WeSAHMI System Specification

April 11, 2006

Abstract.

This is the technical specification of the infrastructure for wireless interactive applications to be defined and implemented in the WeSAHMI research project. The document includes descriptions of utilized technologies and software components to be adapted or implemented. Application-level functionality is illustrated through an example usage scenario.

Contents

1 Introduction **2**

2 Short description of application-level functionality **3**

3 Summary of technologies to be used **4**

3.1 Dynamic configuration of IPv4 addresses 4
3.1.1 Overview 4
3.1.2 Selecting an address 4
3.1.3 Conflict detection 4
3.2 Service Location Protocol 5
3.3 The Session Initiation Protocol Family 6
3.3.1 Instant Messaging and Presence 7
3.3.2 Distributed SIP 9
3.3.3 Security Support 10
3.4 SOAP 11
3.5 HTTP 11
3.6 Web Technologies 12
3.6.1 XHTML 12
3.6.2 XForms 12
3.6.3 SVG 12
3.6.4 Compound Document Formats 12
3.6.5 XBL 13
3.6.6 CSS 13

4 Overview of the infrastructure **13**

5 Infrastructure: Network **14**

6 Infrastructure: SLP **14**

7 Infrastructure: SIP **15**

8 Infrastructure: SOAP **18**

9 Infrastructure: client side **19**
9.1 X-Smiles XML browser 19
9.2 XForms with SOAP 20
9.3 X-Smiles in WeSAHMI 20

10 Infrastructure: server side **20**

11 Security Considerations **21**
11.1 Requirements 22
11.2 Building Blocks for Security 22
11.2.1 End-to-End Measures 22
11.2.2 SIP Security 23
11.2.3 Web Services Security 23
11.2.4 Identity Federation 24
11.3 Security Specification 24
11.3.1 Overview 24
11.3.2 Bootstrapping Trust 24
11.3.3 Client Security 25
11.3.4 Gateway Security 26
11.3.5 Secure Push 26
11.3.6 On Attacks 27
11.4 Implementation Plans 27

12 Detailed application-level description **28**

13 Implementation schedule **28**

Bibliography **31**

1 Introduction

The goal of the WeSAHMI project is to define and implement an experimental infrastructure for interactive wireless applications that can operate in an ad-hoc networking environment. In addition to the infrastructure, a demo application suite for an airport environment is to be implemented.

Some of the features of the infrastructure include identification of mobile users and tracking of their presence, delivery of content, notifications, and status updates to mobile users in a server-initiated fashion, and managing and updating the state of both clients and servers in real time.

State of the art web technologies such as AJAX support asynchronous, real-time updates of client state, but lack support for purely server-initiated interactions. For example, an airline might might want to provide its passengers the option to electronically check in using their wireless terminals, but not require that the users spontaneously navigate to the proper web address to do that. The airline would rather send a notification to the passenger once he arrives at the airport. In addition, the client would probably like to receive state updates

(such as real-time updates to the seating situation on a plane) without having to use polling.

The infrastructure to be developed extends the classical web architecture by catering for highly interactive applications, mobile clients, and targeted asynchronous information delivery, while retaining full server-side control over business logic.

This document serves as a system-level technical specification for the WeSAHMI infrastructure. The utilized technologies and how they are to be adopted in the project are described. System-level functionality is further explained through the use of an application-level usage scenario. This document is however not intended as an application-level functional specification, and the depicted scenario does not necessarily correspond directly to a final demo application.

2 Short description of application-level functionality

On application level the system provides mobile check-in service for passengers on the airport. It is an alternative for the traditional procedures performed either at the check-in desk or kiosk. The users of the system could then be able to perform all necessary functions with their own mobile devices without having to queue. These functions would then be:

1. request to check-in,
2. registration for a flight (with predefined seat),
3. guidance to baggage drop, and
4. guidance to security gate.

In addition to these the system keeps information of the passenger's flight(s) on a separate page. This information is first shown after the passenger has completed registration. It contains official time of departure, boarding time, seat number, and gate number and it can then be updated by the check-in service according to changes that are noticed in the Finnair's core system Amadeus. When a flight is delayed or the gate is changed the new information is automatically updated to the information page. Also when boarding starts a notification about it is displayed on the page.

Two-Dimensional Bar Code (2DBC) can be used to store passenger's name, flight number, airline code, date, and official time of departure. The code could then be presented on the screen of the mobile device for reading at the baggage drop and the security gate. Alternatively the mobile device could tell the same information directly to the terminal at these locations.

3 Summary of technologies to be used

3.1 Dynamic configuration of IPv4 addresses

3.1.1 Overview

The dynamic configuration of IPv4 link local addresses is specified by RFC 3927 [4]. 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.

3.1.2 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 [17], 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.

3.1.3 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.

