

# WeSAHMI Status Report

September 29, 2006

## Abstract

This is the status report of the framework developed in the WeSAHMI research project. The document covers the project deliverables D2-D4. It includes a summary of used technologies, description of the current state of the subprojects, and gives an overview of planned future work.

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Summary of technologies to be used</b>	<b>3</b>
2.1	Dynamic configuration of IPv4 addresses . . . . .	3
2.1.1	Overview . . . . .	3
2.1.2	Selecting an address . . . . .	3
2.1.3	Conflict detection . . . . .	4
2.2	Service Location Protocol . . . . .	5
2.3	The Session Initiation Protocol Family . . . . .	8
2.3.1	Instant Messaging and Presence . . . . .	9
2.3.2	Distributed SIP . . . . .	10
2.3.3	Security Support . . . . .	11
2.4	SOAP . . . . .	12
2.5	HTTP . . . . .	12
2.6	Web Technologies . . . . .	13
2.6.1	XHTML . . . . .	13
2.6.2	XForms . . . . .	13
2.6.3	SVG . . . . .	13
2.6.4	Compound Document Formats . . . . .	14
2.6.5	XBL . . . . .	14
2.6.6	CSS . . . . .	14
2.7	Network . . . . .	14
<b>3</b>	<b>Interaction model</b>	<b>15</b>
3.1	Background . . . . .	15
3.2	Mode of implementation, phase 1 . . . . .	15
<b>4</b>	<b>SOAP and SIP binding</b>	<b>17</b>
4.1	Subscribing to notification service . . . . .	17
4.2	Receiving notifications . . . . .	17

<b>5 Platform</b>	<b>18</b>
5.1 Client . . . . .	18
5.2 Server . . . . .	21
<b>6 Security</b>	<b>23</b>
6.1 Requirements . . . . .	23
6.2 Building Blocks for Security . . . . .	24
6.2.1 End-to-End Measures . . . . .	24
6.2.2 SIP Security . . . . .	25
6.2.3 Web Services Security . . . . .	27
6.2.4 Identity Federation . . . . .	28
6.3 Security Specification . . . . .	28
6.3.1 Overview . . . . .	28
6.3.2 Bootstrapping Trust . . . . .	30
6.3.3 Secure Push . . . . .	30
6.4 Security Domains . . . . .	32
6.5 Privacy . . . . .	33
6.6 Browser Integration . . . . .	33
6.7 Putting it Together . . . . .	33
6.8 Implementation Plans . . . . .	34
<b>7 Detailed application-level description</b>	<b>35</b>
7.1 User Story: Check-in . . . . .	35
7.2 Pilot Environment . . . . .	37
<b>8 Future Work</b>	<b>39</b>
8.1 Pushlet Environment . . . . .	39
8.2 Pushlet environment and SLP integration . . . . .	41
8.3 X-Smiles and SIP integration . . . . .	43
8.4 Security . . . . .	44
<b>9 Conclusions</b>	<b>44</b>
<b>Bibliography</b>	<b>46</b>
<b>A Used open source software and applying licenses</b>	<b>47</b>

## 1 Introduction

During the spring 2006, project Wesahmi has been actively composing an integrated system where components of already existing systems have been used. The goal has been to provide a reference implementation where client-server interaction is enhanced by allowing also servers initiate operations. An existing implementation was running in June 2006, and it was also demonstrated to project's steering group using the agreed case study as an example.

This paper documents the overall design and subsystems used in it. Each subsystem is also associated with information about applicable open source licences, where an already existing component is used. Moreover, the paper defines the main open issues that will be addressed in the subsequent phases of

the project. Some of these are already under current work, but the majority of them remains future work.

The structure of this report is the following. Section 2 provides an overview of the technologies used in the implementation. Sections 3 provides a discussion on the improved interaction model that is needed for allowing servers to sending notifications to clients. Section 4 introduces how interaction is technically implemented at messaging and protocol level, and Section 5 addresses client and server design. Considerations regarding security are give in Section 6. A detailed application level description on the demonstrated system is given in Section 7. Then, in Section 8, we will list open issues that are currently under work or to be solved in the future. Finally, conclusions are drawn in Section 9. Open source licenses applicable to different subsystems that have been used on composing the reference implementation are listed in Appendix A.

## 2 Summary of technologies to be used

This section covers all used technologies. It has the following structure. Section 2.1 presents a method for IPv4 address autoconfiguration. Section 2.2 gives an overview of Service Location Protocol (SLP) and a passive discovery enhancement. Section 2.3 presents baseline Session Initialization Protocol (SIP) and also a distributed version of SIP. Section 2.5 briefly introduces Hypertext Transfer Protocol (HTTP) and Section 2.6 addresses several Web technologies used to create user interface. Finally, Section 2.7 covers the target environment network infrastructure.

### 2.1 Dynamic configuration of IPv4 addresses

#### 2.1.1 Overview

The dynamic configuration of IPv4 link local addresses is specified by RFC 3927 [6]. It defines a mechanism for how a node on the network can configure an IPv4 address without the use of external services such as DHCP server. The address is defined to be link local which means that it can be used to communicate with nodes in the same link. Being on the same link means that the nodes can communicate directly with each other so that the link-layer packet payload arrives unmodified. Address block 169.254/16 is registered for link local addresses.

#### 2.1.2 Selecting an address

The basic mechanism for obtaining an address is following:

1. Selection – a host selects an address using a pseudo-random number generator with a uniform distribution in the range from 169.254.1.0 to 169.254.254.255 inclusive.
2. Probing – the host tests if the selected IPv4 link-local address is already in use. On a link-layer such as IEEE 802 that supports ARP [23], conflict detection is done using ARP probes. The ARP probes query which host has a specific IP address and the owner responds to it. The probe is repeated a few times with small delays. If any host claims to own the

address by sending an ARP reply, or tries to find out the owner in similar manner, the address is considered taken and the algorithm returns to previous state to select a new address.

3. Announcing – When a host has found an available IP address it announces the new address to the network. It broadcasts a number of ARP announcements. The announcements are like probes but the sender and target IP addresses are both set to the host's newly selected IPv4 address.

An example of the mechanism is shown in Figure 1.

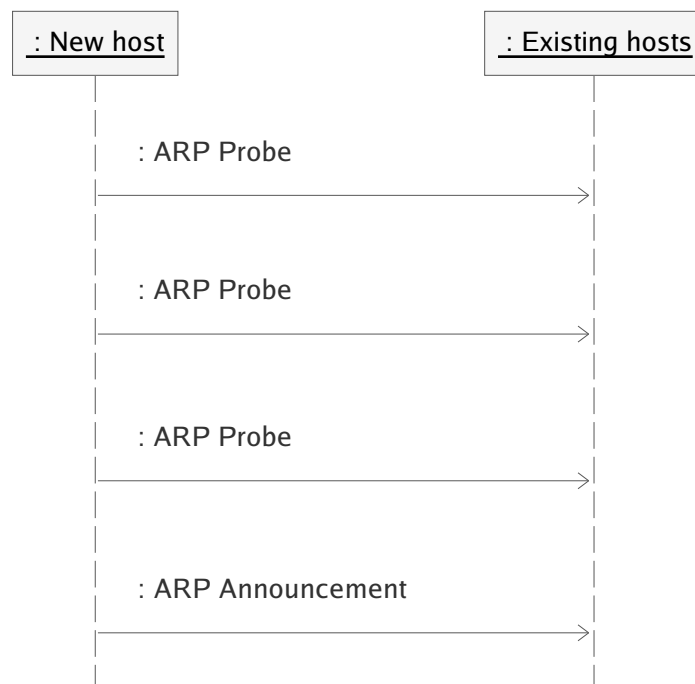


Figure 1: New host selects an address. All messages are broadcasted.

### 2.1.3 Conflict detection

Address conflict may occur at any time of the operation, not only during the address selection. For example, two hosts may select the same address when they do not have connectivity between them and later become aware of each other. At any time, if a host receives an ARP packet on an interface where the 'sender IP address' is the IP address the host has configured for that interface, but the 'sender hardware address' does not match the hardware address of that interface, then this is a conflicting ARP packet, indicating an address conflict.

On a conflict, a host has two possible options:

- The host may immediately configure a new IPv4 link-local address.

- The host may attempt to defend its address by recording the time that the conflicting ARP packet was received, and then broadcasting one single ARP announcement, giving its own IP and hardware addresses as the sender addresses of the ARP. Having done this, the host can then continue to use the address normally without any further special action. However, if this is not the first conflicting ARP packet the host has seen, and the time recorded for the previous conflicting ARP packet is recent, the host must immediately cease using this address and configure a new IPv4 link-local address.

Forced address reconfiguration may be disruptive, causing TCP connections to be broken. However, it is expected that such disruptions will be rare. Before abandoning an address due to a conflict, hosts should actively attempt to reset any existing connections using that address.

## 2.2 Service Location Protocol

Service Location Protocol[12] is protocol specified by IETF. It is targeted to search services from the network based on type of service and attributes. Thus SLP provides a dynamic configuration mechanism without the need to preconfigure service addresses. The services are represented as URLs. URLs consist of type of the service and the address where the service is available. Additionally the services can be grouped together with scopes and they can have attributes assigned.

SLP includes three entities that perform service discovery functions:

- User Agents (UA) perform service discovery.
- Service Agents (SA) advertise the location and attributes of the services.
- Directory Agents (DA) store and distribute service information.

When performing a search for a service the UA sends a multicast or broadcast Service Request (**SrvRqst**) to which SAs with corresponding services reply with unicast Service Reply (**SrvRply**). An example of these is in Figure 2. It shows how a UA multicasts or broadcasts a service request and a SA replies with unicast. In the second example, the UA has found a DA and uses it as a proxy to find services. On the third example a DA informs of its existence when a UA or SA performs multi- or broadcast traffic.

A Passive Discovery (PD) functionality was implemented for SLP within SESSI project. It enables service providers to advertise their services through broadcast advertisements in the ad-hoc network. The clients may then passively accumulate a list of services they are interested in. The functionality is mainly useful when the number of clients is significantly larger than the number of service providers. PD is therefore a suitable method for bootstrapping the Wesahmi framework. The operation of PD is illustrated in Figures 3 and 4.

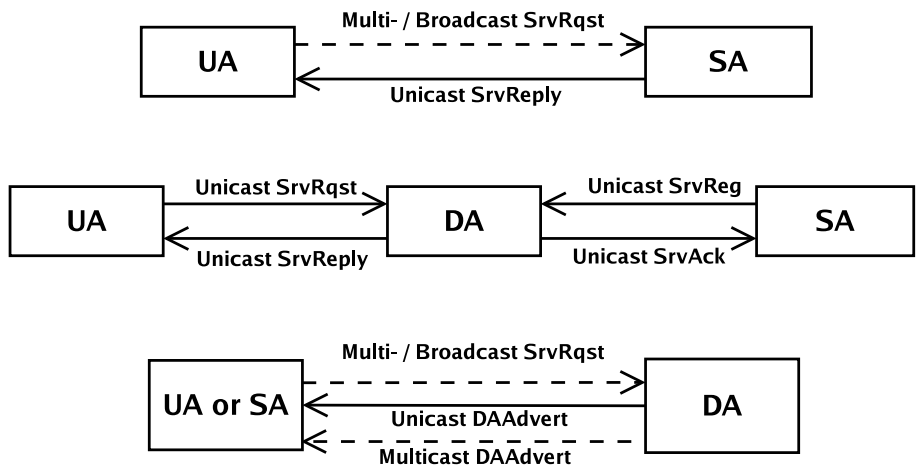


Figure 2: SLP agents and most common protocol messages.

