TECHNISCHE UNIVERSITÄT WIEN

VIENNA UNIVERSITY OF TECHNOLOGY

M A S T E R A R B E I T

A Resource List Server for the IP Multimedia Subsystem

Ausgeführt am

Institut für Breitbandkommunikation
der Technischen Universität Wien

unter der Anleitung von
O. Univ. Prof. Dr. techn. Harmen R. Van As
Dipl.-Ing. Mag. Joachim Fabini

durch

Reinhold Buchinger
Josef Munggenaststraße 27
3591 Altenburg
Austria

Wien, 01. Februar 2008

Zusammenfassung

Ein Presence Dienst stellt zahlreiche Informationen über Benutzer, Dienste und Endgeräte (wie z.B. momentane Aktivitäten, Aufenthaltsort oder verfügbare Kommunikationskanäle) anderen Benutzern und Diensten zur Verfügung. Der Presence Dienst schafft Voraussetzungen für einige andere Dienste, wie z.B. Instant Messaging, und wird voraussichtlich in Zukunft allgegenwärtig sein.
In einem Mobilfunknetz ist aber die Kanalkapazität zwischen einem Endgerät und dem Kernnetz, sowie die Rechnerleistung der Endgeräte, begrenzt. Hinzu kommt, dass in vielen Anwendungsfällen ein Benutzer an der Presence Information mehrerer anderer Benutzer zugleich interessiert ist. Falls ein Endgerät die Presence Information jedes Benutzers einzeln abfragt, müsste es eine unnötig hohe Zahl an Nachrichten senden und verarbeiten.
Ein Resource List Server (RLS) adressiert diese Problemstellung. Er ruft die Presence Informationen aller gewünschter Benutzer ab und sendet diese gesammelt in einer einzigen Nachricht an das Endgerät.
Wir stellen in dieser Arbeit ein generisches Design für solch einen RLS vor. Weiters präsentieren wir die Implementierung eines Prototyps, welcher als Modul für den OpenSER SIP Server realisiert wurde.
Als Erweiterung zum standardisierten Presence Dienst betrachten wir die Aggregation von Presence Information. Ein Benutzer ist dabei an der Presence Information einer Gruppe selbst und nicht an der Presence Information der einzelnen Mitglieder dieser Gruppe interessiert.
Wir schlagen verschiedene Architekturen zur Presence Aggregation vor und diskutieren, wie die Presence Information einer Gruppe von Benutzern aggregiert werden kann. Ein erweiterter Presence Server oder RLS übernimmt dabei die Aufgabe der Presence Aggregation.

Abstract

A presence service provides extensive customized information of end-users and services (like current activities, location or reachability for communication) to other end-users and services. The presence service is likely to become omnipresent in the future and serves as an enabler for several other services (e.g., instant messaging).
In a mobile network, the channel capacity between a terminal and the core network, as well as the processing power of terminals, is limited. Additionally, in many cases a user is interested in the presence information of several resources at the same time. If a terminal retrieves the presence information of each resource separately, it must send and process an unnecessarily high number of messages.
A Resource List Server (RLS) addresses this problem. It fetches the desired presence information of several resources and distributes them to a terminal in a single message.
In this thesis, we propose a generic design for such a RLS and present a prototype, which we have implemented as module for the OpenSER SIP Server.
As extension to the standardized presence service, we discuss the aggregation of presence information. In this scenario, a user is interested in the presence information of a group itself, and not in the presence information of the individual members of the group.
We propose several architectures for presence aggregation and discuss how presence information of a group of users/identities can be aggregated. An enhanced Presence Server or RLS can fulfill the task of presence aggregation.

Contents

1 Introduction

Since the introduction of the IP Multimedia Subsystem (IMS) in 3GPP Release 5, the number of network operators and next-generation solution providers which announce some level of support for the IMS is constantly growing.

The 3rd Generation Partnership Project (3GPP) has standardized the IMS as a cornerstone of the evolution of current networks to a single all IP-based network. The standardized framework enables the delivery of a broad range of services (telephony, messaging, etc.) and media (voice, video, text, etc.) to fixed, mobile and cable customers. Different services can be combined and integrated into a single user experience, independent from the currently used device[1] or access technology.

Camarillo and Garcia-Martin state that presence is one of the basic services that in the future is likely to become omnipresent ([22], p. 295). The presence service can provide an extensive customized amount of information to end-users or other services. The type of presence information ranges from information about different communication means and their status, over location information, to the mood of a person and much more. An industry survey from 2005 among 120 vendors and providers emphasizes the importance of presence. The survey considers presence to be the most important service for a successful IMS implementation [17].