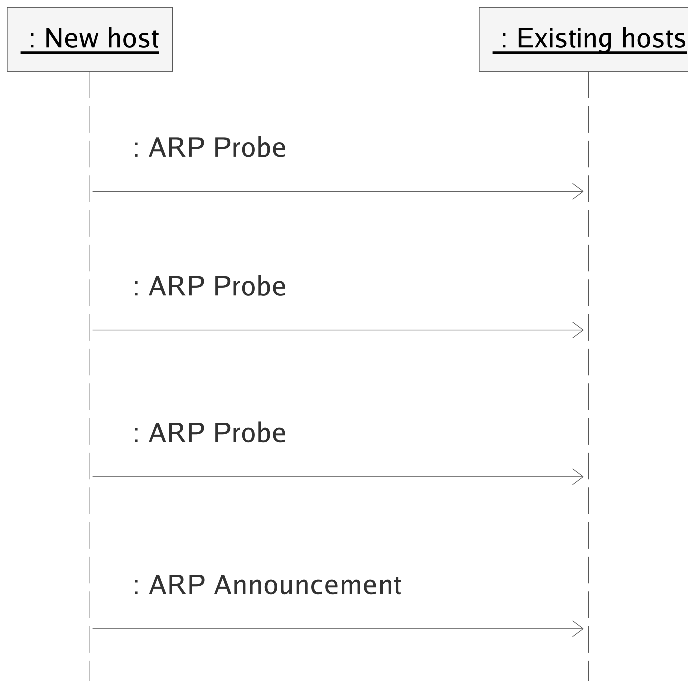

Figure 1: New host selects an address. All messages are broadcasted.

- 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.

3.2 Service Location Protocol

Service Location Protocol[9] 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 addreses. 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 on 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 existance when a UA or SA performs multi- or broadcast traffic.

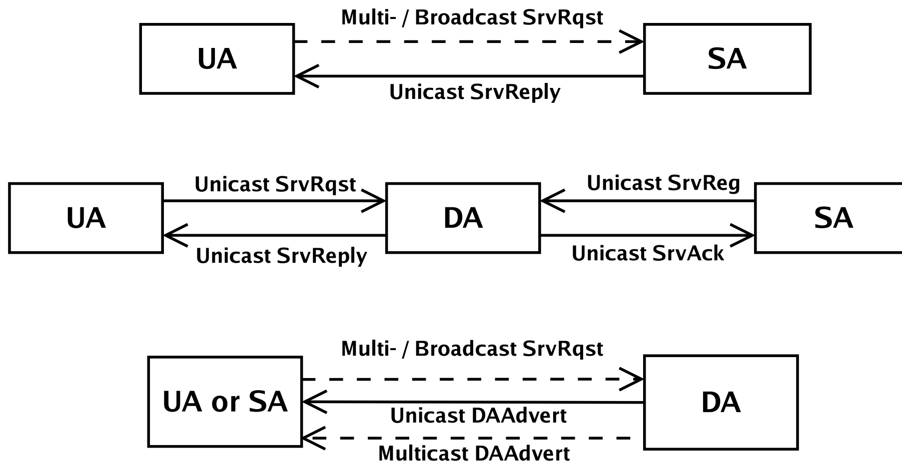

Figure 2: SLP agents and most common protocol messages.

3.3 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) [7] 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. 3). 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 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).

3.3.1 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.

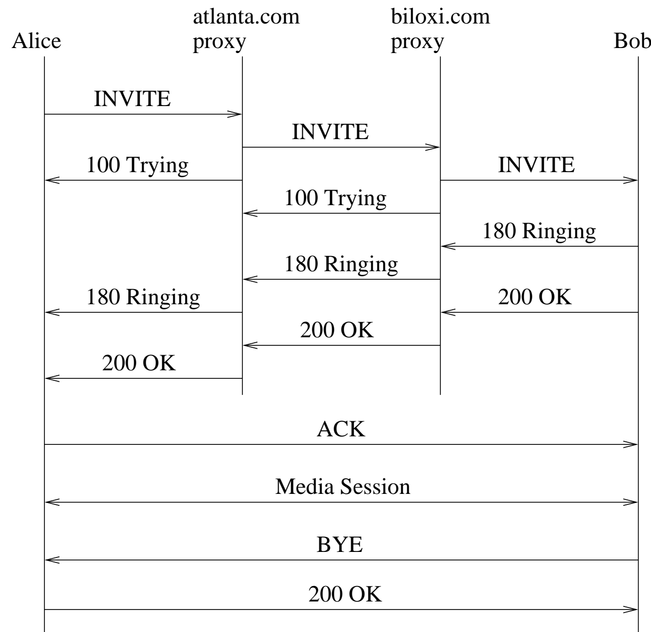

Figure 3: Example SIP session setup

The IMP architecture proposed by the SIMPLE working group builds on top of the SIP Event Notification Framework [2] and realizes a specific event instantation called *presence* [18]. 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 [2]. 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) [6], 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.

3.3.2 Distributed SIP

Decentralized SIP (dSIP) [13] 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 [19]. 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. 4: 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 [13].

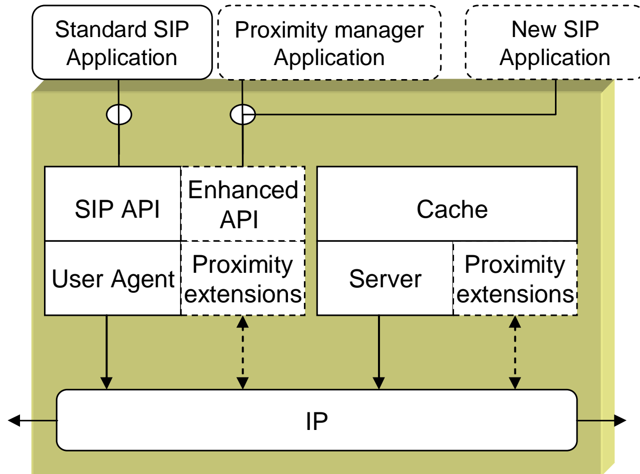

Figure 4: Software Architecture for decentralized SIP

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.