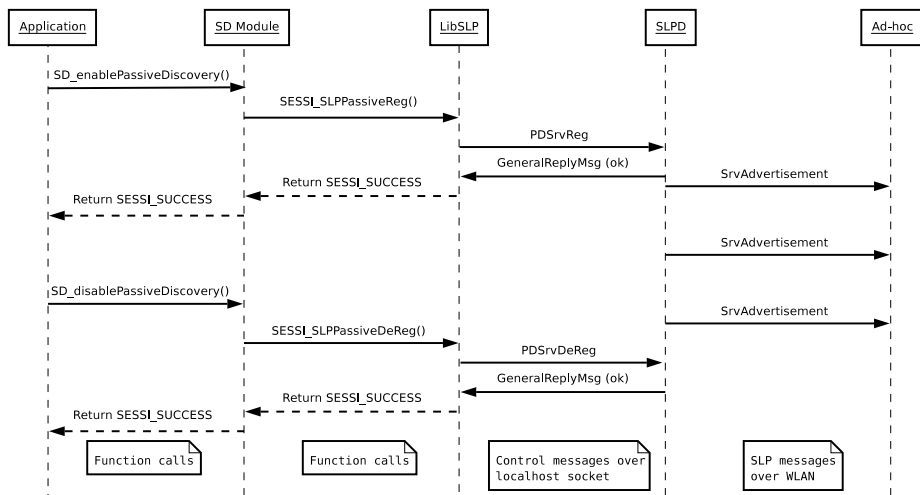


Figure 3: Registering and unregistering services for PD

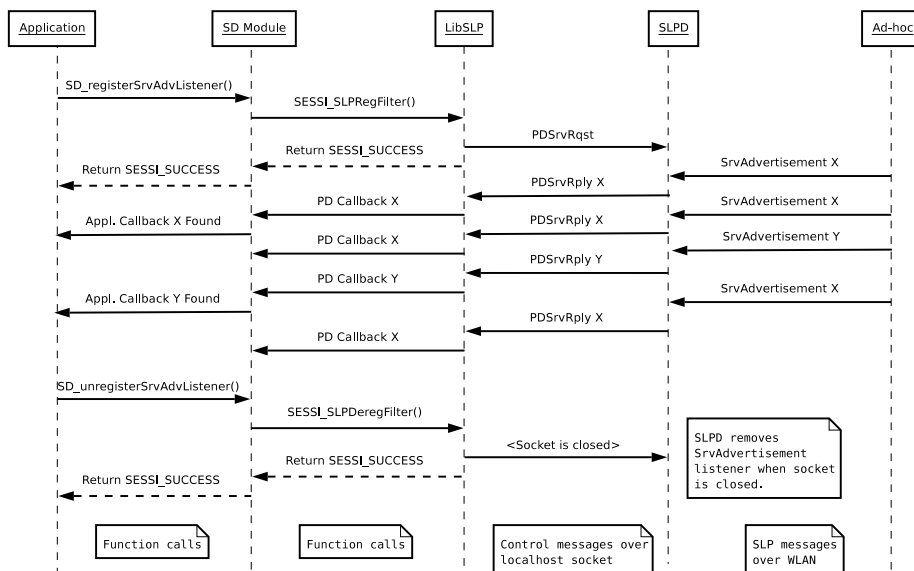


Figure 4: Discovering services with PD

### 2.3 The Session Initiation Protocol Family

The Session Initiation protocol (SIP) is a multi-purpose and flexible signaling protocol for session-based communications in IP networks. SIP only handles the session management phase, and once a session has been established, different communication applications can be used, e.g., Voice over IP, video conferencing, and instant messaging.

The architecture of the Session Initiation Protocol (SIP) [10] is based on centralized entities. Two logical elements play a key role in the architecture, *registrar* and *proxy servers*. Registrars are the SIP entities where SIP users register their contact information once they connect to the network. In a basic registration scenario, a SIP user agent communicates to its registrar server (the registrar IP address is usually preconfigured) the SIP user name of the user(s) using the device, referred to as SIP *address of records* (AOR) for that user, and the addresses where the user is reachable. Usually, contact information is stored in the form of IP addresses or resolvable names, but other kinds of contact information, such as telephone numbers can be registered as well.

An association between a SIP AOR and a contact address is called a *binding*. SIP registrars exploit an abstract service, called *location service*, and return the bindings for the SIP AORs falling under their domain of competence to the SIP entities issuing a binding retrieval request.

Proxy servers are needed because SIP users cannot know the current complete contact information of the callee but only its AOR. SIP presupposes that the AOR (SIP user ID) of the party to contact is known in advance, analogously to what happens when sending instant messages or e-mails. A basic SIP session involves the calling user agent contacting the calling side proxy server, which in turn will forward the message to the proxy server responsible for the domain of the called user agent. The called side proxy server retrieves from the called side registrar (i.e. utilizes the location service) the bindings for the called user and eventually delivers the request to the intended recipient.

Registrars and proxies are logical entities, and it is not an uncommon configuration for them to be co-located in the same node. Usually, user agents have a preconfigured outbound proxy server where all the outgoing requests are sent and through which all the responses to the issued requests, or new requests, are received.

A typical SIP session is set up as follows (Fig. 5). Alice tries to start session with Bob. Alice's phone uses a proxy server that is in atlanta.com domain as it's outbound proxy and Bob's phone uses proxy server in biloxi.com domain as it's outbound proxy. Alice starts by sending sending an INVITE request (1) which is received by Alice's outbound proxy. This proxy appends a *via*-header field containing it's address to the request and forwards it to proxy in the domain of Bob's phone (2). Alice's outbound proxy can use DNS to locate the inbound proxy which is in the biloxi.com domain. The proxy server at biloxi.com receives the INVITE request, also appends a *via*-header field to request, and forwards it to Bob's phone (3). The proxies also send messages back to Alice to inform that they have forwarded the request(4,5).

When Bob's phone receives the INVITE request it sends a message telling that the request has been received by the device and is ringing (6). Message is routed back to Alice's phone through same proxies that the request arrived. This is done by information in requests *via*-header fields. *Via*-header fields are



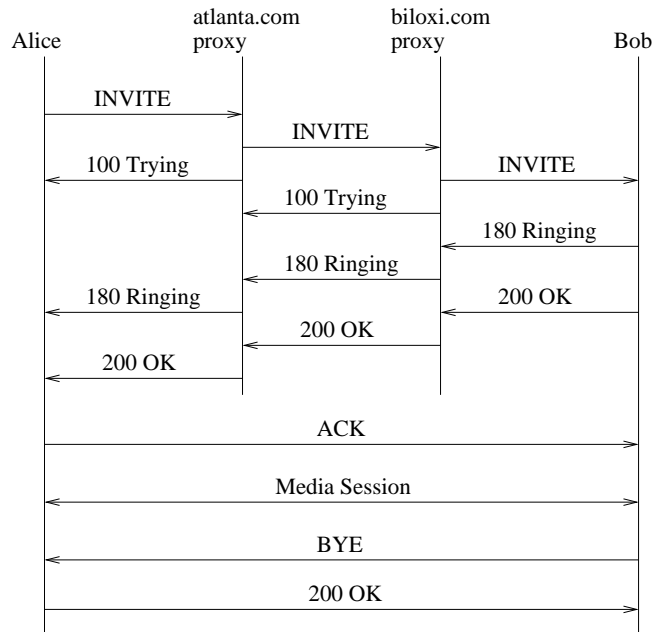


Figure 5: Example SIP session set up

removed from the request message in reverse order by each proxy in the route. When Bob answers the session invitation, a final response message is sent to Alice (7), which Alice confirms with an ACK message (8). the ACK message is sent directly to Bob's phone, because both Alice's and Bob's phones know each others addresses after the INVITE message exchange procedure and no address lookups are needed by proxy servers anymore. The Session is now established. The session is closed with a BYE-200 OK message exchanged (9,10).

### 2.3.1 Instant Messaging and Presence

SIP is essentially a signaling solution. Once the session is set up, an application is started to perform the actual communication between the users. A very popular type of communication between people is instant messaging. SIP has also been extended to support instant messaging and presence (IMP) services. The IMP architecture proposed by the SIMPLE working group builds on top of the SIP Event Notification Framework [4] and realizes a specific event instantiation called *presence* [26]. The general concept is that SIP entities can subscribe to the presence resource state owned by another entity. The entities that have accepted a subscription request send notifications when their presence state changes, to all the (authorized) entities.

Subscriptions and notifications are done using two newly defined SIP methods, SUBSCRIBE and NOTIFY [4]. Both methods are SIP requests; in the event package, the entity that processes such requests, thus handling the presence state of an entity, is called Presence Agent (PA). Usually, the PA is run in a centralized server, to facilitate presence management when a SIP user accesses the network from several different (presence) user agents simultaneously. The

presence state made available by a user can contain, e.g., profile information, such as, interests, and hobbies.

The transfer of messages between two users is done with the Message Session Relay Protocol (MSRP) [9], the protocol designed in SIMPLE for session mode instant messaging sessions. An MSRP IM session is signaled using SIP, exactly like any other media (e.g., audio, video) session. During the SIP session negotiation, the end peers exchange a URI, which will be used throughout the MSRP session as unique peer identifier. Once one party has received the URI identifying the remote peer, the MSRP session can start. The actual instant messages are exchanged in the body of the MSRP messages.

### 2.3.2 Distributed SIP

Decentralized SIP (dSIP) [19] is a solution that allows deploying SIP without support from centralized servers: MANETs are an example of target network environment for dSIP. The key idea of dSIP is of embedding in each enabled device a basic subset of SIP proxy and registrar server functionalities, so that dSIP users are self-capable to discover and contact other users in a MANET. Decentralized SIP is particularly suited for small MANETs, with few dozens of nodes at most, a size that constitutes a realistic deployment scenario for ad-hoc networks [30]. We refer to such particular type of MANETs as proximity networks, and here we use the term proximity interchangeably with MANET.

The software architecture of dSIP is shown in Fig. 6: the modules bordered within solid lines are standard SIP modules in a device. The dashed modules are instead the additions made to enable SIP in proximity networks. In a standard SIP client, only the user agent (UA) side of the stack would be present. In MANETs, the server module is added, and the server standard capabilities are enhanced with proximity functionalities. More details on the role of each module are provided in [19].

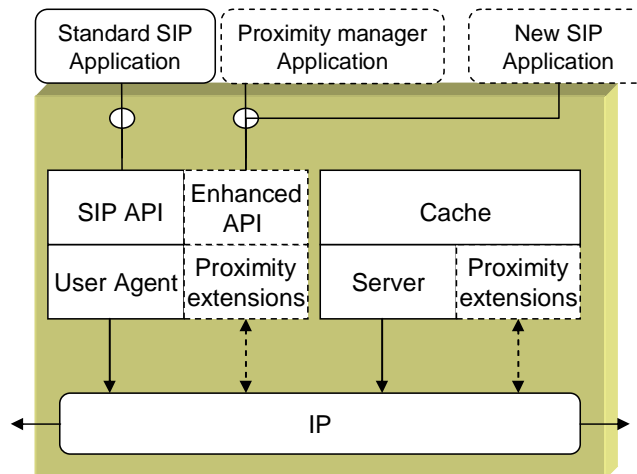


Figure 6: Software Architecture for decentralized SIP

The main point is that proximity capabilities are not realized by modifying

the existing software modules of a SIP device; rather, they are enabled by adding new submodules. This choice allows interoperability of dSIP UAs with standard SIP clients: in fact, a standard SIP application can be deployed on top of dSIP as well as an application that exploits the proximity enhancements. A native SIP application is unaware of the presence of a modified SIP stack in the device, since it only utilizes the standard SIP features. Moreover, a native application can be utilized in MANETs, since the underlying proximity-aware middleware is able to handle all the SIP messages sent by the application in the proper way.

The working principle of dSIP is that in MANETs, the user agent registers with the co-located registrar server, according to standard SIP procedures, by sending a REGISTER message. The server will then register the SIP user to the network spreading a SIP message; message spreading can be done in several ways, broadcast, flooding, or multicasting to the SIP well known multicast address. The server modules in the proximity network receive a REGISTER, update their cache entry with the binding, and can reply to the registering node by sending a 200 OK message. The registering node server module updates its cache with the bindings received from the other nodes. With this procedure, the SIP location service functionalities, usually handled by a centralized entity, the SIP registrar, are distributed among all the MANETs nodes. A native SIP application would register to its predefined external registrar server; the proximity enhanced modules "intercept" this message and route it to the local server, transparently for the application. With this approach it is ensured interoperability.

Inviting a peer to a SIP session is similar: the INVITE message is forced to the co-located server, which checks in its cache if it has a binding for the queried user (i.e., it is exploiting the location service), and forwards the INVITE to the correct address in case a match is found. Furthermore, a proximity aware SIP application may explicitly query the local server for the list of users in the proximity network; the server collects the list of currently stored bindings and sends them back, locally, so that a user in MANETs is able to begin sessions also with previously unknown users. The request and reply for user list is done by means of SIP messages: server and user agent modules are not bound by any function calls.

### 2.3.3 Security Support

Session management with SIP has various security issues, e.g., authentication of the parties, integrity of the messaging, and confidentiality. Because SIP is based on application layer routing, the integrity and confidentiality of SIP messaging is typically handled independently between two hops. Therefore, we concentrate on security issues related to SIP ad-hoc networking, and on how a SIP nodes are able to authenticate each other.

The main security concern in ad-hoc networks is making sure of the identity of the remote party, and the security of the signaling itself. Application data flows can be secured independently of the signaling messages. Verifying SIP users' identity can be handled by the SIP authenticated identity [21] extension to SIP. The key idea of the extension is that SIP UAs connect and authenticate to a SIP server, which runs an authentication service. Once the authentication service receives a message from an authorized UA, it signs the message using its domain certificate. The signature is computed by hashing certain relevant

header fields of the message and added into the new SIP Identity header field. The UA receiving the signed message can verify it using the authentication service domain certificate; the certificate is either previously stored at the receiving UA, or fetched at the address provided by the authentication service in another new Identity Info header field. The receiving UA trusts the authentication service, so by verifying the signature, it can be sure of the identity of the sender of the request and of the message integrity.

We have modified this approach so that each node in an ad-hoc networks signs all the SIP messages sent to the gateway (or to another node) with a self-signed certificate. The gateway node receives the signed message and verifies it using the ad-hoc user's certificate; if signature verification fails, or the gateway can not find the user's certificate, the ad-hoc user is denied the gateway access. Similarly, if the ad-hoc node has stored the gateway certificate in advance, it can verify its authenticity and trust it for accessing the Internet. The ad-hoc user's certificate can be retrieved from a well-known repository in the Internet, or could be previously stored at the gateway; this would be the case of a gateway managed by a network operator, which only provides access to subscribed users with pre-shared certificates. The gateway node, in this case, does not need to be a moving device, but it could be a node connected to the infrastructured network with one interface, and to the ad-hoc network with another. This scenario could find application in hot-spots, such as, airports or internet cafes; we deem it very interesting as it gives to ad-hoc networking a business value even for network operators.