Existing presence architectures for the IMS include a so-called Resource List Server (RLS). A RLS enables a user to subscribe to a list of resources instead of subscribing to each resource individually. The user gets notified about the presence information of all resources in the resource list by receiving a single message. In this way, a RLS saves network traffic and relieves terminals of processing work. We propose a generic design for a RLS and discuss a prototype implementation.

[1]In this thesis we use the terms *device*, *terminal* and *User Equipment (UE)* synonymously.

Apart from the basic functionality a presence service offers, many advanced applications are conceivable. One possibility is the aggregation of presence information. It addresses the question how presence information of users/identities who belong to the same group can be combined to a common presence information of this group. Assuming a call center, a caller only wants to know if at least one agent is available. The aggregate presence status of all call-center agents (i.e., the call center is available if at least one agent is available) is more useful than the presence status of a single agent. We present several architectures for presence aggregation and compare their pros and cons.

1.1 Related Work

A huge number of publications consider distinct aspects of presence (e.g., [76], [36], [32]). Several papers present ways of optimizing message traffic in presence systems. Wegscheider suggests in [83] different ways of minimizing the traffic on the air interface between a user terminal and a RLS while maintaining full responsiveness for the end user. Zhao et al. introduce in [84] the concept of an enhanced RLS, which allows subscriptions to the information of part resources of one or multiple resource list(s). Beltran et al. propose in [19] a Presence Personal Proxy for every user, which, among other improvements, reduces the presence signaling traffic.

1.2 Structure of this Thesis

The remainder of this thesis is structured as follows:
Subsequent to this introductory section, Section 2 builds the theoretical foundation for the Resource List Server by presenting and analyzing existing presence-related standards and concepts, both from an architectural and from a data format point of view. Following the theory part, Section 3 presents the generic design which we propose for the Resource List Server. Section

4 illustrates the RLS prototype, which has been implemented as modular extension to an existing open-source Session Initiation Protocol (SIP) server. Subsequently, Section 5 discusses the aggregation of presence information and Section 6 summarizes the contents of this thesis.

2 Presence Service in the IMS

This section discusses presence service and related terms. Section 2.1 gives an introduction to the IMS before we discuss the concepts and architecture of the OMA Presence SIMPLE enabler in Section 2.2. Section 2.3 presents the OMA XML Document Management (XDM) enabler, which strongly interacts with the OMA Presence SIMPLE enabler, whereas Section 2.4 focuses on presence information modeling. We discuss several presence information data formats.

2.1 Introduction to the IMS

The IP Multimedia Subsystem (IMS) is an architectural framework that aims to combine the quality and interoperability of telecom with the quick and innovative development of the Internet. It intends to provide a single integrated network for all access types to enable common services, no matter a customer uses a mobile device or a PC client. However, the IMS is still aware of the used access network type to adapt a service to the characteristics of the access network. The IMS standard supports multiple access types, including 2G and 3G cellular networks as well as Wireless LAN (WLAN), Worldwide Interoperability for Microwave Access (WiMAX), Digital Subscriber Line (DSL), cable or fiber-to-the-home.

Three strong arguments for the IMS are: Quality of Service (QoS), charging, and integration of different services ([22], p. 7). The IMS enables operators to control the QoS a user is allocated based on the user's subscription and the current state of the network. Regarding charging, the IMS allows very flexible business models. Instead of charging per byte, an operator can charge certain services or actions (e.g., a single instant message). This is feasible because the IMS framework covers both the signaling and the media component, which enables correlation for charging or QoS purposes.

A key aspect of the IMS is a fast and efficient service creation and delivery.

