the following:

```
REGISTER sip:registrar.nada.kth.se SIP/2.0
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKnashds7
Max-Forwards: 70
To: <sip:*@nada.kth.se>
From: Vahid <sip:vahid@nada.kth.se>;tag=456248
Call-ID: 8438176376842 30@998sdasdh09
CSeq: 1826 REGISTER
```

When this message is received by the registrar, hopefully a 200 OK response will be sent back. However, there could be other responses if the validation failed or the request was not accepted. A successful response could look like the following code.

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKnashds7
;received=130.237.226.215
To: <sip:*@nada.kth.se>;tag=2493k59kd
From: Vahid <sip:vahid@nada.kth.se>;tag=456248
Call-ID: 8438176376842 30@998sdasdh09
CSeq: 1826 REGISTER
Contact: <sip:theo@130.237.226.212>; expires=3600
Contact: <sip:chip@130.237.226.213>; expires= 2400
Contact: <sip:calle@130.237.226.214>; expires=7000
Contact: <sip:vahid@130.237.226.215>; expires=7100
```

After processing this message the client has information about the location of all the users in this domain who have a SIP UA online. The client is still not finished, the last step remains, reducing the list to only who are XSP capable. I propose to use the standard SIP method OPTIONS. The client asks each SIP UA “are you an XSP user?” by sending an OPTIONS request to each location in the list. The following example is an OPTIONS request from vahid to Theo:

```
OPTIONS sip:theo@nada.kth.se SIP/2.0
```

```
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKhjhs8ass877
Max-Forwards: 70
To: <sip:theo@nada.kth.se>
From: Vahid <sip:vahid@nada.kth.se>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:vahid@130.237.226.215>
```
Accept: application/sdp

The response looks like the following when Theo is a XSP capable client:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP exjobb.nada.kth.se:5060;branch=z9hG4bKhjhs8ass877
;received=130.237.226.215
To: <sip:theo@nada.kth.se>;tag=93810874
From: Vahid <sip:vahid@nada.kth.se>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:theo@nada.kth.se>
Contact: <mailto:theo@ericsson.se>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, XSPEVENT
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Content-Length: 274
(The body does not show here)
```

Examine the above response we see the Allow header contains all standard methods defined in SIP and a non-standard method XSPEVENT. XSPEVENT is the method Theo will use to send and receive a XSP subscription and notification.

When the client has received all the responses from all other SIP UA users, he is capable to reduce the SIP UA list to a XSP list.

If XSP was an IETF standard extension to SIP the client would be allowed to use the Supported header to inform all SIP UA that I support XSP instead using of using the OPTIONS method to ask them if they are XSP capable agents. However this is not a limitation for XSP because XSP can always define a new header call it XSPSupported and XSPRequired for this aim, but in this case XSP must not expect all SIP UAS to understand this header, The rule in SIP is when a header is not understood it is simply ignored and this rule ensures SIP's interoperability.

4.12.2. Using OPTIONS method with multicast

In previous section we get a list of location of online SIP UAs by using the REGISTER method and reduce the list we used via the OPTIONS method with unicast. In this section we will only use OPTIONS with multicast.

4.12.2.1. Overview

This approach requires that all XSP capable UAs at up-start register with a multicast group and listen to this group. When an OPTIONS request is sent to a multicast group, the message is sent first to the a proxy and the proxy routes the message to the router who is responsible for multicasting in the local area network, thereby the message will be sent to the all nodes on the LAN.

When multicast is in use it is important to note that the UAC can receive many responses to the single one request. The behaviour of UAC when it receives messages is following. The first incoming response to UAC process as a transmission and the other responses process as a retransmission. When we send request by multicasting, multiple responses will be generated depends on number of registered clients to the multicast group. All of these responses will contains exactly same branch parameter in the topmost Via header but different in the To tag. The first received response will be used as a response to the sent request and all response after that will be considered as retransmissions. That is not an error; multicast SIP provides only a rudimentary "single-hop-discovery-like" service that is limited to processing a single response.

4.12.2.2. Constructing the OPTIONS request

To construct the OPTIONS requests we follows the same rules as described for constructing requests as was described in section 4.9.1.1 with the following difference. Apart from the 6 mentioned headers the Accept header is mandatory header for an OPTIONS request. A Content header could be sent within an OPTIONS request, but it is not mandatory. General when a client sends a request to a multicast address it **must** add the maddr parameter to it's Via header field value containing the destination multicast address.