## 2.4 SOAP

SOAP is xml-based lightweight protocol for exchanging information in a decentralized and distributed environment. Typically SOAP-messages are carried over HTTP-protocol but other protocols may also be used. Messages may travel from SOAP sender to SOAP receiver through SOAP intermediaries, which may do some processing with the message.

The three main elements of SOAP-messages are envelope, header and body. Envelope is the top level element. SOAP header must be the first element inside envelope, but it is an optional element. SOAP header may contain child elements which are called header blocks. These header blocks can be used for passing information that can be used by SOAP intermediaries. SOAP intermediaries can inspect, remove and add SOAP headers to the messages. After SOAP header there is a mandatory SOAP body. SOAP body is the place for the information that is meant for the ultimate receiver of the message.

## 2.5 HTTP

The Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is used for data transfer in WWW. The HTTP protocol is a request/response protocol. A client sends a request to the server in the form of a request method, URI, and protocol version, followed by a MIME-like message. The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message. Usually, HTTP communication is initiated by a user agent. [11]

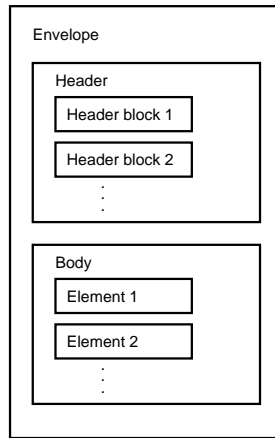


Figure 7: Structure of the SOAP-message

## 2.6 Web Technologies

The user interfaces (UIs) of the applications are build using novel Web technologies, which are mainly XML-based markup languages. The technologies are discussed in the following subsections.

### 2.6.1 XHTML

XHTML, the XML-based counterpart of the traditional HTML, is used to define layout and structure of Web documents. XHTML's layout model (flow layout), makes it easy to create user interface for all sizes of devices. That is, the layout is not tied to absolute positions and sizes. XHTML is modularied. Thus, one can use desired subset of it and add modules from other languages, if needed.

### 2.6.2 XForms

XForms 1.0 Recommendation [8] is the next-generation Web forms language, designed by the W3C. It solves some of the problems found in the HTML forms by separating the purpose from the presentation and using declarative markup to describe the most common operations in form-based applications [5]. It can use any XML grammar to describe the content of the form (the instance data). Thus, it also enables to create generic editors for different XML grammars with XForms. It is possible to create complex forms with XForms using declarative markup, without resorting to scripting. XForms needs a host language, which defines the layout of a form.

### 2.6.3 SVG

Scalable Vector Graphics (SVG) is a format for two-dimensional graphics. Since it is vector graphics, it can be rendered optimally on all sizes of device. SVG drawings can be interactive and dynamic. Animations can be defined and triggered either declaratively (i.e., by embedding SVG animation elements in SVG content) or via scripting.

#### **2.6.4 Compound Document Formats**

Several XML vocabularies have been specified in W3C. Typically, an XML language is targeted for a certain purpose (e.g., XForms for user interaction or SVG for 2D graphics). Moreover, XML languages can be combined. An XML document, which consists of two or more XML languages, is called compound document. A compound document can specify user interface of an application.

#### **2.6.5 XBL**

XML Binding Language (XBL) provides mechanisms to bind an arbitrary XML element to a binding element. The binding element defines the behavior and/or presentation of the arbitrary element. For instance, an XForms control can be bind to a SVG control, which is displayed if a device is capable to do that. XBL has three main usage scenarios. They are:

1. Extending a document.
2. Presentation and behavior encapsulation.
3. Presentation and behavior inheritance.

#### **2.6.6 CSS**

Cascading Style Sheets (CSS) is a mechanism for adding style to Web documents. CSS enables separation of style and content of the Web documents. That makes site maintenance easier and simplifies Web authoring.

### **2.7 Network**

The environment of the WeSAHMI project consists of an ad-hoc network with mobile nodes using services from the fixed infrastructure network using one or multiple gateway nodes. The ad-hoc network uses WLAN as low-level transport. The possible modes of operation are Managed using Access Points to or Ad-hoc that does not use Access Points. The network protocol is IPv4 using autoconfiguration (section 2.1).

There are two possibilities using the services from the fixed infrastructure side: creating a direct IP connection or using application level proxies. Using services from multiple providers makes pure IP connection difficult because the node has to route packets to several different IP gateways. Because of this, the services will be used through proxies and the IP spaces will be completely separate.

On the first phase the addresses can be set manually using private IP address blocks and making the gateway node as the default route. The gateway node performs NAT on the connections and allows direct IP connections to the fixed network. The NATted connection can also be used with autoconfigured addresses by setting only the route manually. However, later the direct connections should be removed.

## 3 Interaction model

### 3.1 Background

Interactive server-initiated services require an interaction model that differs from traditional web-based applications. The most obvious difference is that while web applications are based on "client pull", server-initiated services require support for "server push." Furthermore, a scheme for registering, authenticating, and authorizing users and services is required.

The use of a service consists of several messages passed between the server and the client. For example, when an air passenger arrives at the airport, a Finnair Plus server contacts him to suggest mobile check-in, which is followed by a series of communications related to the actual check-in. Together these communications form a *session*. By default, these sessions are initiated by the server, while either party can be responsible for ending them. For example, the user may choose to close his browser, which results in termination of the session.

### 3.2 Mode of implementation, phase 1

In the WeSAHMI project, SIP is used as the bearer for notifications. On a low level, it is possible to use the SIP SUBSCRIBE/NOTIFY methods to implement push services. Server-controlled sessions that can also be ended by the client are also possible using the associated timeout mechanism, where a subscription has to be renewed before a predefined time interval has passed.

In the initial phase of the project, the notifications mainly instruct the client to load a page to begin a service session or to reload a page to obtain refreshed status information.

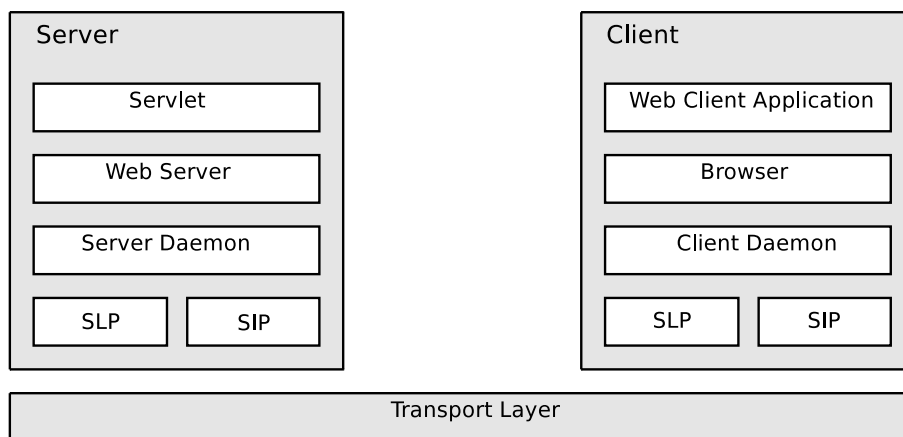


Figure 8: Current Infrastructure of the Framework.

The framework, whose main components are illustrated in Figure 8 and general operation is illustrated in Figure 9, is divided between two primary elements: server and client. In the server-end, Servlets provide the actual service but do not interact with the lower level elements but the Apache web server. Server daemon uses SLP to advertise the service and SIP to provide notifications for clients that have subscribed to the service. In the client-end, client daemon

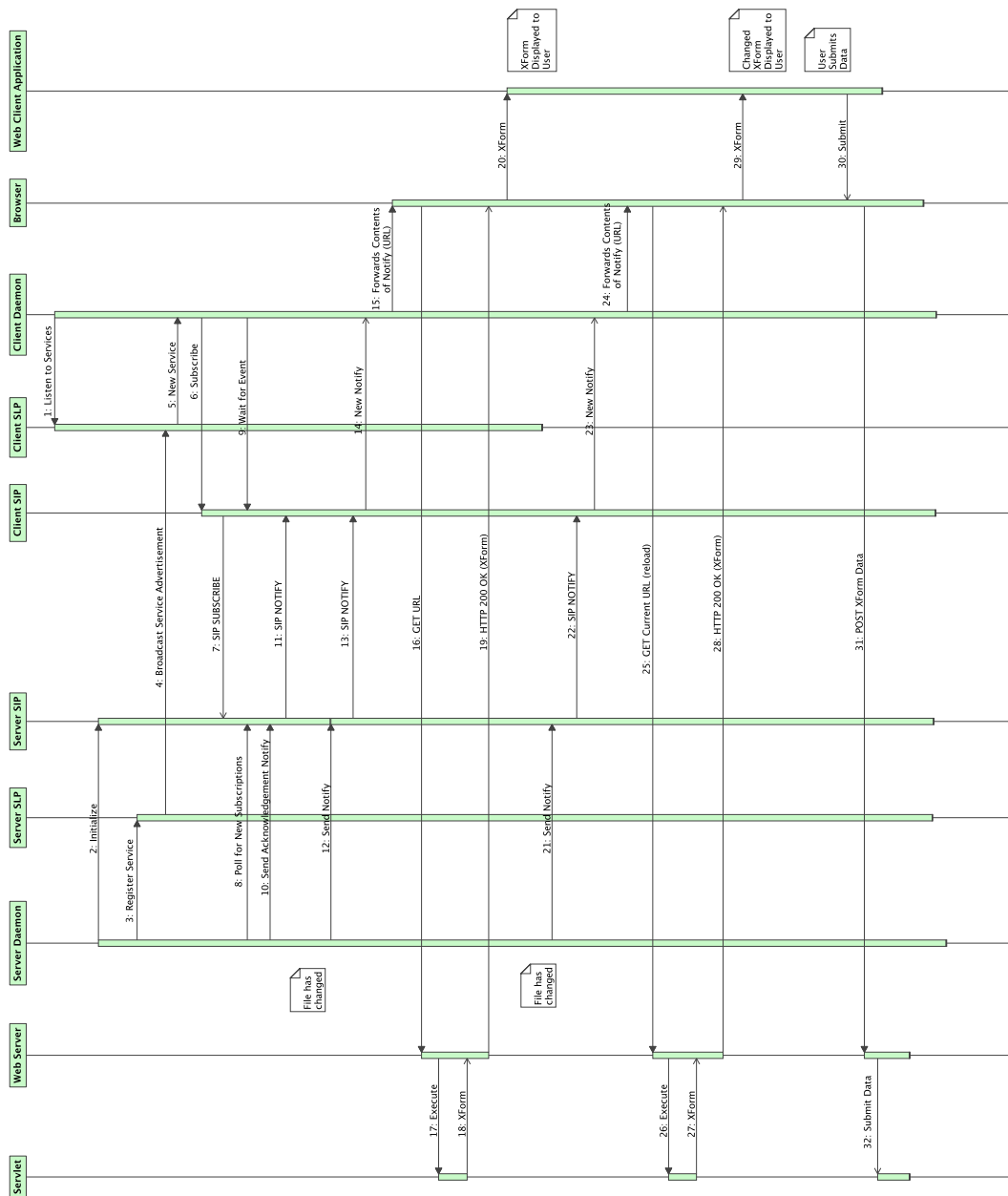


Figure 9: Operation of the Framework.

sets SLP to listen for service advertisements, and uses SIP to subscribe to the found service. Additionally it forwards notifications to the browser through a predefined socket. The notifications are then processed by the web client application that changes the user interface when required. A detailed description of the framework can be found in Section 7.

A user will interact with the web client application roughly the same way



as she would with a traditional web forms. The difference is that the data presented to her may change in real time. For example if the gate or time of her flight is changed the user interface will display the information immediately.

## 4 SOAP and SIP binding

### 4.1 Subscribing to notification service

In WeSAHMI SIP is used for registering to the services of a service provider, e.g., Finnair, and for receiving event notifications. Client must first subscribe to server to tell that it is online, and willing to receive event notifications. After subscription client receives notifications about events that it has subscribed for. At the moment notification carry information that some information has changed and browser should refresh the viewed page. Notifications could also carry more detailed information what has changed. Here is an example of SUBSCRIBE-message sent to server.

```
SUBSCRIBE sip:server.example.com SIP/2.0
To: <sip:server.example.com>
From: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@host.example.com
CSeq: 17766 SUBSCRIBE
Max-Forwards: 70
Event: resource-update
Accept: application/soap+xml
Contact: <sip:user@host.example.com>
Expires: 600
Service: finnair
Content-Length: 0
```

There is Event-header with value “resource-update” which tells the server that client is subscribing to receive notifications about changes in some resource. There is also a header field Service which specifies the service for which the client wants to receive notifications if there is several different type of services in same server.