3.3.3 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. Veryfing SIP users' identity can be handled by the SIP authenticated identity [16] 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.

3.4 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.

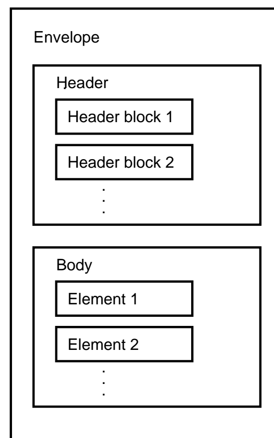

Figure 5: Structure of the SOAP-message

3.5 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. [8]

3.6 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.

3.6.1 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.

3.6.2 XForms

XForms 1.0 Recommendation [5] 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 [3]. 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.

3.6.3 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.

3.6.4 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.

3.6.5 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 capale 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.

3.6.6 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.

4 Overview of the infrastructure

An overview of high-level system components can be seen in Figure 6.

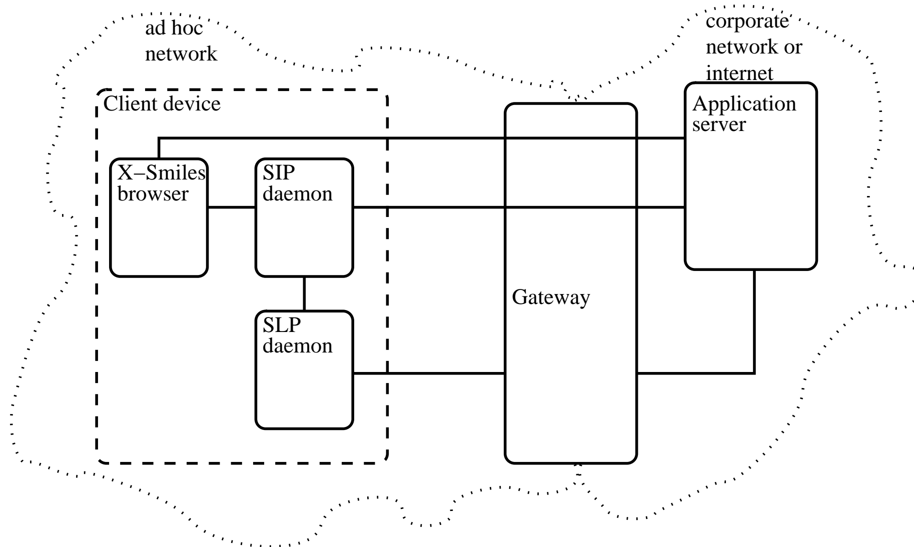

Figure 6: System components.

Passengers' client devices are enabled to find the check-in service through SLP passive service discovery. This functionality is implemented in the SLP daemon process. The SLP daemon passes information of discovered services to the SIP daemon, which subscribes to SIP event notifications from the service. These notifications can be used for pushing interaction initiations (i.e. URLs to be loaded) or relaying real-time update notifications from the service to the

client. The dynamic user interface is implemented in the X-Smiles browser that is capable of handling SOAP messages passed to it by the SIP daemon. SVG is used to provide more specialized controls for the forms that are bound to them by XBL.

5 Infrastructure: 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 3.1).

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 trough 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.

6 Infrastructure: SLP

SLPv2 open source C implementation OpenSLP version 1.0.11 [15] was used in the service discovery scheme of SESSI project, the predecessor of WeSAHMI. It was then deemed to be suitable for dynamic ad-hoc networks and is light enough to run in mobile devices with limited resources. It also provides a non-blocking service discovery implementation for the clients in the sense that the applications are able to function normally while they receive updated service information. Within SESSI project a Passive Discovery (PD) functionality for SLP was implemented. It provides a method for bootstrapping the SIP service in the infrastructure while the address of the client is unknown.

The bootstrapping of the SIP service is implemented as SA initiated service discovery. The SIP service is registered by server side SIP implementation by calling Service Discovery Module (SD Module) function `SD_enablePassiveDiscovery` which in turn calls LibSLP function `SESSI_SLPPassiveReg`. It constructs and sends a control message `PDSrvReg` to SLPD. SLPD will then broadcast `SrvAdvertisement` messages containing the SIP-address of the registered service. The SIP service can be unregistered by calling SD API function `SD_disablePassiveDiscovery`. It calls LibSLP function `SESSI_SLPPassiveDeReg` that constructs and sends a control message `PDSrvDeReg` to SLPD. SLPD responds with `GeneralReplyMsg` and both functions `SESSI_SLPPassiveDeReg` and `SD_disablePassiveDiscovery` return suc-

cessfully. The advertisement messages can be signed by the server side SIP if the advertised services need to be authenticated on SD level. The clients are thus able to verify them. The registration and deregistration processes are illustrated in Figure 7.

The SIP daemon running in the client initiates passive service discovery by calling SD Module function `SD_registerSrvAdvListener` which in turn calls LibSLP function `SESSI_SLPRegFilter` that constructs and sends a control message `PDSrvRqst` to SLPD. It registers a listener that then sends back a service address to LibSLP in another control message called `PDSrvRply` whenever it receives a `SrvAdvertisement` message that contains a service of the observed service type. LibSLP proceeds then to call a callback function registered by SD Module when it receives such an control message. The SD Module in turn then calls a callback registered by the SIP daemon to return the service found new service information. The SIP daemon stops passive discovery by calling SD Module function `SD_unregisterSrvAdvListener` which calls LibSLP function `SESSI_SLPDeregFilter` that closes the control message socket. When SLPD notices that the socket is closed it removes the SrvAdvertisement listener. The listener registration and deregistration are illustrated in Figure 8.