An example of OPTIONS request to a multicast can look like the following code.

```
OPTIONS sip:*@nada.kth.se SIP/2.0
Via: SIP/2.0/UDP xsp.mcast.net ;branch=z9hG4bKhjhs8ass877
Max-Forwards: 70
To: <sip:*@nada.kth.se>
From: Vahid <sip:vahid@nada.kth.se>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:vahid@130.237.226.215>
```

```
Accept: application/sdp
Content-Length: 0
```

The To header contains the special char "*" as a user-ID, but this is ok because a UAS who finally receives the message doesn't care about the To header.
A response to this message will be sent to same multicast group. A 200 OK response to the OPTIONS must contain the following headers: Allow, Accept, Accept-Encoding, Accept-Language, and Supported headers. Therefore a 200 OK response in this case will look like the following:

```
SIP/2.0 200 OK
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4bKhjhs8ass877
;received=192.0.2.4
To: <sip:*@nada.kth.se>;tag=93810874
From: Vahid <sip:vahid@nada.kth.se>;tag=1928301774
Call-ID: a84b4c76e66710
CSeq: 63104 OPTIONS
Contact: <sip:theo@nada.kth.se>
Contact: <mailto:theo@ericsson.se>
Allow: INVITE, ACK, CANCEL, OPTIONS, BYE, XSPEVENT
Accept: application/sdp
Accept-Encoding: gzip
Accept-Language: en
Supported: foo
Content-Type: application/sdp
Content-Length: 274
```
(SDP not shown here)

4.12.3. Using a central server on the public Internet

This solution considers use of a central register server on the public Internet. This approach is very typical today for example most of file sharing applications like Direct Connect, KaaZZA, and Morpheus use this same mechanism. These applications try to have minimal possible control and computing on their central server and therefore the central server acts as intermediary, like a Registrar server in SIP.

All XSP clients at startup send a XSP subscription to the server, the server registers the user and sends back a list of all XSP users that are already registered along with a notification to all users when a new client sends a subscription. This solution is not considered further in this project because this mechanism is not based on SIP. Therefore we will not more discuss about details of this solution, but we will comment on advantages and disadvantages of this solution when we compare this solution with others in section 4.13.

4.12.4 Implicit discovery

With implicit discovery the XSP client doesn't care about the location of all other XSP clients, it simply sends all XSP subscriptions to a known multicast group which all other XSP listen to and it receives notification on same multicast group. This solution actually is not a discovery solution, but a way to make a XSP subscription.

4.12.5 Sending and receiving XSP subscriptions and notifications

When the XSP client knows the location of other XSP agents, it can start to send subscriptions and receiving notifications. However, an XSP agent would prefer to receive notifications from the start. In this section we will propose different ways to carrying XSP subscriptions and notifications by using or extending SIP. Before introducing some of them I want to present the SUBSCRIBE and NOTIFY extension to SIP.

4.12.5.1. How SUBSCRIBE and NOTIFY works

4.12.5.1.1. Overview

As we mentioned earlier SUBSCRIBE and NOTIFY were not designed for all class of events. SUBSCRIBE and NOTIFY requires event packages as an extension. Thus to use SUBSCRIBE and NOTIFY requires defining a new event package for XSP. An example of an already defined package is the presence package. A natural way to think of an event is that the state of a resource changes without any memory. The following example makes it easy to understand: we assume a subscription for new incoming messages, which are saved in a database until the user, decides to read them. We assume two designs of notification for incoming of messages. The first type of notification could be a simple notification that informs the uses of newly received messages and the second type of notification could be a notification that says that there are 7 new messages in the database. The latter type is more complicated than the first and can cause to synchronization problems.

The SUBSCRIBE request is used for registering, refreshing, or deregistering of subscriptions. While the NOTIFY request is used to send notification of an occurred event. Event and Allow-Event are the two newly defined headers that carry information about the type of event and a list of allowed events. 202 accepted and 489 bad events are the two newly defined responses. All of these are defined in the draft SIP-specific event notification [SEN].

4.12.5.1.2. Description of SUBSCRIBE behavior

A subscription contains the following headers: To, From, and Call-ID. If some of the headers are different in two subscriptions then they are two different subscriptions. An Expires header is always included within a SUBSCRIBE method. This Expires header works as a proposal of the expiring time of subscription to the notifier. This proposal can

be accepted or changed to a lower value, but never to a larger value. Expires also indicates how many minutes remain in the subscription going out.
A 200 OK response sends to this SUBSCRIBE request and this response contains the expires time as determined by the notifier. Refreshing of subscription could be done by sending a SUBSCRIBE message with a new Expires time. Finally by sending a SUBSCRIBE message with an Expires header with value 0 corresponds to leads to unsubscribe.

A SUBSCRIBE message contains exactly one Contact header and exactly one event header. The event header contains a single opaque token that identifies the class of the event.

4.12.5.1.3. Subscriber's SUBSCRIBE behavior

When a subscriber decides to subscribe to a SIP UA it forms a SUBSCRIBE message. A subscribe message looks like the following:

```
SUBSCRIBE sip:presentity@pres.example.com SIP/2.0
 Via: SIP/2.0/UDP  watcherhost.example.com:5060
 From: User <sip:user@example.com>;tag=9988
 To: Resource <sip:presentity@example.com>
 Call-ID: 3248543@watcherhost.example.com
 CSeq : 1 SUBSCRIBE
 Expires: 600
 Accept: application/cpim-pidf+xml
 Event: presence
 Contact: sip:user@watcherhost.example.com
```

The Headers From, To and Call-ID from a unique identification for the message. Expires is the proposed lifetime of this subscription. CSeq increases when a new subscription is sent, this also applies for refreshing and unsubscribing. Via and Contact works as described in section 4.9.1. The Accept header indicates the allowed formats of the body in subsequent NOTIFY requests. The To headers indicates address of the resource the subscription is being made to. Generally this value matches the request URI of the message. The From field is the address of the subscriber.