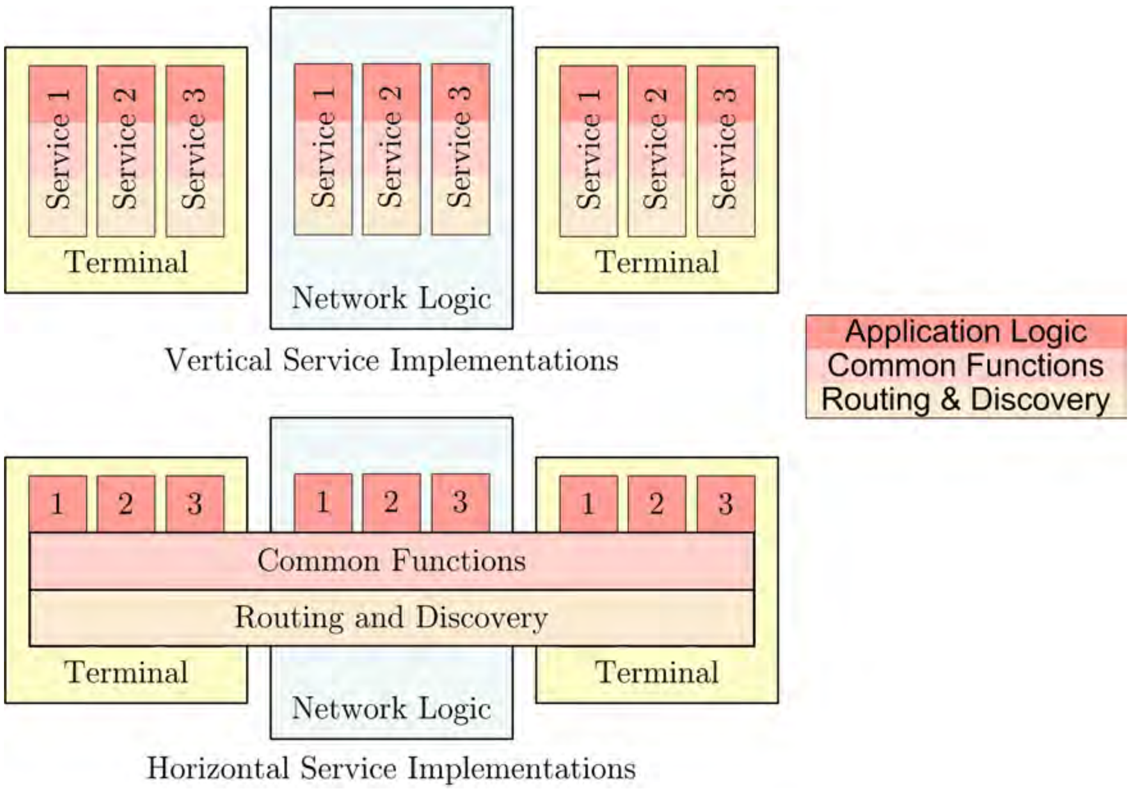


Figure 1: Vertical service versus horizontal service implementations (source: [27], p. 5)

The IMS provides a number of common functions which can be reused by virtually all services in the network. For example, the IMS defines how service requests are routed, how charging is performed, and how a subscriber is authenticated. In a non-IMS network each service is typically designed, implemented and tested from the scratch, and must be separately maintained. Fig. 1 depicts the difference between vertical service implementations in non-IMS networks and horizontal service implementation in IMS networks. A horizontal service implementation provides less parallel development, more reliable systems and higher-level abstractions for developers, who can focus on the actual application. A common service framework eases as well the combination of services. E.g., a service can insert an announcement in an

ongoing video conference every time a colleague goes online.
Aside from service-oriented aspects, a strong argument is cost savings. The IMS allows separating the costly access part of a network from the part of the network responsible for service creation and management. According to a partner with IBM, "[...] an opex reduction in excess of 20 per cent is achievable with IMS compared with running legacy networks." [37]

2.1.1 IMS Standardization

The IMS is an integral part of the 3rd Generation Partnership Project (3GPP) standardization work for 3G mobile phone systems in Universal Mobile Telecommunications System (UMTS) networks. It first appeared in 3GPP Release 5 in 2002. 3GPP Release 6, which followed at the end of 2004, includes the requirement of access independence and adds support for WLAN. The European Telecommunications Standards Institute (ETSI) Telecoms and Internet converged Services and Protocols for Advanced Networks (TISPAN) Working Group adopted the IMS architecture and added support for fixed networks. 3GPP, in turn, integrated TISPAN-IMS Release 1 in its 3GPP Release 7. The standardization body for cellular networks in North America and Asia, 3rd Generation Partnership Project 2 (3GPP2), defined its own version of IMS. Both IMS are fairly similar but still differ in a few aspects.
One outstanding characteristic of the IMS, when compared to previous 3GPP standards, is that both signaling and media specifications rely on Internet Engineering Task Force (IETF) based protocols. Most notable is the Session Initiation Protocol (SIP) [73], which 3GPP has selected as the main signaling protocol in IMS networks.

2.1.2 IMS Architecture

In this section we give a short and compact introduction to the IMS architecture. For a deeper insight, the reader can refer to 3GPP TS 23.002 [12].