### 4.2 Receiving notifications

After subscription server will send NOTIFY-messages back to the client when there is some change which the client needs to be notified, e.g., changes in flights. These NOTIFY-messages also have an Event-field which is set to value “resource-update” and Service-field has the same value as in the SUBSCRIBE-message. There is also a SOAP-body which has a more detailed description of event. Value of Content-Type-field is set to “application/soap+xml” to indicate that body contains a SOAP-message. Here is an example of NOTIFY-message.

```
NOTIFY sip:user@host.example.com SIP/2.0
From: <sip:server.example.com>;tag=ffd2
To: <sip:user@example.com>;tag=xfg9
Call-ID: 2010@host.example.com
```

```
Event: resource-update
Subscription-State: active;expires=599
Max-Forwards: 70
CSeq: 8775 NOTIFY
Contact: sip:server.example.com
Service: finnair
Content-Type: application/soap+xml
Content-Length: 242
```

<SOAP-body>

The body of the NOTIFY-message is a SOAP-message which can contain detailed information about what has changed in resource and client can update the user interface. For now the SOAP-message contains the URL-address where client retrieve the updated page to be shown in a browser. Here is an example of SOAP-body.

```
<SOAP-ENV:Envelope
  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
  SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
  <SOAP-ENV:Body>
    <m:ContentChanged xmlns:m="Some-URI">
      <service>finnair</service>
      <url>http://www.finnair.fi/service.html</url>
    </m:ContentChanged>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

In the body-part of the SOAP-message there is a ContentChanged-element which tells that something has changed and user interface needs to be refreshed. The service-element inside the ContentChanged-element tells the service for which this update notification belongs and the url-element the url-address where to retrieve the page.

eXosip and osip are the SIP-libraries used in the current implementation. eXosip is a higher level library built on top of osip which is low level SIP-library. This system could be implemented also using any other SIP-library which supports SUBSCRIBE/NOTIFY-messages.

## 5 Platform

The Wesahmi infrastructure is divided to client and server side implementations. The following sections discuss their current implementation in detail.

### 5.1 Client

There are three components on the client side, namely X-Smiles browser, SLP User Agent (SLP UA), and SIP client stack. They all interact with the different server components. The interaction is depicted in Figure 10. To simplify the Figure, all the server components are presented by a single entity in the Figure.

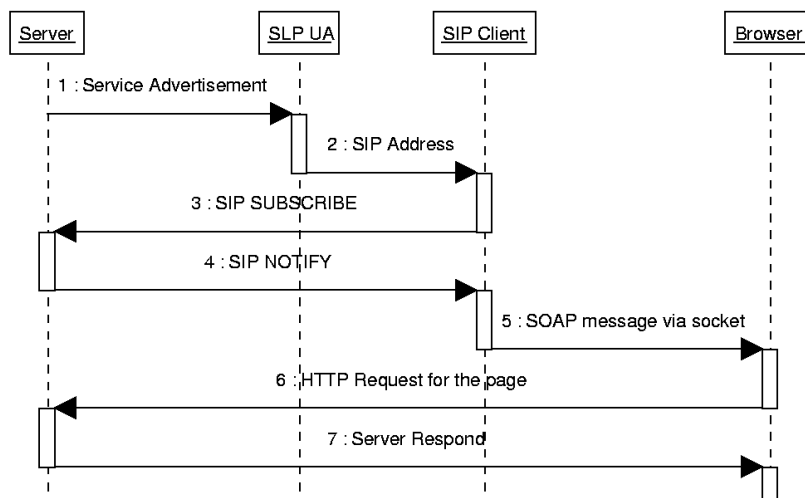


Figure 10: Interaction between the server and the client side components.

When SLP UA discovers a correct service (action number 1 in the Figure 10), it receives a SIP address, which it gives to the SIP client (2). The SIP client instantiates the session according to the address (3). During the session, SIP client receives NOTIFY messages (4), which contain SOAP messages. The system sends messages when it starts a new service with the user or updates ongoing session. The SIP client transmits the SOAP messages to the browser via a socket (5). There is a predefined socket port for communication between the SIP client and the browser. The browser parses the SOAP message and extracts the service id and the URL from it. Every session has its own view within the browser and is identified by the service id. If the SOAP message's service id matches with one of the ids of the browser views, then the browser reloads the current document of the view. Otherwise browser opens a new view for the service and fetches a document from the given URL. The browser fetches the documents via HTTP (6-7).

The implementation requirements are represented in Table 1. The Table discusses the requirements for both current implementation and the general case.

Table 1: Implementation Requirements.

<p><b>Current Implementation</b></p>	<ol style="list-style-type: none"> <li>1. The device must be able to run Java-based X-Smiles browser.</li> <li>2. The device must be able to run SLP client.</li> <li>3. The device must be able to run SIP client stack.</li> <li>4. The device must be able to run client daemon, which provides interfaces for other components.</li> <li>5. The device communicates with the server over WLAN.</li> </ol>
<p><b>Generalization</b></p>	<ol style="list-style-type: none"> <li>1. The browser must support XForms.</li> <li>2. The browser must be able to communicate with SLP client.</li> <li>3. The browser must be able to communicate with SIP client.</li> <li>4. Needs a wireless connection to the server (e.g., WLAN, GPRS, 3G)</li> </ol>

## 5.2 Server

The infrastructure of the server side implementation is illustrated in Figure 11. The servlets provide the actual service but are isolated from the service advertisement and client notification functions. They are managed by the server daemon that utilizes both SLP and SIP.

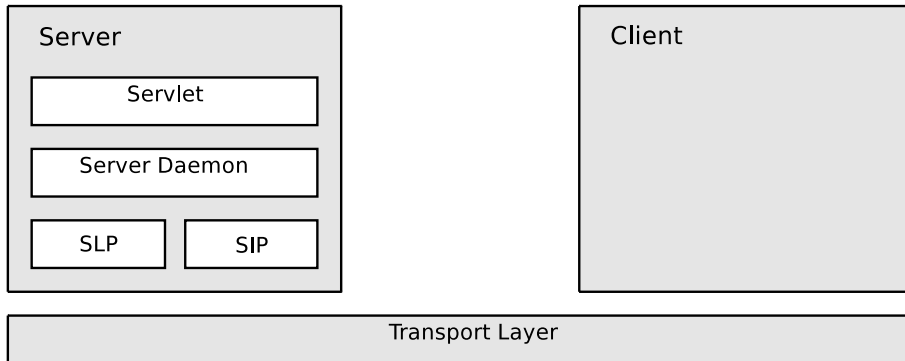


Figure 11: Structure of the server side implementation

The current implementation of server daemon sends broadcast advertisements of its SIP service. When it has received a SIP subscription from a client the SIP session ID is stored. All clients with open sessions are then informed when a change is made to a given file on the server. The operation of the server side operation is presented on a high level in Figure 12.

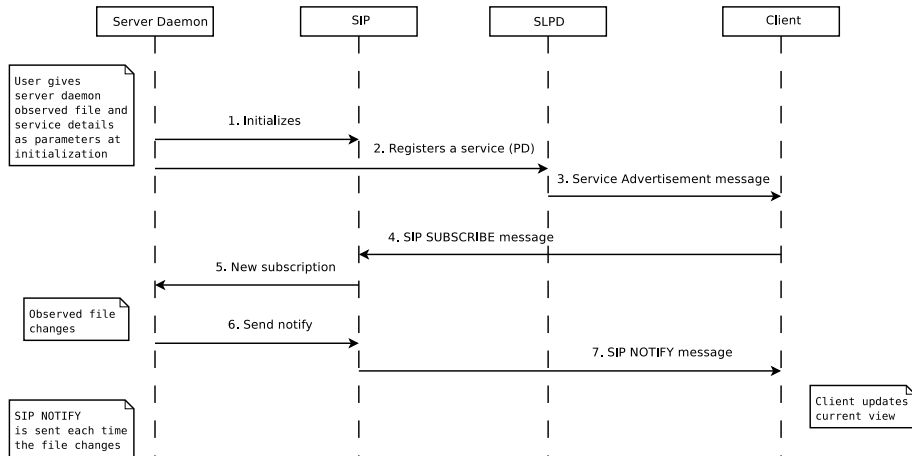


Figure 12: Server side operation on a high level.

First, the server daemon initializes SIP and sets `exosip` to listen for UDP traffic at given port (1.). Then it proceeds to register a local SIP service to the SD module with service ID `wesahmi`, service type `wesahmi.sip`, node's own IP address as URL, and attributes `service` and `event` (2.). SD module broadcasts the service advertisement messages (3.) and the server will listens for

SIP subscriptions (4.). It answers immediately with a notification to each new subscribe message that has correct service field (4.)(5.)(6.)(7.). Additionally the SIP session ID is stored. The server daemon will then check periodically if a given file has been changed. Each time it detects a change in the modification time of the file it will send a SIP notification message to all of them.

Table 2: Implementation Requirements.

<p><b>Current Implementation</b></p>	<ol style="list-style-type: none"> <li>1. Service has to communicate directly with the SLP and SIP. The server daemon takes care of these actions in our example implementation.</li> </ol>
<p><b>Generalization</b></p>	<ol style="list-style-type: none"> <li>1. The environment must keep track of all data associated with certain sessions, views, and clients.</li> <li>2. The environment must provide a notification interface. Clients that are associated with certain information are automatically notified when changes occur.</li> <li>3. The environment must support integration with traditional web technologies.</li> <li>4. The Service Discovery (SD) module, based on SLP, must provide bootstrapping service for the services using the platform.</li> <li>5. An API must be implemented to enable the java-based environment to communicate with C based SD module.</li> </ol>

## 6 Security

Security is an important part of distributed system requirements in today's world. Authentication and authorization play central roles in service provision in both fixed and wireless environments. It is expected that hybrid networks require novel security solutions, because the assumption on the existence of a dedicated and static security server must be relaxed for ad hoc and peer-to-peer operation.

The security considerations presented in this section are based on the airline scenario and we highlight the designs needed for secure operation in this environment. The airline scenario is an example of a single hop hybrid network. The security solutions needed for multi-hop hybrid networks are based on similar concepts, but this environment is more complex and requires further analysis. We focus on SIP security, but outline relevant Web service security technologies as well.

### 6.1 Requirements

In the airline case, a number of servers offer services to mobile and wireless clients. The servers may be located on the fixed network, or they may employ a single-hop wireless protocol, such as the wireless LAN protocol, in order to communicate with the clients. The interactions in this environment have three phases: First, there is the discovery phase. Second, there is a registration phase. Third, there is the communication phase, which is either client-driven or server-driven. In client-driven communication, the client requests information, which is provided by a server. In server-driven communication, a server pushes information to the client terminal.

The basic requirements of the security solution are as follows:

- Model for user/client identities and their federation.
- Authentication of clients at servers.
- Authorization and access control for authenticated users.
- Authentication of servers and control messages at client systems.
- Confidentiality of client interests and delivered content.
- Basic Denial-of-Service attack prevention both at clients and servers. Secure push functionality.

The non-functional requirements are as follows:

- Identification and elimination of performance bottlenecks.
- The employed solutions should be energy-efficient.
- The solutions should impose minimal state requirements for nodes.
- The solutions should integrate well with the X-Smiles browser and the SIP and SOAP security models.
- The solution should work also in an environment where NAT is deployed

In the hybrid environment a malicious entity may attempt to disrupt a service. Typical attacks include registration hijacking, server impersonation, message dropping, and message tampering.

Given the closed nature of the airline scenario and that an out-of-band trust mechanism is employed, it is difficult for an attacker to impersonate a server, hijack sessions, replay messages, tear down sessions (inject BYEs), or tamper messages. These attacks become difficult when the service revolves around a known service, whose certificate is trusted, and the well-known authentication, encryption, sequence number techniques are used.

However, an attacker may disrupt the communication in various ways by dropping messages, injecting bogus messages, and simply flooding the network. SIP creates a number of opportunities for distributed DoS attacks. Especially, the SIP technique of forking may result in a message being replicated to multiple recipients.

The Wesahmi system must be able to cope with a number of security issues. The following list illustrates the main security issues and typical solutions. Namely, DoS attacks, eavesdropping, message spoofing, message replaying, and message integrity compromises.