A response of other than 2XX class indicates that no subscription was created and therefore no notification will be generated.
Note that subscribers are always prepared to receiving a notification even if the subscription transaction is going on and it is not yet complete.

4.12.5.1.4. Notifier's SUBSCRIBE behavior

Notifier checks incoming SUBSCRIBE message to understand the event header indicated in the SUBSCRIBE message. The mean of the 2XX class response is the only reliable information, as generally the body of response will not contains any information beyond the subscription duration.

4.12.5.2. Description of NOTIFY behavior

4.12.5.2.1. Overview

A NOTIFY request contains the same Call-ID, local URI, and remote URI as the corresponding SUBSCRIBE request. The NOTIFY and SUBSCRIBE have same call-leg. To differentiate distinct NOTIFY methods the From header indicates a tag.

When forking of a subscription occurs, the subscriber will receive notifications with different From tag's in the NOTIFY message than the To tags sent in the SUBSCRIBE message.
As we described for the SUBSCRIBE message the NOTIFY also has a simple opaque token that identifies the class of event.

When a body is present the format of the body contains matches the indicated format in Accept header in SUBSCRIBE request.

It is possible that a NOTIFY is received without a subscription has been sent in some situations, therefore subscribers are always prepared to receive notifications not corresponding to any subscription.

4.12.5.2.2. Notifier's NOTIFY behavior

When a state with a subscribed resource changes the notifier forms a NOTIFY request to send to the subscriber. The notification will be sent to the same address as is indicated in the Contact header in the corresponding SUBSCRIBE request.

The behaviour of the notifier when constructing a call leg is the same as we described in section 4.9.1.1. The Contact header present in a NOTIFY method is that used by the subscriber to send the subscription. A NOTIFY message may contain an Expires header to inform the receiver of the remaining subscription time.

4.12.5.2.3. Subscriber's NOTIFY behavior

When a notification is received by a UAS, the subscriber checks a corresponding Call-leg. Generally if there is no matching call leg found, then the subscriber will send a "481 call

leg/transaction does not exist“. A notable exception to this behavior will occur when clients are designed to receive NOTIFY messages for subscriptions set up via a means other than a SUBSCRIBE message (e.g. HTTP requests, static provisioning). Such clients will need to, under certain circumstances, process unmatched NOTIFY requests as if they had prior knowledge of the subscription. If, for some reason, the event package designated in the "Event" header of the NOTIFY request is not supported, the subscriber should respond with a "489 Bad Event" response. To prevent spoofing of events, NOTIFY requests may be authenticated, using any defined SIP authentication mechanism. If the Expires header value defined in the NOTIFY request is 0, then the subscriber should terminate that subscription.

4.13. Analysis and Comparison of solutions

In this section we compare all proposals for XSP discovery. The central server on the public Internet as we mentioned is not consider suitable because that is not SIP related. Furthermore it clearly does not scale. Furthermore there is extremely high reliability requirement of such a globally administrator of server, if for some reason the server goes down the discovery mechanism will not work for any link except when the clients use earlier awareness information stored in its ACM. In addition the server may be overloaded when there are many clients online at same time. However, there are some solutions to this problem which today are used by web servers that receive many requests per unit time.

The advantage of the solution with a central server is that the discovery will works on the global Internet without any additionally requirements, but in the other non-central server solutions it is not sure that all online XSP agents can indicate to all other XSP clients in some other LAN, this depends on the configuration of routers between different LANs.

All solutions with multicasting have the advantage that it is simple and flexible to use it. Assume the solution with the OPTIONS method and multicast. By sending only one OPTIONS request to a multicast group the client obtained the location of all other online XSP agents, but the solution with REGISTER in the latter part of solution sends an OPTIONS request to all SIP UA produce the same result. All solutions with multicast have the disadvantage that multicast registration may be inappropriate in some environments, for example, if multiple businesses share the same local area network. The requirement for the solution with REGISTER works as a registrar server. However the registrar server needs SIP to become the localization mechanism.

4.14. Conclusions

As we discussed in earlier sections non-standard extensions such as XSP are not allowed to use Support and Require headers. In the discovery section we propose to define new methods and OPTIONS method to find other XSP agents. Defining new methods in SIP to go around the problem that XSP is not allowed to use Support headers is not merited because the semantics of the defined method is not used if we don’t use the methods to

make an XSP subscription and notification. From the other side as we studied in section 4.12.5.1 all we need is make an XSP subscription and notification are already defined by SUBSCRIBE and NOTIFY methods in SIP and to redefine the same methods. While XSP is an extension of SIP so SIP's firewall, security, and NAT problems will be XSP's problems too. For authentication and authorization XSP may use the same mechanisms defined in RFC 3261.