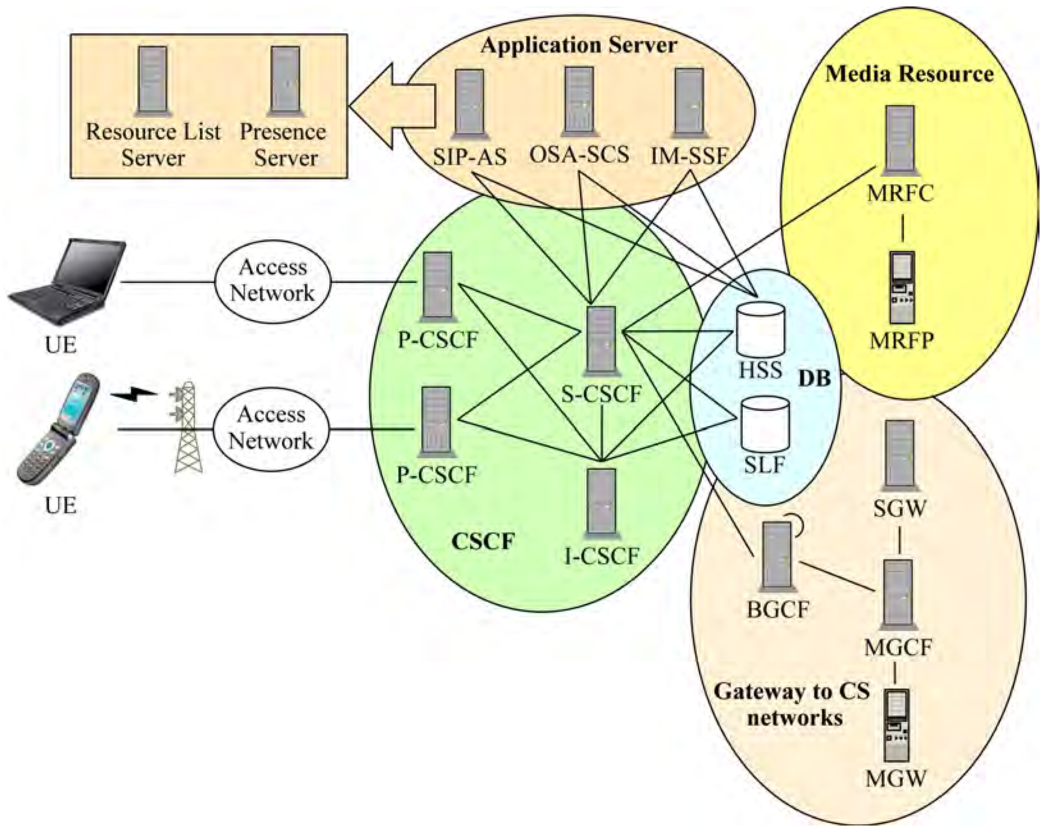


Figure 2: 3GPP IMS architecture overview

The book “The 3G IP Multimedia Subsystem (IMS). Merging the Internet and the Cellular Worlds“ [22] gives a good comprehensive introduction to the IMS.
Fig. 2 shows an overview of the IMS architecture. The nodes in Fig. 2 represent logical functions and not standardized physical nodes. These functions can be implemented in several nodes or a single node can combine several functions. On the left-hand side, Fig. 2 shows two different User Equipments (UEs), a mobile phone and a laptop. They attach to the network through a packet-switched access network. Examples for such access networks are General Packet Radio Service (GPRS) or DSL.
The IMS core is divided into Call/Session Control Functions (CSCFs), Ap-