**Denial-of-Service (DoS) attacks** Firewalls/NATs/packet filtering offer some resistance. Efficient SIP message authentication prevents spoofed messages

**Traffic eavesdropping** Data encryption

**Message and packet spoofing** Sender verification

**Replay attacks** Encrypt and sequence number messages. SIP CSeq and Call-ID headers

**Message integrity** Authenticate messages and perform integrity checking

## 6.2 Building Blocks for Security

The basic security building blocks are provided by the different standardization organizations, namely W3C and IETF. A key observation is that security is needed on multiple layers. Basic network and transport-layer security ensure data confidentiality and they may also be used for mutual authentication. Session and application layer security is needed for environments with multiple security domains, for example, environments with gateways and different service providers.

### 6.2.1 End-to-End Measures

Transport Layer Security (TLS) [7] provides session-layer security with mutual certificate-based authentication. IP Security (IPsec) and Internet Key Exchange (IKE) [13] may be used to set up security association for network layer security. The Host Identity Protocol defines a namespace for hosts that is based on public keys and integrates this new namespace with the transport layer APIs and network layer security.



### 6.2.2 SIP Security

SIP security solutions leverage S/MIME [24], digest authentication, and transport-layer security. The digest mechanism is the SIP baseline technique for authentication. S/MIME encryption requires that the public key (X.509 certificate) of the recipient is known. S/MIME may also be used to encrypt the payload of the Session Description Protocol. The main security mechanisms for SIP are as follows:

**S/MIME** For encrypting message payloads. The public key of the recipient must be known.

**SIPS URI** Tight coupling between SIP and TLS. Applications and proxies need to be TLS aware. Integrates well with current browsing technologies. Must be applied on a hop-by-hop basis with SIP intermediaries.

**IPsec** Tight coupling not required. IKE key agreement protocol is heavy. Must be applied on a hop-by-hop basis with SIP intermediaries. NAT/firewall traversal issues.

**SIP over TLS daemon** A single TLS session is used to tunnel SIP messages between a terminal and the next hop SIP intermediary.

S/MIME is used to encrypt message payloads. It is useful for end-to-end secrecy. RFC 3261 defines the SIPS URI that forces the communication over a set of TLS connections. This provides end-to-end secrecy given that the intermediaries are trusted. For this security mechanism, the applications need to be aware of the SIPS URI. According to RFC 3261, in a Secure SIP (SIPS) session, the SIP user agent contacts the SIP proxy server and requests a TLS session. The proxy server then responds with a public certificate. The user agent and the proxy exchange session keys. If there are multiple hops, the proxy then contacts the next hop until the final destination is reached.

IPsec does not require a tight coupling between applications and the security solution; however, in order to support SIP message forwarding, it must be applied on a hop-by-hop basis rather than end-to-end. IPSec can be seen as a heavier protocol than the session layer SSL due to complex key agreement and there are also a number of NAT and firewall traversal issues.

The default SIP security mechanism for Wesahmi is TLS for the last hop between the gateway and the trusted service domain. IPsec can also be used, but it is less portable due to the coupling with network layer mechanisms. It is expected that the Host Identity Protocol would also be useful in this case due to its simple authenticated key agreement scheme and mobility support. TLS support can be realized using at least three ways:

- **SIPS URI in applications.** The SIP stack handles the TLS connections and authentication. RFC 3261 states that TLS is used until the message reaches the SIP entity responsible for the domain portion of the destination URI. Inside the destination domain the use of TLS is up to the local policy. The SIP framework does not guarantee true end-to-end security and the SIPS scheme assumes transitive trust for intermediaries. A limitation of this approach is that SIP clients must understand SIPS URIs and support TLS. The specification mandates that a resource described using a SIPS

URI cannot be downgraded to a SIP URI. A SIP URI can be upgraded to a SIPS URI.

- Use transport URI parameters in Contacts in REGISTER. This is not recommended by the SIP guidelines. The use of "transport=tls" is deprecated and the if the SIP registrar is co-located with the proxy it can infer if TLS is used. In addition, TLS can be specified as the desired transport protocol within a Via header field value or a SIP-URI. TLS is most suited to architectures in which hop-by-hop security is required between hosts with no pre-existing trust association. This may require that a TLS server is present on the terminal for incoming connections.
- A client initiated communication channel using TLS. This requires connection management at the client and at the edge proxy, but does not require a server on the terminal.

We distinguish between client-initiated and server-initiated connections. In order for a client to be reachable, it must have open listening TCP or UDP ports. On some systems, open ports are not supported. Client-initiated connections open a persistent connection with a server and the client can receive data from the server through the connection without opening a server socket. We expect that the terminal does not have TLS server functionality so for TLS a client-initiated connection is needed. IPSec is realized on a lower layer than transport layer so it may be used independently of how the client-server communication is implemented on the higher layers.

On-going work at IETF is looking at client-initiated connections [17]. The key idea of the specification is to re-use the connection that was used to send the REGISTER request. This connection can be a bidirectional stream of UDP datagrams, a TCP connection, or a some other type of transport protocol. It is the responsibility of the UA to maintain connectivity. The UA may also employ multiple flows to the proxy or registrar. In addition, a keep alive mechanism is included so that the UA can detect when a flow has failed.

The proxy does not need to be collocated with the registrar. If they are distributed the edge proxy includes a Path header [31] with a unique flow identifier to any REGISTER messages. Requests to the UA are routed through the edge proxy. The flow identifier allows the edge proxy to find the correct flow for messages.

None of the above schemes guarantee end-to-end integrity or secrecy if intermediaries cannot be trusted. It is stated in RFC 3261 that S/MIME may also be used by the originating UAC to ensure that the original form of the To header field is carried end-to-end. Another approach is to use the SIP Identity mechanism defined in [21]. SIP Identity creates a signed identity digest which includes the AOR of the sender (from the From header) and the AOR of the original destination (from the To header). In order to work in practice, the vouching domain's certificate has to be publicly available through some secure channel. This means that the vouching domain's HTTPS server certificate should be signed by a widely known certificate authority. Hence, SIP identity mechanism is basically a trusted third party solution.

To provide flexibility in choosing different security schemes, Arkko et al. [2] have defined a negotiation mechanisms between UA and its first-hop SIP entity.

In addition, RFC 3263 [27] defines a mechanism for locating TLS capable servers using DNS NAPTR (Naming Authority Pointer).

As described above SIP supports hop-to-hop security with TLS and end-to-end security using S/MIME. However, one limitation with S/MIME and public key cryptography is the reliance on a certificate distribution infrastructure. Jennings et al. [18] propose a new service combined with SIP Identity [21] specification for handling certificate distribution in a manner that does not require well known certificate authority while still binding the user's identity to the certificate. The specification handles two cases. In the first case the service stores just public certificates. Second, the service could store also credentials. This would be advantage, when (mobile) devices with limited memory are used. However, the system is highly dependant on trusting the operators of the service and that the system is not compromised.

### 6.2.3 Web Services Security

W3C has a number of XML-related security specifications. The base specifications are the XML Encryption and XML Signature, which allow flexible encryption and signing of elements in XML documents. The signature operation is more difficult of the two, because of challenges in XML document canonicalization. These two specification may be used with the SOAP protocol, for example, for flexible header-based security.

The WS-Security specification defines the SOAP security header [20]. SOAP messages can contain security tokens with authentication information. This kind of support is needed for coping with multiple security contexts.

A security token represents a set of claims. In the WS-Security model a trusted third party, the Security Token Service, issues these tokens. A security token may be self-generated, as in the case of username/password, or it may be given by a trusted third party.

The security tokens should be signed and encrypted. In this case, the WS-Security model prevents unauthorizes accesses and modifications also in the presence of untrusted intermediaries.

A standard Web services interface is needed for creating, exchanging, and validating security tokens issued by other domains. This is specified in WS-Trust [16]. In addition, a set of concrete security policy documents are needed that allow sites and services to document their security requirements. A security policy might require that a message should be encrypted using a specific algorithm and have a certain key length.

There are two interaction models for establishing trust. First, we have the pull model and then the push model. In the pull model, the receiver contacts a security token service when it receives a token. In the push model, the sender contacts the token service and obtains a signed token. In this latter case the receiver does not have to contact the security service. The Kerberos Ticket Granting Ticket (TGT) is an example of the latter strategy. The push model is more efficient in terms of network operation, but the signed tokens may be revoked. The revocation requires that the token service is contacted at some point.