5. Implementation

In the previous section we demonstrated the use of SUBSCRIBE / NOTIFY methods to realize the goal of this project. In this section we will introduce the implementation of the model as a prototype. The general code is written in Java 1.2.

The SIP user agent we used was implemented in Java. The XSP will be implemented in Java while the ACM will make use of SICSTUS Prolog v.3.9.0. The ACM part will be implemented in Java and Prolog, using JASPER for access to Prolog via Java and vice versa. In the MSK part of the model, the resource description format will be in prolog syntax. In addition we will make use of JAXB to generate a java object from XML and eventually regenerate XML from the manipulated java object, because XSP messages are sent in XML.

5.1. Design

The figure 5.1 shows the messages passing between a subscriber and notifier.

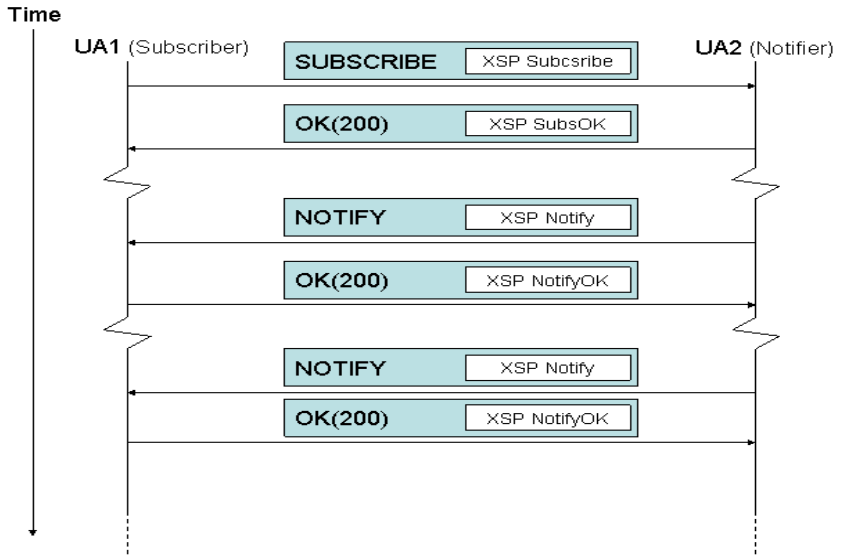


Figure5.1 XSP's subscription and notification carry by SUBSCRIBE / NOTIFY

The XSP subscription message is put into a SIP SUBSCRIPTION message and sent to the other side, in the same way XSP Notifications are put in a SIP NOTIFICATION message. Each SIP SUBSCRIBE and NOTIFY message will be acknowledge by a SIP success OK 200 response which carries a XSP -SubscriptionOK or -NotifyOK.

Each User Agent is collocated with an XSP agent. When a SUBSCRIBE or NOTIFY message is received the User Agent looks for the event type of the message, if the event type is "XSP" the body of message will be delivered to the XSP component.

XSP consist of an XSP-Engine, ACM, and SICStus-Prolog. The XSP-Engine receives XSP messages, and registers a subscription if the message is an XSP-subscription after that sends a SIP OK messages to the subscriber by SIP, this message carries an XSP-SubscriptionOK. Note a "Deny" message could be sent instead of XSP -SubscriptionOK if the User agent didn't accept subscription. XSP-Engine will call functions from the ACM to register this subscription in Prolog. The ACM is actually a thread which at some interval will check for new events; if there are some new events which must be sent to a user, ACM will create a MSK message and include XSP tags and deliver to the XSP-Engine to be sent by a SIP NOTIFY method to the subscriber. Notice that the XSP message is completely dependent on stored rules in ACM.

The figure 5.2 is an overview of this mechanism.