The SIP service used in Wesahmi is registered with ID `finnair` and attribute `event` that contains the name of the event to which it can subscribe to. The SIP daemon may use SD Module function `SD_getAttribute` to obtain it separately.

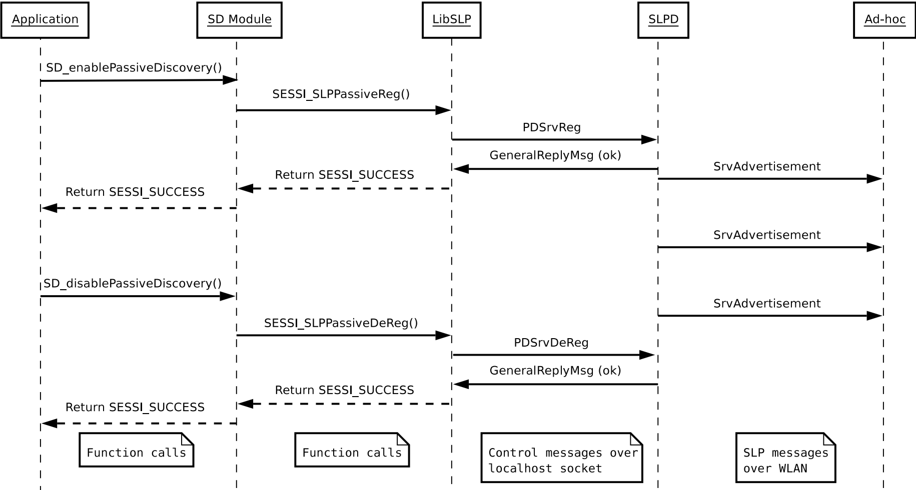

Figure 7: Registering and unregistering services for PD

7 Infrastructure: SIP

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. In the first phase these notification carry information that some information has changed and browser should refresh the viewed page.

The server side must be able to process SUBSCRIBE-messages for event "resource-update"and send NOTIFY-messages. NOTIFY-messages will carry SOAP-messages in the body part of the message and the SOAP-message will have more detailed descriptions of the event. Here is an example of SUBSCRIBE message.

```
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
Content-Length: 0
```

At the client side there is SIP-daemon process which is responsible for handling the SUBSCRIBE/NOTIFY-messages. The daemon process will receive an SLP-advertisement to inform that there is a notification service available. The advertisement has the SIP-address of service and the daemon process may then do a query for the attributes of this service. The notification service will have an attribute that tells the notification event type. When the daemon process has received the advertisment and made a query for the attributes it sends a SUBSCRIBE-message to the address that was in the advertisement with Event-field set to value "resource-update".

The client side has actually two processes for handling SIP-messages. SIP-proxy and the SIP-daemon itself. This is because we are using the distributed

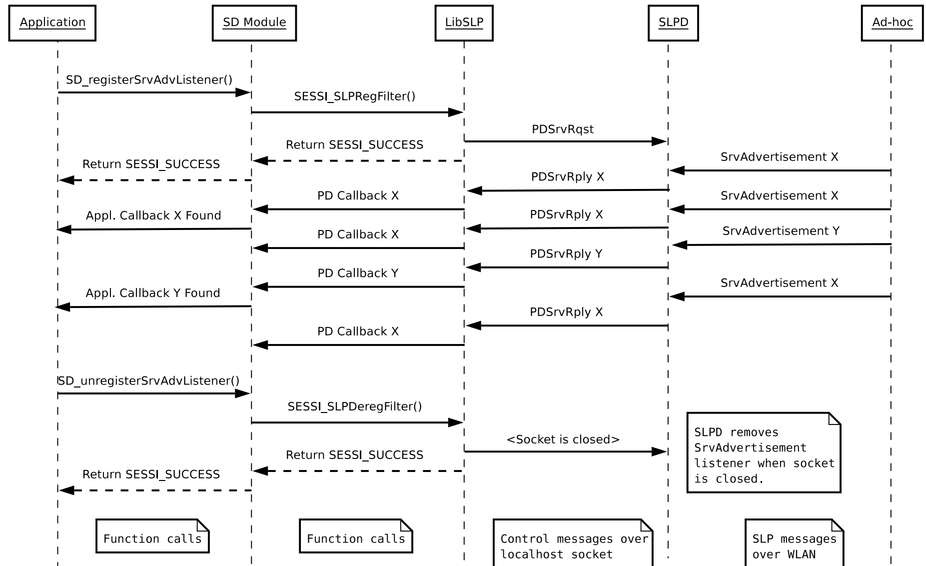

Figure 8: Discovering services with PD

version of sip (dSIP) implemented in SESSI-project and it has local SIP-proxys on client machines. All the SIP-messages are sent and received through the local SIP-proxy. If we wouldn't want to use dSIP the SIP-daemon and local SIP-proxy could be replaced by single SIP-daemon which would receive the NOTIFY-messages and send the SOAP-part of the message to the browser.