Using asymmetric cryptography in each message is computationally demanding. The WS-SecureConversation [15] specification defines a session-key-based

model for WS-Security. The model is based on Security Context Tokens issued by servers or generated by the requesters. The SCT contains a shared secret.

Each Web service endpoint implements a trust engine that understands the WS-Security and WS-Trust model [16]. For the hybrid network environment, each peer must implement a trust engine and be able to process security tokens.

#### 6.2.4 Identity Federation

WS-Federation defines a federated identity and mechanisms to broker and federate identity, trust, and claims about them [14]. Single-sign-on means the ability to use federated services without reauthentication by signing into one of the federations. In addition to WS-Federation, the Open Mobile Alliance (OMA) has defined a system for identity-federation [1].

Peterson et al. [21] have proposed enhancements for SIP identity management in interdomain context. Their proposal defines a mechanism for authenticating the sender of SIP messages by introducing two new SIP header fields: Identity and Identity-Info. This draft is used as a base for a few other internet drafts such as SIP SAML Profile and Binding [29], Trait-based Authorization Requirements for SIP [22], SPAM Prevention using SAML [28] and Certificate Management Service for the SIP [18].

### 6.3 Security Specification

#### 6.3.1 Overview

The system model consists of terminals, gateways, and services. Gateways advertise service access and services using SLP (passive mode). Terminals use gateways to access services, and the gateways allow content to be pushed to terminals. The security challenge is solved by requiring a secure connection between terminals and gateways. Gateways and services may or may not have a secure connection, depending on the environment and requirements.

The gateway performs the following functions:

- SIP proxy server, and optionally redirect server and registrar server (IETF RFC 3261).
- An incoming and outgoing proxy, providing integrity checks for malformed SIP messages, ensuring that only correctly formatted messages are forwarded.
- NAT and firewall traversal for SIP messages.
- Limited SIP spam prevention (through mutual authentication).

We assume that communication in the SIP domain is trusted and monitored. Therefore the weak point of the system is the final wireless hop and the gateway. The gateway can provide privacy support for both clients in the ad hoc domain and in the SIP domain by concealing, obfuscating, and encrypting SIP message headers.

For the single service provider case without call control, it is expected that general SIP anonymity support is not needed. According to RFC 3323 user agents should indicate a Privacy header when network-provided privacy is required.

Figure 13 presents an overview of the interactions in the airline case. In the first phase, the terminal receives an SLP advertisement from the gateway. The advertisement message is signed. The client authenticates the service using a pre-installed certificate (2).

Then, the client accesses the service URI that is specified in the advertisement (3). This may be standard web browsing or multi-hop message-based interaction. In the former case, TLS or IPSec is used for security. In the latter case, either SIP (S/MIME) or WS-Security needs to be used for security.

The service authenticates and authorizes the client. Authentication may be performed through challenge/response, client signature, or a security token. The service access results in the desired content or an authentication failure (5).

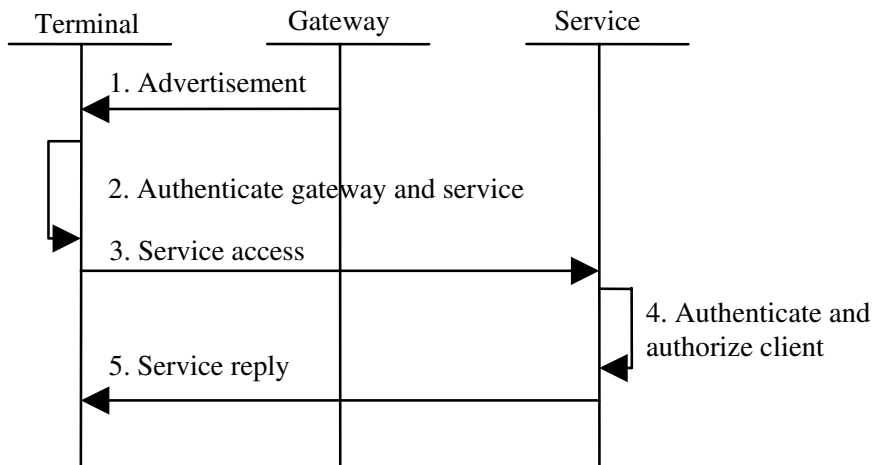


Figure 13: Overview of interactions.

The secure connections can be realized using different techniques, for example using network level IPSec security associations, transport level SSL/TLS, or session-based mechanisms such as SIP security mechanisms. Each of these is possible in our environment, but we propose to leverage session-based security mechanisms in order to keep the system flexible.

The basic design of the Wesahmi SIP security is as follows:

- Bootstrap trusted identities with an identity provider. Either symmetric keys or public key cryptography. Gateway certificates issued also by a trusted third party.
- Communication is divided into control and content channels. Control channel consists of SIP signalling. Security is provided explicitly for control channel.
- Use TLS for securing wireless hop (client/gateway).
- Use client-initiated connections to maintain client-gateway connectivity. This does not require server functionality on the client and supports NATs and firewalls better than IPsec. Keep alives are used to detect flow failures.

- Use S/MIME attachments or SIP Identity Digests for end-to-end security and prevent message tampering.

The following pertain also to the security specification but are not considered for implementation:

- Support SIP privacy options at the gateway to improve message privacy.
- SIPS URI can be used for better end-to-end security.

### 6.3.2 Bootstrapping Trust

A key design choice in the security specification is how trust between clients and servers is bootstrapped. Clearly, a solution is needed to enable the mutual authentication of these different systems. In the first prototype, trust is established through digital certificates issued by a trusted third party. This is the conventional way of enabling security in web browsers.

Currently, the requirement for end-user certifications is seen as a serious scalability limitation in a distributed system. End-user certificates are difficult to provide and maintain on a global scale. Self-signed certificates avoid this scalability limitation, but are prone to man-in-the-middle attacks.

One key assumption that we need to make is whether or not random encounters should be supported. Man-in-the-Middle (MiM) attacks cannot be prevented unless trust is bootstrapped somehow. For some scenarios, ssh-like security is enough. This type of approach can be used to ensure future trust in an entity.

The memory restrictions on mobile devices limit the number of stored certificates. A Credential Service is proposed to discover the certificates of other SIP users and as well as store own certificates or even private keys remotely.

The following three trust observations form the base of the proposed security solution.

- Service and gateway certificates are shared by all entities. Certificates are issued by a trusted third party.
- Client terminals have a self-signed certificate or a certificate issued by a trusted third party. In the former case, application-level interaction is required to verify identity. In the latter case, these are known to the gateway and the services. In both cases the client certificate is used for authentication and message security.
- For message intensive operation, a temporal session key may be derived using asymmetric crypto to improve performance.

### 6.3.3 Secure Push

Figure 14 illustrates secure push. A service sends a push message to the client (1). The client is identified using a SIP URI. The server should have previously verified using some mechanism that this SIP URI belongs to the intended recipient.

The client verifies the push message (2). Since asymmetric crypto is computationally expensive it is also possible to use HMAC here to drop bogus messages. The message signature is checked and if the check fails the message

is silently dropped. Otherwise, the client terminal accepts the message and it is processed according to the local message processing rules.

In the airline scenario, the push message results in a web resource being used (3). This entails also some level of client authentication and authorization (4) at the server. Finally, content is delivered for legitimate client systems (5).

Push messages are handled by the security system and they are passed to higher levels only after their authenticity has been established.

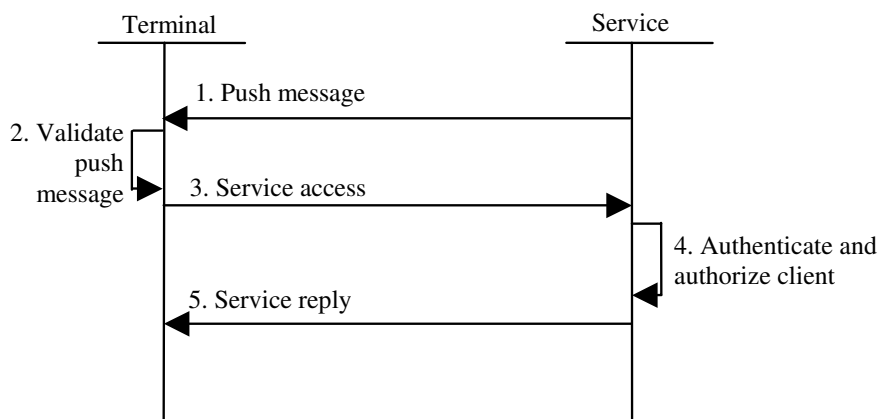


Figure 14: Secure push

Figure 15 gives an overview of the secure push system model and the required components on each of the three main entities, namely the terminal, gateway, and push service. Each gateway keeps a location server up-to-date the current users in its domain. This supports multi-mode delivery, in which the current operating modes and the reachability of a user are determined, and then a suitable push protocol is employed.

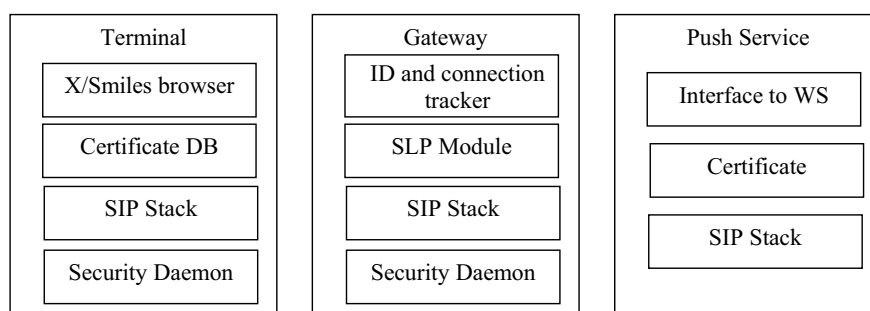


Figure 15: Secure push system model

## 6.4 Security Domains

Figure 16 illustrates a single Wesahmi service domain. The gateway bridges SIP clients in the single-hop ad hoc network with the service domain. The service domain consists of the SIP entities, the identity provider service (IdP), and a number of services. The gateway consults the IdP in order to authenticate and authorize clients. The gateway also updates the location server and performs a SIP registration with the local SIP proxy on behalf of the client. After this process, the SIP client is reachable through the gateway and messages may be pushed to the client by the services.

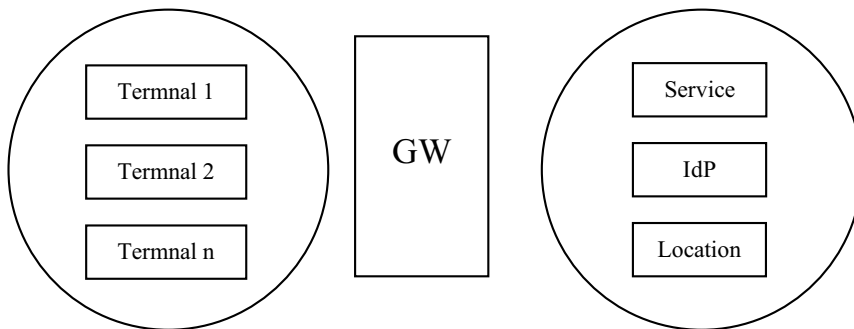


Figure 16: Single security domain

Figure 17 illustrates the Wesahmi service model with multiple domains and federated identity providers. Each SIP client has one home identity provider, which was used to bootstrap and verify the identity of the client. When the client contacts a gateway at a different domain than the home domain, the local IdP contacts the home IdP in order to authenticate and authorize the client. After credentials have been established, the client is registered with the local SIP proxy and the location databases are updated to reflect the current location of the client.

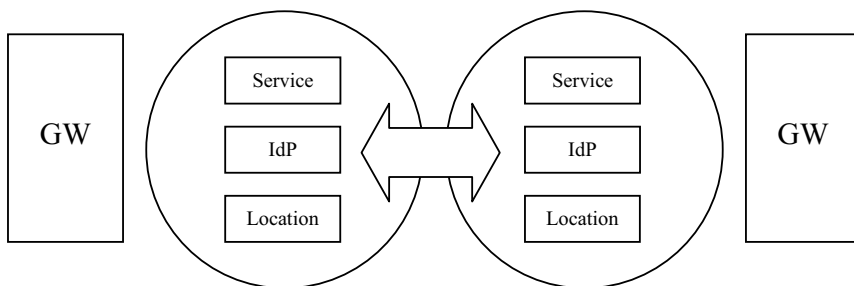


Figure 17: Two security domains



## 6.5 Privacy

User privacy is one of the key requirements for the Wesahmi system. Leakage of confidential information must be prevented and it must not be possible to track the location and behaviour of certain hosts. Privacy requires confidentiality and integrity for messages.

RFC 3323 defines a privacy mechanism for SIP that prevents the dissemination of personal identity information. A new logical privacy service role is defined for intermediaries. A user can request particular functions from a privacy service.

SIP identities are commonly carried in the form of SIP URIs and optional display names. These identities are typically found in the To and From header fields. There are also other fields that have privacy implications. Namely, the Contact header field.

Simply encrypting all fields and data with potential privacy implications does not suffice, because SIP agents and intermediaries must be able to forward and route SIP messages. Moreover, proxy servers may add headers of their own, such as Record-Route and Via headers, which can have potential privacy risks.