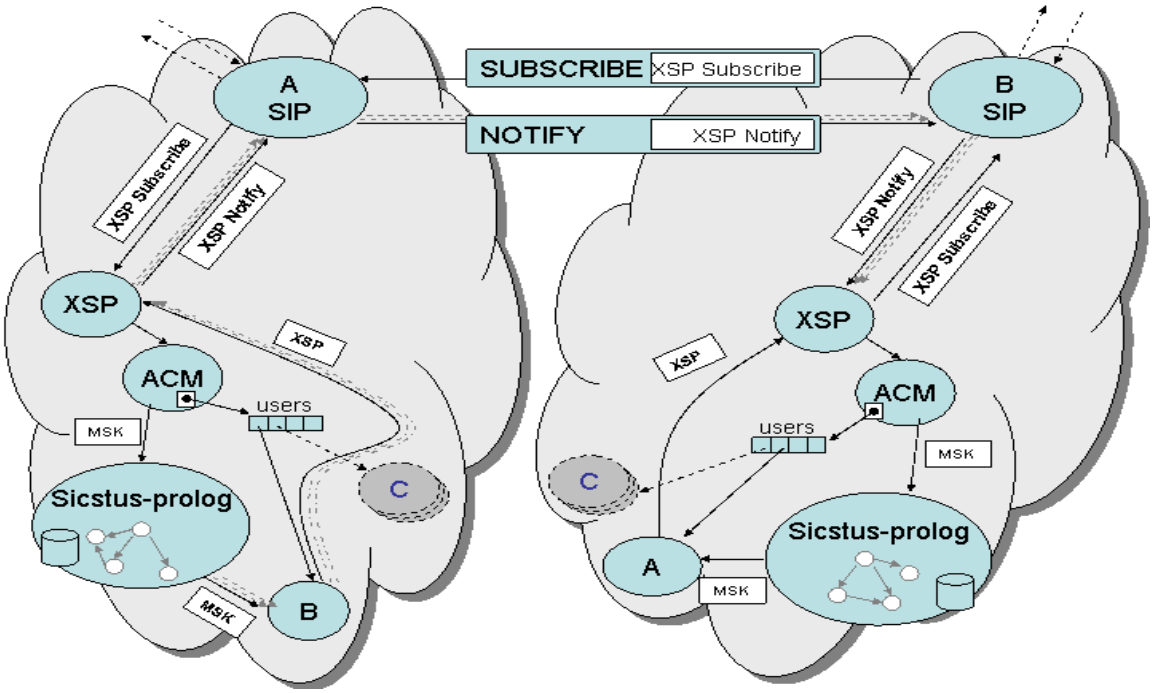


Figure5.2 overview of SIP, XSP and ACM

This picture shows an overview of XSP messages passing between SIP User agents.
Each ACM object has a list of online XSP-users, for example the ACM object in the left side of the picture has a user list "users", and the list contains objects of java class user. These objects represent an online XSP-agent, so each XSP agent has a representative object in other XSP-agents, these representative objects contain minimal information about the XSP agent, such as which subscriptions this object has registered, and the CallLeg which initiated a subscription, etc.

Information in SICStus-prolog is presented here as graphs. Each incoming "MSK" from an ACM object can lead to a change of these graphs. SICStus-prolog delivers new notifications to the representative objects in the list. The representative objects include XML tags to make an XSP message of it for sending by SIP to the respective XSP agent.

5.2. Format of XSP message

The form of an XSP message is XML. Below is a BNF like description of the XSP format.

xsp: subs | xspevent | register | eventOK | subsOK | deny
subs: msk,user,callID
xspevent: msk,user, callID
register: user
user: SIPUID
msk: Prolog
calleg: CALLID
eventOK: userid of subscriber, callID
subsOK: userid of notifier, callID
deny: userid of sender of message, callID

xsp : this message is an xsp message
subs: xsp subscription
xspevent: xsp notify
register: register as a xsp-agent
eventOK: acknowledgement to the notification
subsOK: acknowledgment to subscription
deny: subscription fails.
user: is SIP user ID.
Msk: is msk description in prolog format.
Callleg: the callID SIP UA has initiate for a SIP subscription.

5.3. JAXB and XML

Actually using JAXB and XML was not necessary in this project, but it made it considerably easier to parsing and manipulate XSP messages. Below is a brief introduction to XML and how JAXB works.

5.3.1. XML

XML is a language used to describe data. Consider the following example:
We want to send some data to a client information about a library. Assume these data are: "TCP/IP Illustrated" and "Stevens". Consider sending this data as a lump to a client. The client need to understand different parts of the data, the sender therefore need to describe in some way for the client that "Stevens" is the *author* and "TCP/IP Illustrated" is the *title* of his *book* in a *library*.This description can be done in XML as follows:

```
<library>
 <book>
  <title>TCP/IP Illustrated</title>
  <author>Stevens</author>
 </book>
<library>
```

Although the receiver of this message could to parse and acts upon this message. The problem is not solved yet. The client will be confused if the sender sends the following message and doesn't describe the document type or format of this message.

```
<library>
 <book>
  <title>TCP/IP Illustrated</title>
  <author>Stevens</author>
 </book>
 <book>
  <title>Introduction to Algorithms</title>
  <author>Cormen</author>
 </book>
<library>
```

Thus we need something to describe the format of the document. Definition of documents will be done by a DTD (Document Type Definition) description. In this case the DTD description will look like the following:

```
<!DOCTYPE library [
  <!ELEMENT library   (book)* >
   <!ELEMENT book   (title,author)>
  <!ELEMENT title  (#PCDATA)>
  <!ELEMENT author (#PCDATA)>
]>
```

Here * means 1 or more.

Description the client understands format of the message, and is able to pause the message.

<!ELEMENT library (book)* >	between each pair of *library* elements must be more than one book element.
<!ELEMENT book (title, author)>	between each pair of *book* element must be two other elements *title* and *author*.
<!ELEMENT title (#PCDATA)>:	the text is the title
<!ELEMENT author(#PCDATA)>:	the text the name of author of the book.