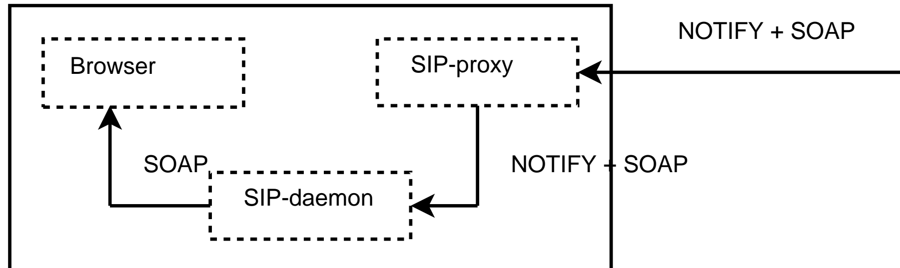

Figure 9: Processes responsible of handling SIP and SOAP messages in the client side.

Notification server receives the SUBSCRIBE-message and after that it 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 also a SOAP-body which has a more detailed description of event. 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
Content-Type: application/soap+xml
Content-Length: 242

<SOAP-body>
```

The SIP-daemon receives the NOTIFY-message (which is received through local SIP-proxy) and by looking the value of the Event-field it knows how to handle the message. SIP-daemon forwards only the SOAP-body of the message to the browser application which should be able to process it. The SOAP-message is sent via a local socket to browser. If the browser is not running the daemon will create a new instance. In the case the browser is running, the daemon opens a new tab to the browser window. This could be done, e.g., by giving a command line parameters to the browser.

The new components here are the daemon process and server side application which is able send SUBSCRIBE-messages and handle incoming NOTIFY-messages. In the first phase the server side can be a simple test application

which receives SUBSCRIBE-messages and can send NOTIFY-messages by users command.

The daemon process will use a modified version of the eXosip library for handling SIP-messages and Service Discovery module for listening service advertisements. eXosip is modified to support dSIP.

8 Infrastructure: SOAP

SOAP-messages are carried in the body part of the NOTIFY-messages. SOAP-message itself contains the detailed information aboud the event. Here's example of the 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
Content-Type: application/soap+xml
Content-Length: 242

<SOAP-body>
```

In NOTIFY-message there is Event-header field which value is set to "resource-update". This header field indicates the type of the event which receiver needs to know to process it correctly. Content-Type header is set to value "application/soap+xml" to tell that the payload is SOAP-message.

When the SIP-daemon receives the message it checks the Event-header field. If it matches to "resource-update" it launches the browser (if it is not already running) and sends the SOAP-message to the browser via socket.