The baseline SIP specification supports some level of user controlled privacy. For example, the From header in a request may be populated with an anonymous value. A SIP body may be encrypted end-to-end thus concealing the contents from intermediaries. Header information can be concealed from intermediaries by placing it in encapsulated 'message/sip' S/MIME bodies [10].

The SIP client-initiated connection mechanism discussed previously [17] has privacy enhancing properties. The mechanism allows inbound traffic from outside to an authenticated UA. The UA can be behind a NAT or firewall. It follows that UA does not necessarily need to have a globally routable IP address or hostname.

## 6.6 Browser Integration

The main integration points with the X/Smiles browser are the following:

- Securing SIP signalling from X/Smiles browser. This is accomplished by the underlying security daemon and SIP stack.
- Certificate storage.

It is expected that the first point can be realized by intercepting any SIP messages from the browser in the SIP stack and then using the appropriate connection. Privacy extensions may also be applied at this point.

For the second point, the expectation is that the X/Smiles browser and the underlying security daemon use the same certificate storage. This is mainly motivated by the fact that both need access to the user's key material when initiating TLS connections and signing messages.

## 6.7 Putting it Together

Figure 18 illustrates the Wesahmi security architecture. Initial bootstrap is used to create identities for users and gateways (1). The identity provider (IdP) is trusted by all entities. After the bootstrap, the client system starts a secured

session with a gateway (3). This session is typically started after receiving an SLP advertisement and checking that the advertisement is valid (2).

The secure channel is client-initiated and used to open ports for the client in the gateway and to transfer SIP messages. Hence, the secure channel is the SIP control plane. The baseline solution uses TLS and client initiated connections [17]. SIP REGISTER message is sent to the gateway in order to establish the secure channel. The registrar is updated at this point to reflect the current gateway (4). The registrar may be collocated with the gateway. After the secure channel has been setup, the channel is kept open by periodic keep alives. When a keep alive fails, the channel is closed and port access for the terminal is blocked.

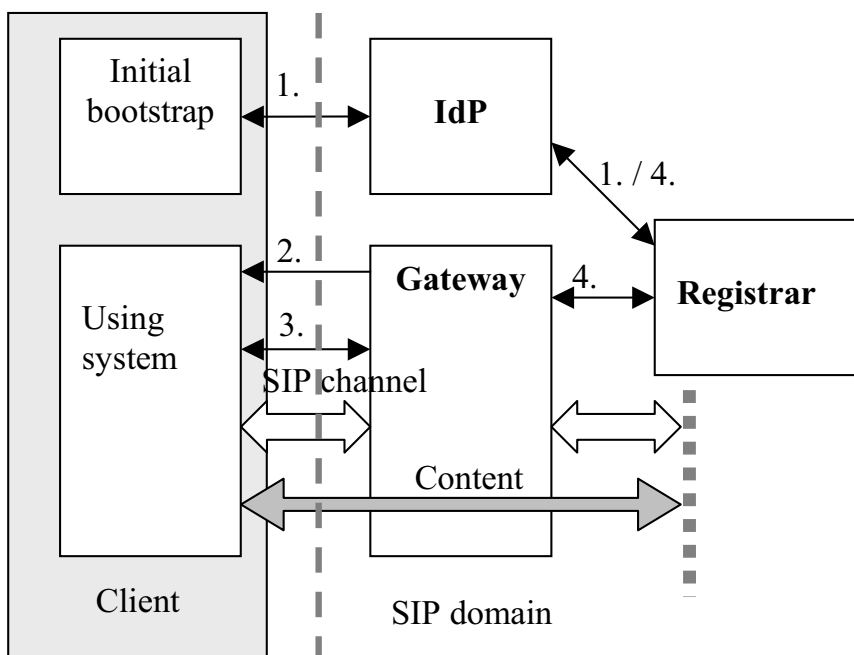


Figure 18: Putting it together

## 6.8 Implementation Plans

Implementation work for the second prototype focuses on securing the SLP advertisements, mutual authentication of the client and server with Web browsing techniques, and securing SIP push messages. The implementation of the client authentication mechanism is an important part of this work. The simplest strategy is to use TLS and username/password for browsing and server certificate for SIP push messages. A more advanced implementation uses a client certificate for authentication, and security tokens for messaging.

## 7 Detailed application-level description

### 7.1 User Story: Check-in

When a user enters the airport she switches on her laptop, and starts the client daemon and X-Smiles browser. After these actions the client daemon sets up its SD module in order to receive service advertisements of a check-in service provided by the airport server. After the daemon has received such advertisement it will use SIP to subscribe to this service. The server will from now on send a SIP notify message to the client whenever it notices that a certain file has changed. In our current implementation the notify message just triggers a page refresh operation in the browser. Now the user is able to go through the check-in procedure and receive updated information on every form.

This implementation is a basis for stepwise design and development of our platform. It has given us essential insight into issues associated with the new interaction model and its implementation details.

#### Bootstrap

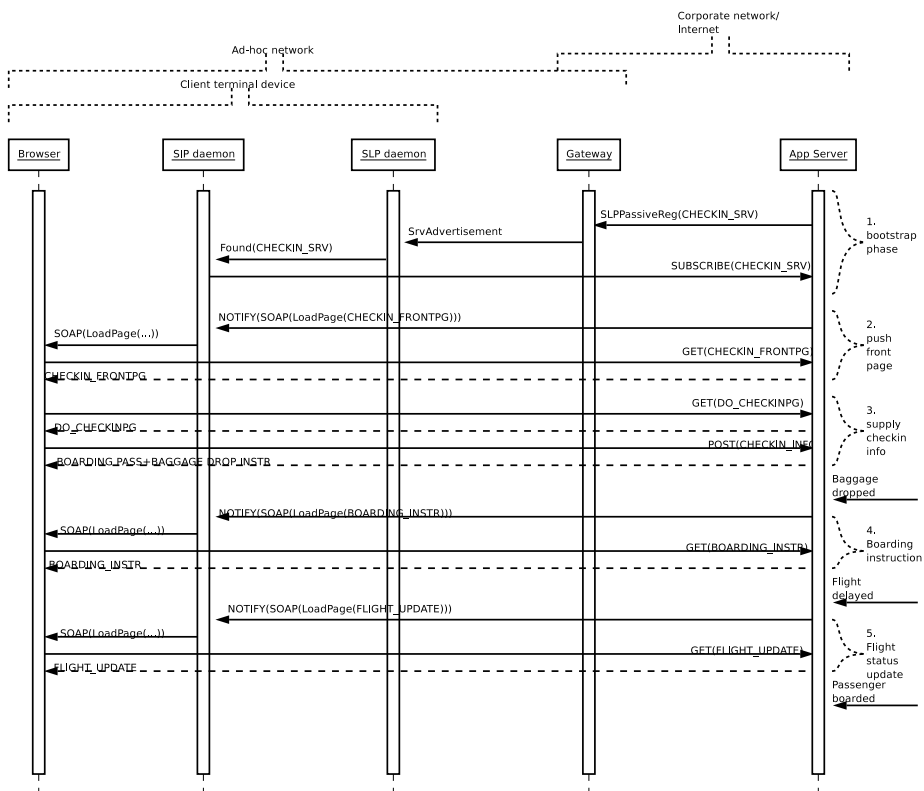


Figure 19: Sequence diagram of application behavior.

Figure 19 gives an overview of application functionality in the flight check-in

service case. Application execution starts with bootstrap phase (1). The application server requests that the gateway starts to advertise the check-in service (message  $SLPPassiveReg(CHECKIN\_SRV)$ , where  $CHECKIN\_SRV$  stands for the URL and service parameters of the service). The Gateway will start to broadcast passive service advertisements in the ad-hoc network. Eventually the mobile device of a passenger will receive an advertisement ( $SrvAdvertisement$ ). The SLP daemon process in the device forwards the service URL to the SIP daemon process that sends a  $SUBSCRIBE$  message to the application server.

### First Notification

After the bootstrap, the application server can push the front page of the check-in service to the passenger (2). This is accomplished using a SIP  $NOTIFY$  message that carries a SOAP message instructing the browser to load the check-in front page. The page asks the passenger whether he wants to check in on his flight and is presented in Figure 20. The  $NOTIFY$  message is received by the SIP daemon process which forwards the SOAP body to the browser.

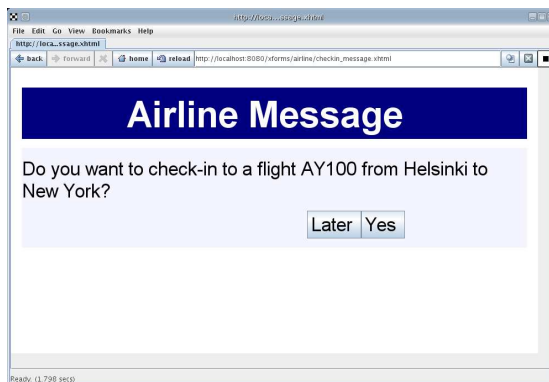


Figure 20: Screenshot of the Checkin Message.

### Normal User Interaction

When the user selects the positive option, the actual check-in page is loaded (3). The page is presented in Figure 21. Here the passenger supplies the relevant information, such as number of baggages. In return, the user receives his electronic boarding pass together with instructions on where to drop baggage.

When the application server receives a notification that the baggage has been dropped<sup>1</sup>, boarding instructions are pushed to the passenger (4). This includes information on how to find the security check point and the gate. The actual page is presented in Figure 22.

### Update Notification

Later the system receives information that the flight has been delayed. A flight status update is thus pushed to the passenger (5). The pre-flight interaction

<sup>1</sup>The availability of such information is not known. This notification is not essential, but makes the interaction smoother.

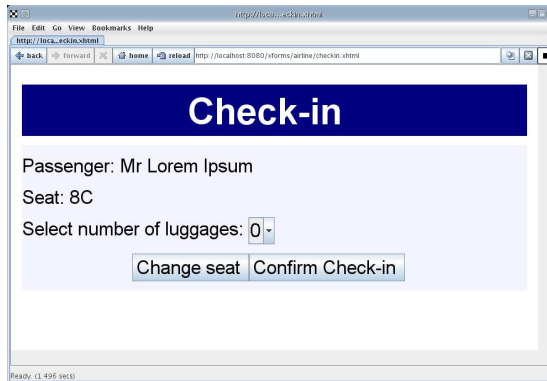


Figure 21: Screenshot of the Check-in Page.

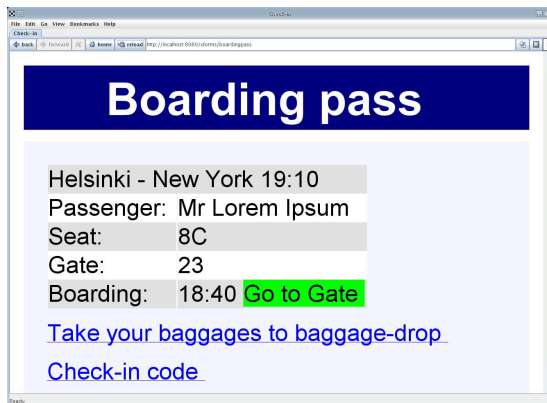


Figure 22: Screenshot of the Boarding pass.

ends when the application server receives a notification that the passenger has boarded.

The interaction depicted in Figure 19 corresponds to phase 1 functionality of the WeSAHMI infrastructure. Later in the project it will e.g. be possible to send partial view updates to user terminals.

## 7.2 Pilot Environment

The pilot environment consists of one web server and two clients all running on Linux laptops. Figure 23 shows the setup. Webserver laptop is also responsible of advertising services (e.g. check-in service) to clients and sending update notifications to clients when there is change in information that client is interested (e.g. flight is delayed). Clients communicate with server using IEEE 802.11b WLAN in ad-hoc mode.

On client machine there is x-smiles browser which shows the graphical user interface to the user of the service. On the web server machine there is a Apache web server and an application that keeps track if there is changes in information like flight schedules. When there is change the applications is responsible of informing clients that are interested about changes. This application also takes

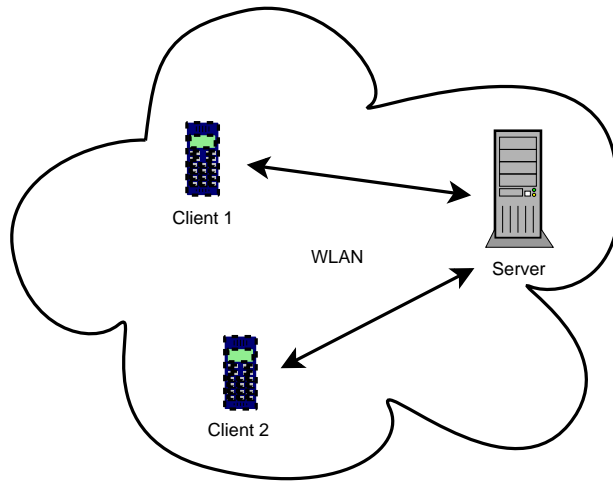


Figure 23: Pilot environment.

care of advertising service to clients.

## 8 Future Work

This section discusses features that are to be implemented in the next phase of Wesahmi project. First, Section 8.1 describes a “Pushlet” environment that abstracts most common tasks related to the interaction model. Then, Section 8.2 presents an interface that is used to enable interaction between Java -based “Pushlet”environment and the C -based server daemon. Then, Section 8.3 describes methods that can be used to integrate SIP with the XSmiles browser. Finally, Section 8.4 addresses future work for the security subproject.

### 8.1 Pushlet Environment

To make writing server-initiated services easier, a programming model is required that simplifies or hides tasks related to the interaction model on both client and server sides. The term "pushlet" has been employed to refer to the general idea of a managed runtime environment for push-based sessions.

On the client side, the browser needs to provide applications with a runtime environment that transparently performs e.g. SIP SUBSCRIBE renewals as long as a given UI view is active. At the moment we think that it is natural to associate notification content types with UI views. The runtime should also include an application programming interface (API) for passing notification events from the infrastructure to the application.

On the client side there should be a session management interface that provides the application with a means of sending the necessary updates associated with the active UI views and clients. It should therefore internally store some sort of information about the currently active views (based on SUBSCRIBE events coming from the clients).

Ideally, a mechanism for high-level (declarative) specification of trigger conditions form notifications should be included. For example, a change in a certain EJB or object (e.g. representing in Finnair flight) would trigger the sending of a notification to all clients meeting certain criteria (e.g. that they have a ticket for this flight) and that have a certain UI view active (e.g. the check-in view).

Preliminary the sturcture of the environment is illustrated in Figure 24. The Controller instance initializes instances of Session. Each Session has atleast one associated instance of Client and may also have multiple instances of View. Target is to also eventually support incremental updates on displayed UI views.

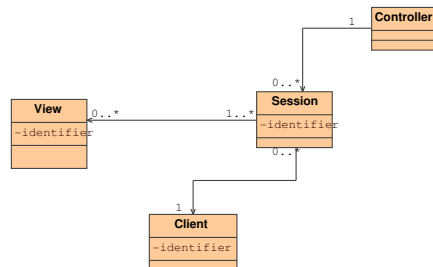


Figure 24: High Level Structure of the Environment.

Operation of the environment is illustrated in Figure 25. After the environ-

ment is informed by the SIP stack that a new subscription has arrived (5.) it will create a new Session instance (8.) and stores it in a dynamic data structure. The Session will in turn create instances of View (9.) and Client (10.) and store them in dynamic data structures. The controller will also trigger SIP (11.) to send a NOTIFY message to the client terminal (12.). This NOTIFY will contain address to a first page of the service. When resource associated with the Session changes it is informed and will trigger the controller (13.) to send a notify message (14.) through an associated SIP session (15.). When client sends a SIP SUBSCRIBE message with zero timeout (16.), the Controller is informed that a subscription is cancelled (17.). It will then tell corresponding Session to close the associated View (18.)(19.). When the SIP stack receives a new SUBSCRIBE message (20.), it will inform Controller that a new subscription has arrived (21.). It then instructs corresponding Session to open a new View (22.)(23.). The Session may then again access the Client data (24.).