Notice in XML and DTD no semantics are described. The semantics of each XML data element must be specified in other ways; currently there is no general way to describe the semantics of XML document elements.

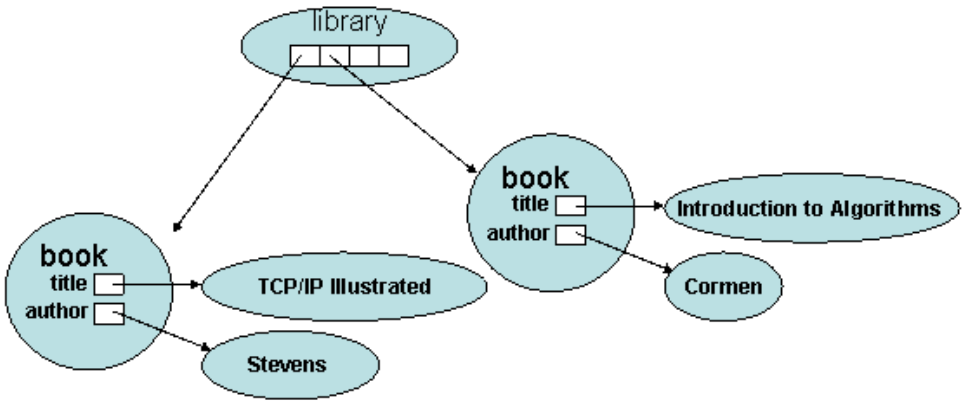
5.3.2. JAXB

In this section we will see how JAXB was useful in this project. Consider the following XML document:

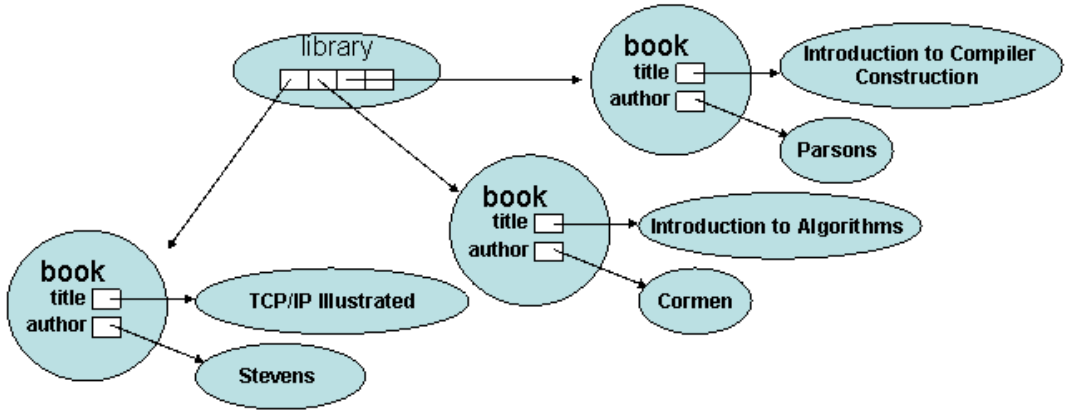
```
<library>
 <book>
  <title>Introduction to Algorithms</title>
  <author>Cormen</author>
 </book>
 <book>
  <title>TCP/IP Illustrated</title>
  <author>Stevens</author>
 </book>
</library>
```

When the above code is sent to a JAXB application, JAXB will create three instances of java objects:

1. library
2. book
3. book



The library instance will contain a list that points to objects 2 and 3. Assume the java application creates a new instance of a book and adds it to the list in the library object.



The resulting java object now has four objects: library, book, book, book, and therefore the XML document JAXB gives us will be:

```
<library>
 <book>
  <title> Introduction to Algorithms</title>
  <author> Cormen</author>
 </book>
 <book>
  <title> TCP/IP Illustrated </title>
  <author> Stevens </author>
 </book>
 <book>
  <title> Introduction to compiler construction </title>
  <author> Parsons </author>
 </book>
</library>
```

The above example has used a very simple structure of java classes, to illustrate what JAXB does. However, the structure of java classes can be very complex and thus the java objects will be complex, therefore JAXB is a very useful tool to generate XML documents from the manipulated java objects.

Since according to [APMC, Pg.150] the ACM stores information in the form of graphs, every new information item or events may change the state of the graph in the ACM, which may lead to some conclusion which should be sent to another user. I believe JAXB is a very useful tool to generate XML document from the java objects, thus allowing of the java classes and the hierarchy to be complex.