In the first phase the SOAP-message contained in the body has only information that something has changed and the browser should refresh the user interface by retrieving the viewed page from the server. Later the SOAP-messages can contain more detailed information about what has changed so that browser can update the user interface only by using the information in the SOAP-message. The SOAP-message can now be something simple like

```
<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">
         <url>http://www.site.com/page.html</url>
      </m:ContentChanged>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

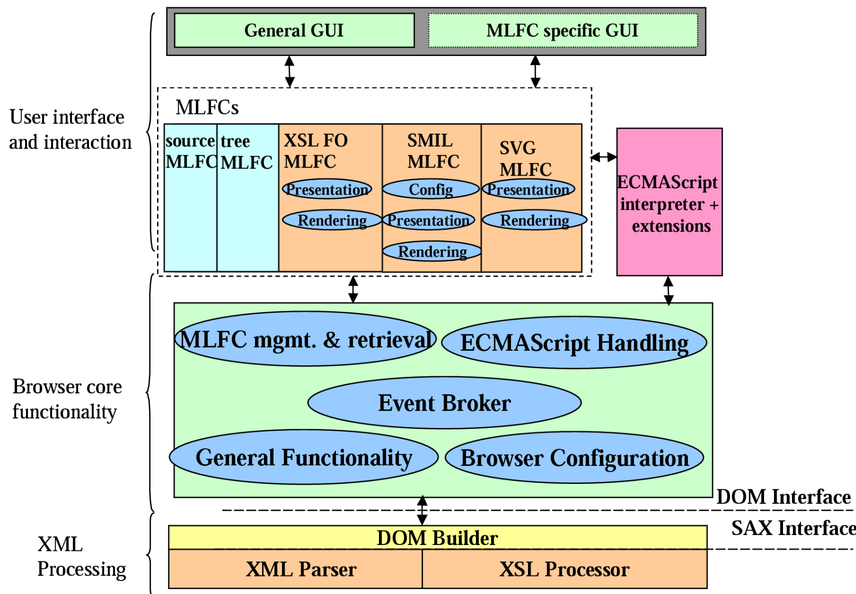

Figure 10: X-Smiles architecture

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 url-element inside the ContentChanged-element tells the url-address where to retrieve the page.

9 Infrastructure: client side

9.1 X-Smiles XML browser

X-Smiles[1] is an open source XML browser developed at the Helsinki University of Technology. The main components of the X-Smiles browser can be divided into four groups: XML processing, Browser Core Functionality, Markup Language Functional Components (MLFCs) and ECMAScript Interpreter, and Graphical User Interfaces (GUIs). They are depicted in Figure 10.

The core of the browser controls the overall operation of the browser. It includes browser configuration, event handling, XML Broker, etc. MLFCs handle different XML languages and render the documents. There are several GUIs in the X-Smiles distribution. They are used to adapt the browser to various devices or as virtual prototypes, when prototyping content targeted to diverse range of devices. [20]

In WeSAHMI project, we are using X-Smiles to render the UIs. We will use compound documents, which consist of XHTML, XForms, and SVG. XBL is used to tie custom controls to the documents. X-Smiles is extended as a part of the project to support required UIs. The extensions are discussed below.

[1] X-Smiles, available online http://www.x-smiles.org

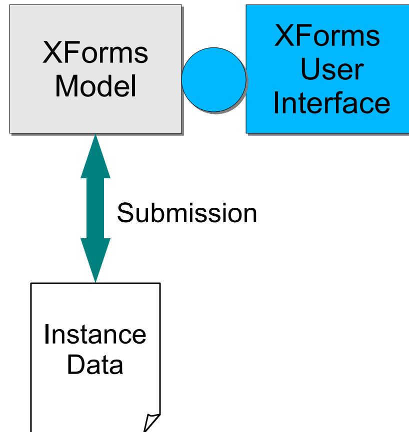

Figure 11: XForms main aspects.

9.2 XForms with SOAP

XForms consists of data instance, XForms model, and UI (cf. Figure 11). The model modifies the XML-based data instance. There can be more than one data instance for a form. It is possible to sent data instances and parts of them asynchronously all the time. Inputs and outputs of the UI are bound to elements of the data instances. Thus, the UI can be kept updated all the time.

Since SOAP messages are also XML-based, they can be used directly as a XForms data-instance. In the form, it is possible automatically to add SOAP header to the instance to be sent. An addition, a subtree of an instance can be moved to another instance. Thus, a set of SOAP messages can be collected to an instance and, on the other hand, a part of the instance can be extracted to a SOAP message and submitted.

9.3 X-Smiles in WeSAHMI

By the summer, we will implement the interaction with SIP daemon and X-Smiles browser. The interaction is realized through the socket. At this phase, the daemon informs the browser that it should refresh the UI. For the user interface, we will use compound documents by inclusion. That is, SVG elements included directly to an XHTML document.

Later, the SIP daemon will send SOAP messages through the socket and the messages are used as a part of XForms data instances as explained above. Thus, the whole interface has not to be updated every something changes. Some of the XForms controls will be replaced by custom controls through XBL. The controls will be realized with SVG.

10 Infrastructure: server side

The server side infrastructure is depicted in Figure 12. The server side contains primarily the Web server. In addition, there is a need for Web Application server, and later perhaps also a Workflow server. The Web server is a standard

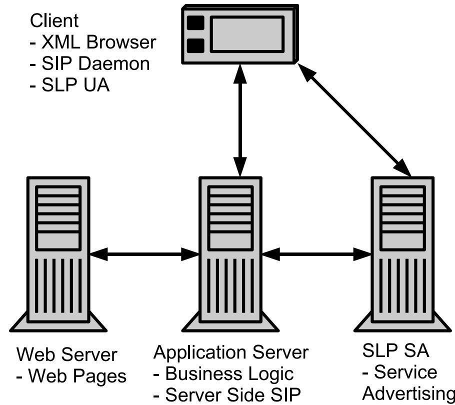

Figure 12: Server side infrastructure.

web server, which stores the web pages of the use case. The most common web server is Apache HTTP server, which also an open source server. Thus, it can be used also in WeSAHMI project.

The Web Application server runs the actual business logic. The Web Application server provides the service advertisement information to the SLP SA. In addition, it receives the service registration information from the SLP SA. The web application also processes the SIP SUBSCRIBE and NOTIFY messages. After the SIP session has been established, the Web Application server can exchange SOAP messages with the client. The Web Application server processes also the HTTP request, while the actual content comes from the Web server.