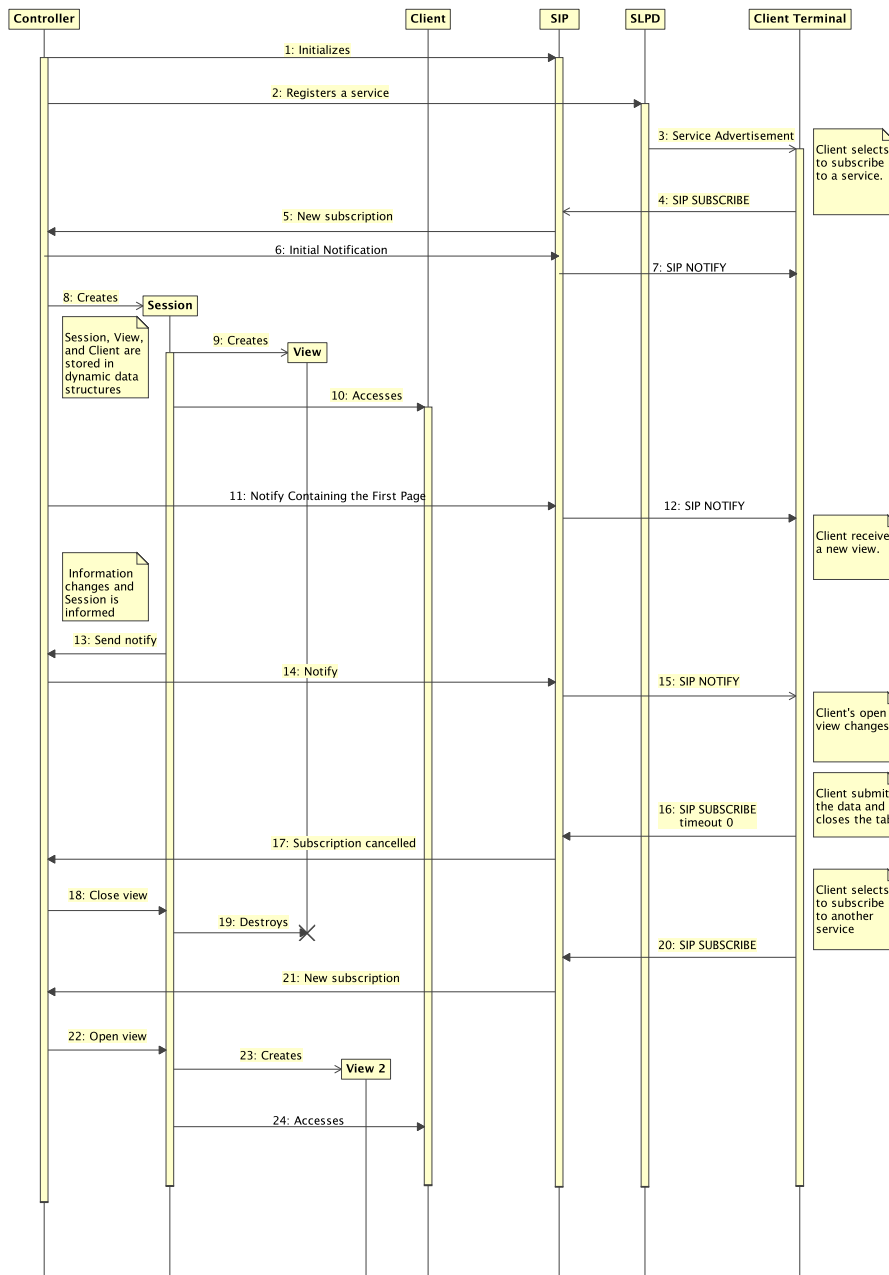


Figure 25: Operation of the Environment in a Check-in Case.

## 8.2 Pushlet environment and SLP integration

Since the Pushlet environment will be Java-based and we are using C implementation of SLP as basis for our Service Discovery (SD) module, they will need an additional adaptation layer between them. This layer could be implemented either with SOAP or as a Java Native Interface (JNI).

The benefit of using SOAP would be that the SOAP skeleton can be implemented independently of the stub. While the stub is tied to C language the skeleton can be implemented with Java that enables it to be directly used by the Pushlet environment. The structure of this alternative is illustrated in Figure 26. The SOAP SD API stub could be implemented using gSOAP which is an open source framework for developing C-based web services. The SOAP SD API skeleton could then be implemented using Axis developed by Apache [3]. It is a framework for developing Java-based web services. Axis provides a tool that enables skeleton generation based on WSDL description of the original stub.

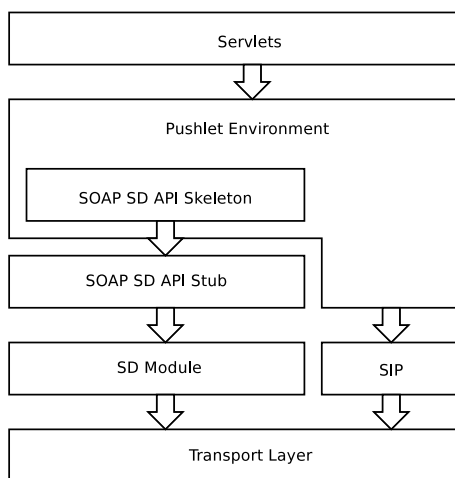


Figure 26: Structure of Server-side Implementation.

JNI might provide a more efficient and compact solution but would be less adaptive and limit the systems re-usability by binding it to Java. The structure of this alternative is illustrated in Figure 27.

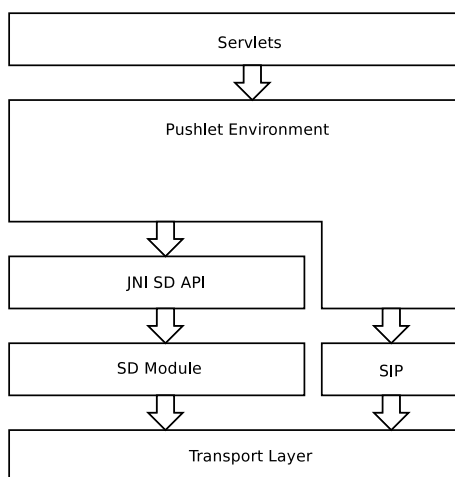


Figure 27: Structure of Server-side Implementation.

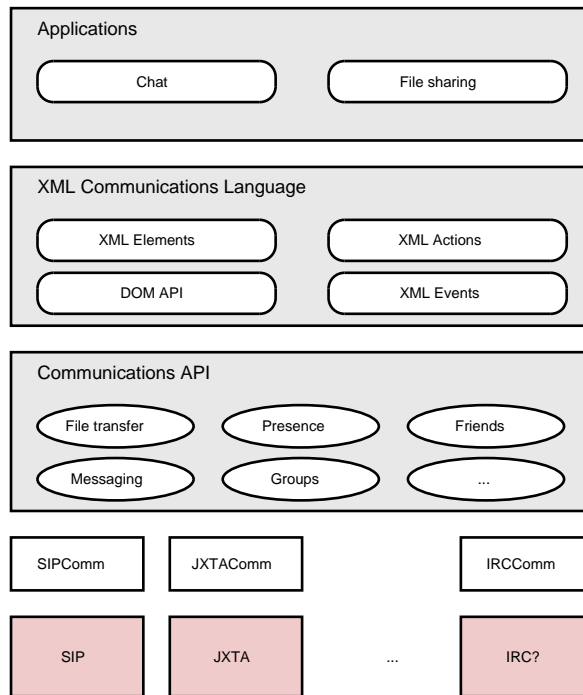


Figure 28: Overview of the X-Smiles Communication API.

### 8.3 X-Smiles and SIP integration

The current client side implementation was intended to give an general idea of the functionality of the final system. However, it lacks some features, which will exist in the final system. For instance, the browser does not utilize SIP at all. That is, it does not initialize the sessions nor send or receive messages. Instead, the SIP client maintains the sessions and uses a socket to deliver a content of the messages for the browser.

X-Smiles already has Communication API through which it can be integrated with the SIP protocol. The overview of the Communication API is depicted in the Figure 28. Characteristics of the several communication protocols are abstracted into the Communication API. On top of the API is the XML Communications Language, whom elements, events, and actions represent features of the API. Compounding the language with other XML languages (e.g., XHTML), it is easy to create different communication applications (e.g., chat, file sharing etc.).

To use SIP via the communication API, one needs a external SIP client stack integrated to the API. In the current demo, the SIP client stack was implemented in the C programming language. Integrating it with X-Smiles requires Java Native Interface (JNI) since X-Smiles is implemented in Java.

The JNI interface is not needed, if a Java-based SIP client is used. NIST JAIN SIP is a Java-based SIP stack which could be a good option. In addition to basic SIP-messages it also supports SUBSCRIBE and NOTIFY-messages. There is also a Java-based SIP client implemented in the TML laboratory at TKK [25]. Drawback is that it has limited feature set; it supports INVITE,

ACK, OPTIONS, MESSAGE, and REGISTER methods. In addition, it can handle only one session at the time. That is, it may need further developing in order to be used in the project.

Third option is to stick with the current setting and continue to use the sockets to exchange messages between the SIP client and the browser. Then the programming language would not be issue but, on the other hand, the browser cannot control the SIP sessions at all.

## 8.4 Security

Future work for the security subproject includes IMS integration issues. Secure push can be implemented using standard 3GPP IP Multimedia Subsystem (IMS) technology and the Session Initiation Protocol (SIP), but this approach requires that the services are deployed within IMS Application Server, and that the user has an active IMS registration. The benefit of this approach is that the IMS billing system can be used, but one crucial limitation is that access is coupled to those 3G/WLAN base stations that are part of the ISPs access network or whose operator has a roaming agreement with the home operator.

The proposed architecture for secure federated push supports common-of-the-shelf (COTS) WLAN base stations that can be deployed in ad hoc manner. This motivates the integration of the lightweight Wesahmi system with IMS systems for improved service deployment and access flexibility. We are also planning to examine how different service domains can be federated together to form global service provision platforms.

## 9 Conclusions

This paper has described the current version of the system developed by project Wesahmi. At this stage it is a fairly loosely integrated collection of technologies and is mostly based on pre-existing open source components. The paper serves also as a documentation of the first reference implementation demonstrated to the steering group in June 2006.

The next phase of the project will be to design interfaces and components that enable the building of a seamless software platform based on the current implementation. The main target of the platform is to simplify the development of new applications with our system by providing applications with advanced services and transparent automation of commonly recurring tasks.

## References

- [1] Open Mobile Alliance. Oma network identity federation framework specification, 2006.
- [2] J. Arkko, V. Torvinen, G. Camarillo, A. Niemi, and T. Haukka. Security Mechanism Agreement for the Session Initiation Protocol (SIP). RFC 3329, IETF, Jan 2003.
- [3] Axis website. At <http://ws.apache.org/axis/>, February 2005.
- [4] Roach A. B. Session initiation protocol (sip)-specific event notification. Request for Comments (Standards Track) 3265, Internet Engineering Task Force, June 2002.
- [5] Richard Cardone, Danny Soroker, and Alpana Tiwari. Using XForms to simplify web programming. In *WWW '05: Proceedings of the 14th international conference on World Wide Web*, pages 215–224, New York, NY, USA, 2005. ACM Press.
- [6] Stuart Ceshire, Bernard Aboda, and Erik Guttman. Dynamic configuration of link-local ipv4 addresses. RFC 3927, Internet Engineering Task Force, March 2005.
- [7] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.1. RFC 4346, IETF, Apr 2006.
- [8] Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. XForms 1.0. W3C Recommendation, 2003.
- [9] Campbell B. ed., Mahy R. ed., and Jennings C. ed. The message session relay protocol (msrp). Internet draft (work in progress), Internet Engineering Task Force, December 2005.
- [10] J. Rosenberg et al. SIP: Session initiation protocol. RFC 3261 (Standards Track), IETF, June 2002.
- [11] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. Technical report, IETF, June 1999.
- [12] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, IETF, June 1999.
- [13] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). RFC 2409, IETF, Nov 1998.
- [14] IBM, BEA Systems, Microsoft, et al. Web Services Federation Language (WS-Federation), 2003.
- [15] IBM, BEA Systems, Microsoft, et al. Web Services Secure Conversation Language (WS-SecureConversation), 2005.
- [16] IBM, BEA Systems, Microsoft, et al. Web Services Trust Language (WS-Trust), 2005.

- [17] C. Jennings and R. Mahy. Managing client initiated connections in the session initiation protocol (SIP). Internet draft (work in progress), IETF, June 2006.
- [18] C. Jennings, J. Peterson, and J. Fisch. Certificate Management Service for The Session Initiation Protocol (SIP). Internet-Draft draft-ietf-sip-certs-01, IETF, Jun 2006.
- [19] S. Leggio, J. Manner, A. Hulkkonen, and K. Raatikainen. Session initiation protocol deployment in ad-hoc networks: a decentralized approach. In *2nd International Workshop on Wireless Ad-hoc Networks (IWVAN), London, May, 2005*.
- [20] OASIS. Web Services Security (WS-Security), 2004.
- [21] J. Peterson and C. Jennings. Enhancements for authenticated identity management in the session initiation protocol SIP. Internet draft (work in progress), Internet Engineering Task Force, October 2005.
- [22] J. Peterson, J. Polk, D. Sicker, and H. Tschofenig. Trait-Based Authorisation Requirements for the Session Initiation Protocol (SIP). Internet-Draft draft-ietf-sipping-authz-02, IETF SIPPING WG, Feb 2006.
- [23] David C. Plummer. An ethernet address resolution protocol. RFC 826, November 1982.
- [24] B. Ramsdell. Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification. RFC 3851, IETF, Jul 2004.
- [25] Jukka Rauhala. Universal sip client for consumer devices. Master's thesis, Helsinki University of Technology, Finland, April 2003.
- [26] J. Rosenberg. A presence event package for the session initiation protocol SIP. Request for Comments (Standards Track) 3856, Internet Engineering Task Force, August 2004.
- [27] J. Rosenberg and H. Schulzrinne. Session Initiation Protocol (SIP): Locating SIP Servers. RFC 3263, IETF, Jun 2002.
- [28] D. Schwartz, B. Sterman, E. Katz, and H. Tschofenig. SPAM for Internet Telephony (SPIT) Prevention using the Security Assertion Markup Language (SAML). Internet-Draft draft-schwartz-sipping-spit-saml-01, IETF SIPPING WG, Jun 2006.
- [29] H. Tschofenig, J. Hodges, J. Peterson, J. Polk, and D. Sicker. SIP SAML Profile and Binding. Internet-Draft draft-ietf-sip-saml-00, IETF SIP WG, Jun 2006.
- [30] C. Tschudin, P. Gunningberg, H. Lundgren, and E. Nordström. Lessons from experimental MANET research. *Elsevier Journal on Ad-Hoc Networks*, 3(3):221–233, March 2005.
- [31] D. Willis and B. Hoeneisen. Session Initiation Protocol (SIP) Extension Header Field for Registering Non-Adjacent Contacts. RFC 3327, IETF, Dec 2002.

## A Used open source software and applying licenses

Table 3: List of Open Source Software and their Licenses.

<b>Software</b>	<b>License</b>	<b>Usage</b>
<b>GNU oSIP Library</b>	LGPL	Low layer SIP-library
<b>eXosip - the eXtended osip Library</b>	GPL	Higher layer SIP-library built on top of oSIP
<b>OpenSLP</b>	BSD	Bootstrapping of environment
<b>X-Smiles browser</b>	The Telecommunications Software and Multimedia Laboratory, Helsinki University of Tehcnology Software License, Version 1.0 (based on the Apache Software License Version 1.1)	Web browser
<b>Apache Tomcat</b>	Apache License, Version 2.0	Web server