5.3.3. Example

Sara is interested in following the football match between teams AIK and Djurgården.
The following XML document is an example of an XSP message that contains her subscription for getting such notification. A user agent with user id "vahid@nada.kth.se" sends the following subscription request to all XSP clients.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
  <subs>
    <msk>wants(sip:sara@nada02.nada.kth.se,football)</msk>
    <user>sip:sara@nada02.nada.kth.se</user>
    <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
  </subs>
</xsp>
```

Note the data in "callID" tags will be different for each different notifier, it means if A sends a subscription message to B and C, the callID will not be the same in these messages. The CallID is a unique identifier for any new subscription. Acknowledgement of the above message will look like:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
 <subsOK>
  <user>sip:vahid@nada01.nada.kth.se</user>
  <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
 </subsOK>
</xsp>
```

After a while the state of the match (i.e., current score) between AIK and Djurgården changes to 1-0 and the new state will be sent to Sara as an XSP message.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
<xspevent>
 <msk>has(AIK Djurgarden 1 0,football)</msk>
  <user>sip:vahid@nada01.nada.kth.se</user>
  <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
     </xspevent>
    </xsp>
```

and the following XML is sent as an acknowledgment to the above XSP -notification

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
 <eventOK>
```

```
  <user>sip:sara@nada02.nada.kth.se</user>
  <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
 </eventOK>
</xsp>
```

After a while again the score changes to 1-1, a new notification will sends to Sara:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
 <xspevent>
  <msk>has(AIK Djurgarden 1 1,football)</msk>
  <user>sip:vahid@nada01.nada.kth.se</user>
  <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
 </xspevent>
</xsp>
```

and the following XML is sent as an acknowledgment to the above XSP -notification

```
<?xml version="1.0" encoding="UTF-8"?>
<xsp>
 <eventOK>
  <user>sip:sara@nada02.nada.kth.se</user>
  <callID>b5018ed7a706d1cf@nada02.nada.kth.se</callID>
 </eventOK>
</xsp>
```

In the above we see some redundancy, for example “sip:sara@nada02.nada.kth.se” both in user and MSK tags. The reason for this is that the ACM consists of two components in java and prolog.
“wants (sip:vahid@nada.kth.se, football)” is from Prolog part of ACM and “sip:vahid@nada.kth.se” is from the java part of ACM. It is possible to avoid this type of redundancy, but would require some parsing of MSK messages, as this was not central to this project and since efficiency was not the main focus. I have not attempted to remove this redundant data. Including CallID is important to identify notify messages.

5.3.4. The DTD file

The DTD file for this XML document looks:

```
<?xml version="1.0" encoding="US-ASCII"?>
<!ELEMENT xsp (subs|xspevent|register|eventOK|subsOK|deny)?>
<!ELEMENT subs (msk,user,callID)>
<!ELEMENT xspevent (msk,user, callID)>
<!ELEMENT register (user)>
<!ELEMENT user (#PCDATA)>
<!ELEMENT msk (#PCDATA)>
```

```
<!ELEMENT callID (#PCDATA)>
<!ELEMENT eventOK (user, callID)>
<!ELEMENT subsOK (user, callID)>
<!ELEMENT deny (user, callID)>
```

To generate java classes, JAXB needs an **xjs** file that describes the relation between each class. In this example XSP will be the root class (we must always define one root class), by root we means that all other tags will be between the tag of root class, in this example all other tags are between <xsp> and </xsp>. In this case we used a simple format and no inherent is necessary therefore the xjs file looks like:

```
<xml-java-binding-schema version="1.0-ea">
<element name="xsp" type="class" root="true" />
</xml-java-binding-schema>
```

We don't need to describe all other the tags and classes as JAXB includes a default value automatically if nothing is given.

We give **xjc** the JAXB compiler the xsp.dtd and xsp.xjc files as arguments and it will generates the following classes: Deny.java, EventOK.java, Register.java, Subs.java, SubsOK.java, Xsp.java, and Xspevent.java.

The following java code shows how we create a new XSP subscription message:

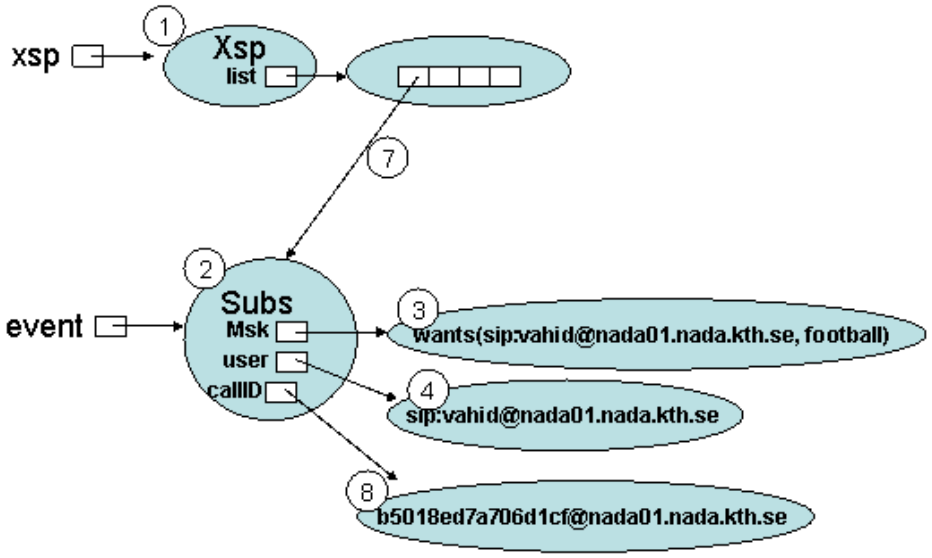
```
1. Xsp xsp=new Xsp();
2. Subs event=new Subs();
3. event.setMsk("wants(sip:vahid@nada01.nada.kth.se,
                        football)");
4. event.setUser("sip:vahid@nada01.nada.kth.se");
5. xsp.emptyContent();
6. List l=xsp.getContent();
7. l.add(event);
8. event.setCallID(callID);
           try{
9. xsp.validate();
           }catch(Exception e){
               e.printStackTrace();
           }
```

Here is a description for each line of the above code:

1. Creates a new Xsp object.
2. Creates a new Subs object.
3. Includes the Msk tags into the Subs object.
4. Includes the user tags into the Subs object.
5. Clears objects from the list in Xsp object.
6. Accesses the list in the Xsp object.
7. Adds Subs object to this list.

8. Includes CallID tag into the Subs object.
9. Validates this format. If an error occurs, it means that the structure of the object are not in agreement with the DTD file.

The resulting structure is shown below:



References:

[AKCIT] J. Rosenberg and R. Shockey, *"The Session Initiation Protocol (SIP): A Key Component for Internet Telephony"*, www.convergence.com/article/CTM20000608S0019/1, June 2000

[APMC] T. Kanter, *"Adopted Personal Mobile Communication"*, Doctoral Dissertation, Teleinformatics, Royal Institute of Technology, November 2001.

[D-ISP] J. Rosenberg, D. Willis, H. Schulzrinne, C. Huitema, B. Aboba, D. Gurle, D. Oran," *Session Initiation Protocol (SIP) Extensions for Presence"*, www.ietf.org/internet-drafts/draft-ietf-simple-presence-07.txt, May 2002

[D-ISEIM] B. Campbell,J. Rosenberg, H. Schulzrinne, C. Huitema, D. Gurle, "*Session Initiation Protocol Extension for Instant Messaging*", www.ietf.org/internet-drafts/draft-ietf-sip-message-04.txt, Internet draft, May 2002

[ISO] International Organization for Standardization, www.iso.org

[GAES] J. Rosenberg, D. Willis, H. Schulzrinne, "Guidelines for Author of Extension to SIP", www.ietf.org/internet-drafts/draft-ietf-sip-guidelines-06.txt, November 2002

[JAXB] "The Java Architecture for XML Binding", java.sun.com/xml/jaxb/jaxb-docs.pdf, May 2001

[KQML] T. Finin, J.Weber, G. Wiederhold, M. Genesereth, R. Fritzson, D. McKay, J. McGuire, R. Pelavin, S. Shapiro, S. Buffalo, C. Beck "*Specification of the KQML Agent-Communication Language*" www.cs.umbc.edu/kqml/papers/kqmlspec.pdf, June 1993

[RFC 2976] S. Donovan, "*The SIP INFO method*" , www.ietf.org/rfc/rfc2976.txt?number=2976, October 2000

[RFC 2543] J. Rosenberg, E. Schooler, H. Schulzrinne, M. Handley "*SIP: Session Initiation Protocol*", www.ietf.org/rfc/rfc2543.txt?number=2543, March 1999

[RFC 3261] J. Rosenberg, E. Schooler, H. Schulzrinne, M. Handley, G. Camarillo,

A. Johnston, J. Peterson,R. Sparks
"*SIP: Session Initiation Protocol*",
www.ietf.org/rfc/rfc3261.txt?number=3261, June 2002

[RFC 3265] A.B.Roach, "*SIP-Specific Event Notification* ",
www.ietf.org/rfc/rfc3265.txt?number=3265, June 2002

[SICS] Swedish Institute of Computer Science, www.sics.se/sicstus

[SICSJ] Swedish Institute of Computer Science,
www.sics.se/sicstus/docs/3.9.0/html/sicstus.html#Jasper

[USIP] Alan B. Johnston, "*Understanding the Session Initiation Protocol*",
ISBN 1-58053-168-7, Artech-house 2000, Artech-house

[XML] Petter Åström, "*XML*", ISBN 91-7882-497-4, Docendo AB, 1999

Acronyms

ACM	Active Context Memory
ATM	Asynchronous Transport Mode
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISO	International Organization for Standardization
KQML	Knowledge Query and Manipulation Language
MIME	Multi-Purpose Internet Mail Extensions
MSK	Mobile Service Knowledge
OSI	Open System Interface
RDF	Resource Description Form
SDP	Session Description Protocol
SIP	Session Initiation Protocol
TCP	Transmission Control Protocol
UA	User Agent
UAC	User Agent Client
UAS	User Agent Server
UDP	User Datagram Protocol
URL	Uniform Resource Locator
XML	eXtensible Markup Language
XSP	eXtensible Service Protocol