For example, Apache Tomcat can be used as the Web Application server. In that case, either Java Servlets or JavaServer Page can be used for coding the actual web application logic. In addition, interfaces to the SLP SA and server side SIP user agent have to be developed (???).

11 Security Considerations

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 described in this document 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.

11.1 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.

11.2 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.

11.2.1 End-to-End Measures

Transport Layer Security (TLS) provides session-layer security with mutual certificate-based authentication. IP Security (IPsec) and Internet Key Exchange (IKE) (RFC 2409) 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.

11.2.2 SIP Security

SIP security solutions leverage S/MIME, 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.

11.2.3 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 [14]. 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 [12]. 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 [11] 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 [12]. For the hybrid network environment, each peer must implement a trust engine and be able to process security tokens.

11.2.4 Identity Federation

WS-Federation defines a federated identity and mechanisms to broker and federate identity, trust, and claims about them [10]. 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].

11.3 Security Specification

11.3.1 Overview

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 is used for end-to-end 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 perfomed through challenge/response, client signature, or a security token. The service access results in the desired content or an authentication failure (5).

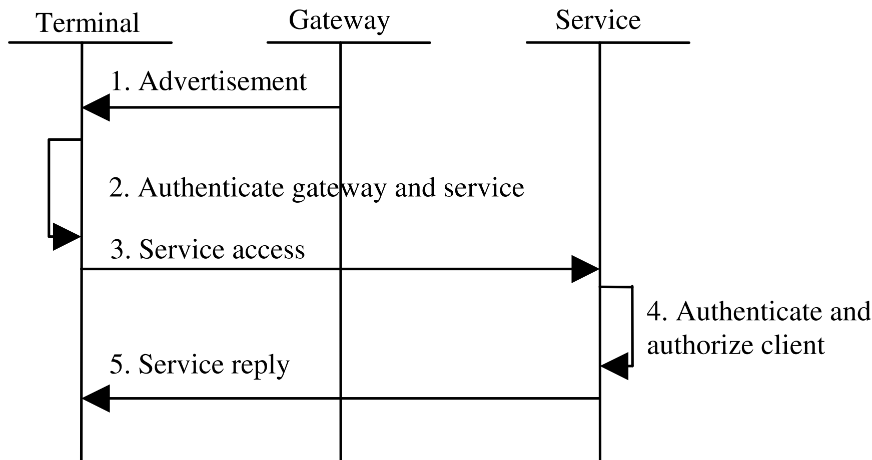

Figure 13: Overview of interactions.

11.3.2 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 following three trust obvervations 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.

11.3.3 Client Security

Figure 14 illustrates how a client contacts a server and the server verifies the identity of the client. The interaction proceeds as follows:

- Client contacts server using a secure connection (TLS) (1-2).
- Server certificate is verified. The client may also issue a self-signed certificate that asserts its SIP identity (3).
- Client account is verified. The server needs to know the client SIP URI for push functionality. Alternatively a phone number for SMS / email address is needed. The SIP URI will then be the identifier of the user.
- The server issues a reverse-routability test message (4). The client must respond succesfully to this message in order for the accound to be activated. This test ensures that the client is reachable through the SIP URI it has provided (5).

Security techniques used for subsequent communications depend on the employed protocol. The transport connectivity is protected using TLS and this supports secure web browsing. Any SIP messages between the client and the server are signed and encrypted.

For subsequent messaging, each message needs to be transmitted over SSL or encrypted and signed separately. This may be optimized by generating a temporal session key as follows: The server sends a signed message to the client

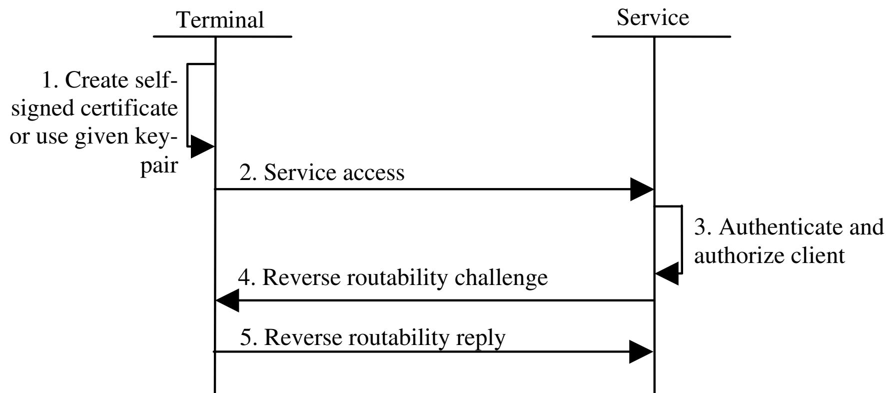

Figure 14: Bootstrapping client identity

that contains a temporal symmetric key. The services authenticate the client using this temporal key. The key has a life-time after which the client must retrieve a new key. The key generation mechanism should guarantee perfect forward secrecy. After the temporal symmetric key is established between the client and the server, public key crypto is not needed for messages between the systems, but rather each end-to-end communication is encrypted using the session key. If the session is key is expired, the server may push a new session key to the client.

11.3.4 Gateway Security

The gateway may or may not be a trusted entity. The SLP advertisement messages broadcasted by the gateway may be secured using S/MIME or using signatures. This requires that the public key of the gateway and the server providing the advertised service are known. If the gateway simply advertises services, it is enough to verify that the advertised message is not bogus.

11.3.5 Secure Push

Figure 15 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 the 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.

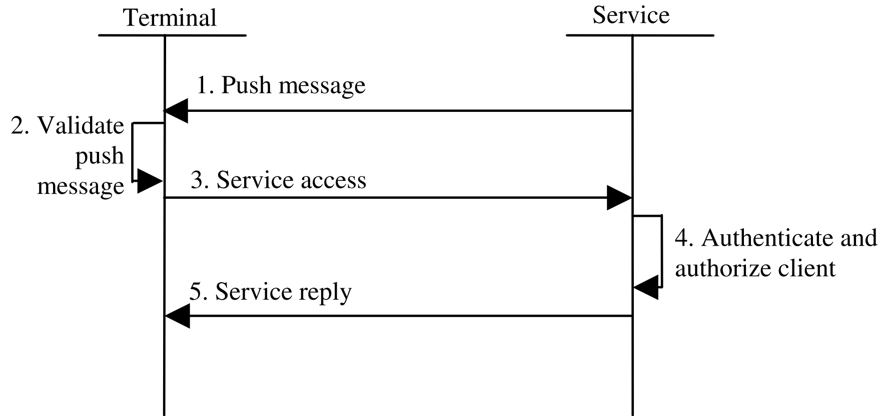

Figure 15: Secure push

11.3.6 On Attacks