plication Servers (ASs), Media Resource Functions (MRFs), Public Switched Telephone Network (PSTN)/Circuit-Switched (CS) gateways and databases (DBs).
The main database is the Home Subscriber Server (HSS), which stores all user-related information. This includes location information, security information, user profile information and the Serving-CSCF (S-CSCF) allocated to the user. Only networks with more than one HSS require a Subscription Locator Function (SLF). This database maps users' addresses to HSSs.
The CSCFs handle the SIP signaling in the IMS and therefore play an essential role. A CSCF is a SIP server and either a Proxy-CSCF (P-CSCF), Interrogating-CSCF (I-CSCF) or S-CSCF.
The P-CSCF is the first point of contact and all requests from or to an IMS terminal traverse this CSCF. Some tasks of the P-CSCF are related to security. For instance, the P-CSCF asserts the identity of a registered user to the rest of the network after a successful registration. Moreover, it verifies the correctness of SIP-requests, (de-)compresses SIP messages for a possibly narrowband channel to the terminal and generates charging information.
The I-CSCF is located at the edge of an administrative domain. Its address is listed in the DNS and messages which are targeted to this domain arrive at the I-CSCF. The I-CSCF interfaces the HSS, respectively SLF, to route incoming SIP requests to the assigned S-CSCF.
The S-CSCF is the central node in the IMS architecture. Every message traverses a S-CSCF, which routes the message to its destination directly or via one or several Application Servers. The S-CSCF also maintains the binding between a user and its location, authenticates the user, enforces policies and downloads the user profile from the HSS. The user profile includes the service profile which contains a set of triggers to route messages to one or several Application Servers.
The Application Servers host and execute services. Three different types of Application Servers exist and every type contains a SIP interface towards

the S-CSCF. A SIP AS is the native Application Server and provides IP multimedia services based on SIP. The Open Service Access-Service Capability Server (OSA-SCS) interfaces the S-SCSF with SIP on the one side but provides an interface to the OSA framework Application Server [13] on the other side. The IP Multimedia Service Switching Function (IM-SSF) acts in a similar way but allows reusing Customized Applications for Mobile network Enhanced Logic (CAMEL) [9] services. As shown in Fig. 2, a Presence Server (PS) and RLS are examples for SIP Application Servers. These servers are mandatory components of the presence service, which we will discuss in Section 2.2.

The Media Resource Function serves as source of media in the home network. For instance, the MRF is in charge of playing announcements, mixing media streams or transcoding between different codecs. Sessions originated by an IMS terminal to a user in a circuit-switched network are routed by the S-CSCF to a Breakout Gateway Control Function (BGCF). For more information on the MRF and BGCF, the reader may refer to [12] or [22].

2.1.3 Identities in the IMS

Fundamental to the IMS architecture is its integrated identity framework. While in pure SIP a public SIP URI[2] [73], known as Address of Record (AoR), identifies a user, the IMS differentiates between IP Multimedia Private Identities (IMPI) and IP Multimedia Public Identities (IMPU). TS 23.228 [11] defines these identifiers in Section 4.3.3.

Private User Identities take the format of a Network Access Identifier (NAI) as defined in RFC 2486 [18]. The format is `username@operator` (e.g., `joe.black@operator.com`). Private User Identities are used, for instance, for registration, authorization, administration and accounting but they are not used for routing.

[2]Alternatively, SIPS URIs [73] are used if messages are secured with TLS between network entities.

Public User Identities take the format of either a SIP URI (RFC 3261 [73]) or a tel URI (RFC 3966 [74]). An example of a SIP URI is:

```
sip:  joe.black@operator.com
```

An example of a tel URI is:

```
tel:  +1-222-555-123
```

Additionally, it is possible to include a phone number in a SIP URI (e.g., `sip:+1-222-555-0123@operator.com;user=phone`). The Public User Identity is used as contact information and, for instance, printed on business cards. Public User Identities are used to route SIP signaling and by any other user to request communication. The presence service, which we will discuss in Section 2.2, is entirely based on Public User Identifiers. But each Public User Identity must be registered with the IMS network whereby associated Private User Identities are involved.

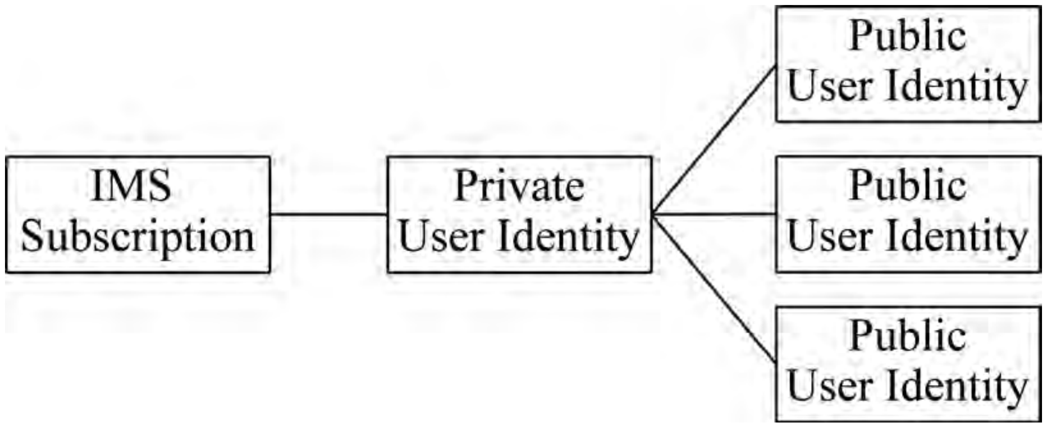


Figure 3: Relation of Private and Public User Identities in 3GPP R5 (source: [22], p. 41)