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.

11.4 Implementation Plans

The first prototype is implemented without security features. The prototype is then used to study the environment and the airline scenario. Subsequent implementation work 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. Any client authentication solution needs to be compatible with the X-Smiles browser.

12 Detailed application-level description

Figure 16 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.

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.

When the user selects the positive option, the actual check-in page is loaded (3). 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[2], boarding instructions are pushed to the passenger (4). This includes information on how to find the security check point and the gate.

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

The interaction depicted in Figure 16 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.

13 Implementation schedule

Currently the implementation schedule is divided into two phases:

1. Initial phase: functionality to be completed by the summer of 2006.
2. Final phase: functionality to be completed during the rest of the project.

Initial phase

The functionality to be completed in the initial phase corresponds to that described in Figure 16 in the previous section.

[2]The availability of such information is not known. This notification is not essential, but makes the interaction smoother.

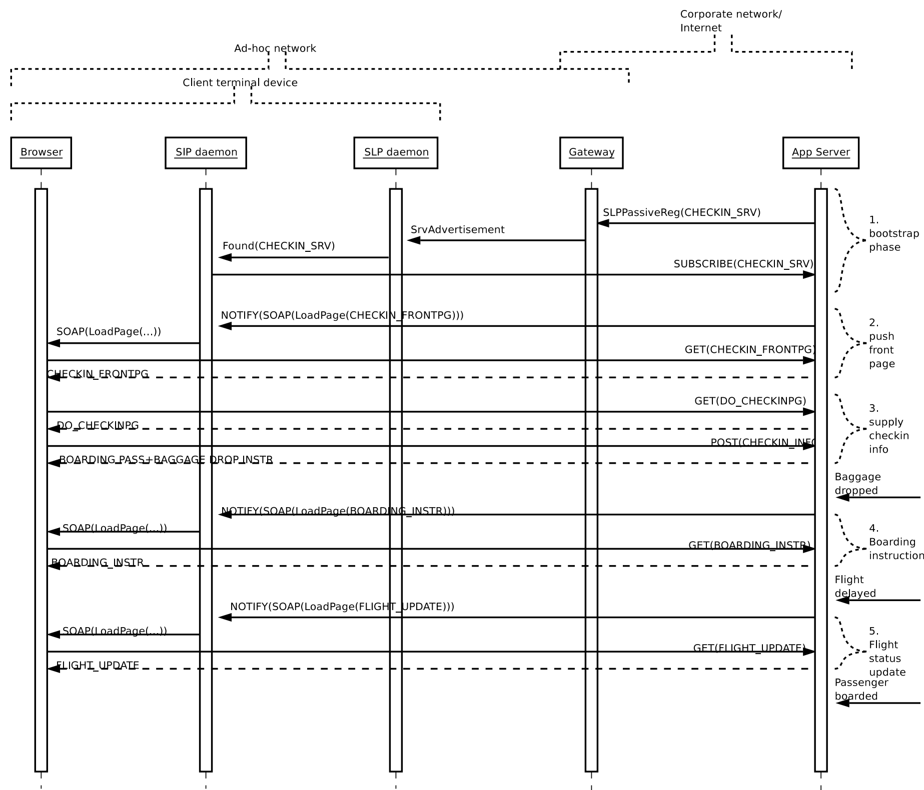

Figure 16: Sequence diagram of application behavior.

By the summer we will have ready an implementation that is able to bootstrap the SIP service for an arbitrary mobile node.

Final phase

The final phase will be divided into subphases after the completion of the initial phase. Some of the planned improvements are described in the following. This is, however, not a complete treatise of the overall technical goals of the project.

The Web Application server can be partly replaced by a Workflow engine. The idea is that the business logic is expressed using some workflow modeling language, which is then executed in the Workflow engine. The Workflow engine receives SOAP/SIP messages, processes them according the modeling language and sends further messages. It also keeps track of the different activities, active processes and their stage. This simplifies the development and maintenance of complex business processes.

References

[1] Open Mobile Alliance. Oma network identity federation framework specification, 2006.

[2] Roach A. B. Session initiation protocol (sip)-specific event notification. Request for Comments (Standards Track) 3265, Internet Engineering Task Force, June 2002.

[3] 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.

[4] Stuart Ceshire, Bernard Aboda, and Erik Guttman. Dynamic configuration of link-local ipv4 addresses. RFC 3927, Internet Engineering Task Force, March 2005.

[5] Micah Dubinko, Leigh L. Klotz, Roland Merrick, and T. V. Raman. XForms 1.0. W3C Recommendation, 2003.

[6] Campbell B. ed., Mahy R. ed., and Jenning C. ed. The message session relay protocol (msrp). Internet draft (work in progress), Internet Engineering Task Force, December 2005.

[7] J. Rosenberg et al. SIP: Session initiation protocol. RFC 3261 (Standards Track), IETF, June 2002.

[8] R. Fielding et al. Hypertext Transfer Protocol – HTTP/1.1. Technical report, IETF, June 1999.

[9] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. RFC 2608, IETF, June 1999.

[10] IBM, BEA Systems, Microsoft, et al. Web Services Federation Language (WS-Federation), 2003.

[11] IBM, BEA Systems, Microsoft, et al. Web Services Secure Conversation Language (WS-SecureConversation), 2005.

[12] IBM, BEA Systems, Microsoft, et al. Web Services Trust Language (WS-Trust), 2005.

[13] 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 (IWWAN), London, May*, 2005.

[14] OASIS. Web Services Security (WS-Security), 2004.

[15] OpenSLP Project Group website. At `http://www.openslp.org`, April 2004.

[16] 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.

[17] David C. Plummer. An ethernet address resolution protocol. RFC 826, November 1982.

[18] J. Rosenberg. A presence event package for the session initiation protocol SIP. Request for Comments (Standards Track) 3856, Internet Engineering Task Force, August 2004.

[19] 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.

[20] Juha Vierinen, Kari Pihkala, and Petri Vuorimaa. XML based prototypes for future mobile services. In *Proc. 6th World Multiconf. Systemics, Cybernetics and Informatics, SCI 2002*, pages 135–140, 2002.