The network operator assigns Public and Private User Identities to each subscriber. 3GPP Release 5 allows only one Private User Identity per IMS subscriber but several Public User Identities can be assigned to a single Private User Identity (Fig. 3). For instance, a subscriber can have one Public User Identity for business use and another one for private use. In the case of UMTS, a smart card (i.e., an Universal Integrated Circuit Card (UICC)) stores the Private User Identity and at least one Public User Identity.

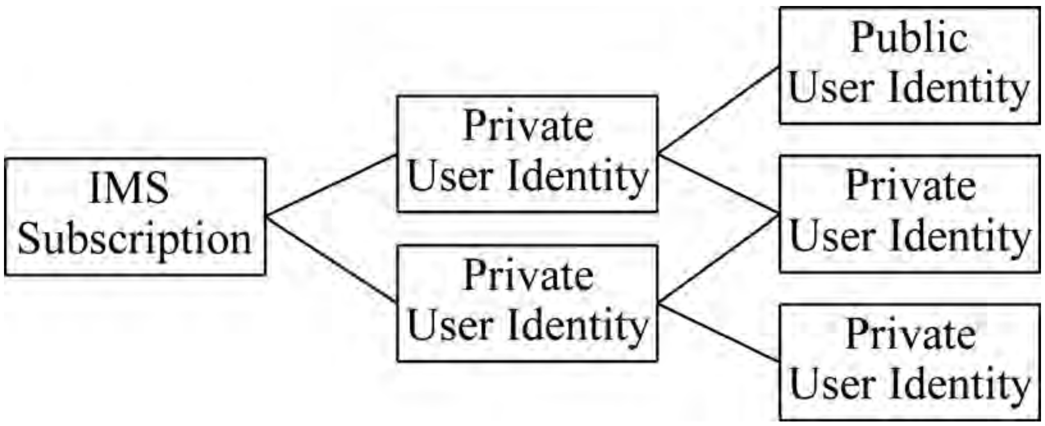


Figure 4: Relation of Private and Public User Identities in 3GPP R6 (source: [22], p. 42)

3GPP Release 6 has extended this relationship and allows more than one Private User Identity per IMS subscriber (Fig. 4). Public User Identities may be shared across several Private User Identities for the same subscriber. In the case of UMTS, the restriction of one Private User Identity per smart card is still valid but one subscriber may have several smart cards, which store different Private User Identities. It allows using the same Public User Identity simultaneously from different IMS devices.
A single IMS subscription can be valid for a group of people. For instance, a call center has a single subscription with an IMS service provider and every call-center agent gets his or her own terminal with a smart card that stores a unique Private User Identity. But all of them share the same Public User Identity (e.g., `sip:callcenter@example.com`).

2.2 Presence Service

As mentioned in the introduction, presence is one of the most promising future services. In [22], p. 295, we can find the following definition of presence: "Presence is the service that allows a user to be informed about the reachability, availability, and willingness of communication of another user." In the simplest case, the presence service indicates if a user is online or offline. A more complex service indicates, for instance, if a user is on the phone or in a meeting. Additionally, it can inform about possible communication means (e.g., audio, instant messaging, voice mail) and which terminals support these communication means. The OMA Presence SIMPLE requirements document [48] defines a presence service even more generic and gives as an example a radio station which publishes the songs currently playing as presence information.

2.2.1 Presence Service Specifications

For pure SIP, instant messaging and presence are enabled through specific SIP extensions, collectively known as SIMPLE. SIMPLE stands for the IETF's SIP for Instant Messaging and Presence Leveraging Extensions Working Group [6].
Based on the RFCs defined in the SIMPLE working group, both 3GPP (in 3GPP TS 23.141 [14]) and the Open Mobile Alliance (OMA) have defined their own presence service specifications. OMA is a standardization body which develops open standards to provide interoperable mobile data services. OMA's efforts result in Release Packages which contain a set of OMA specifications. The OMA presence service is specified by the Enabler Release Package for OMA Presence SIMPLE [45].
OMA does not extend the IMS but communicates new requirements on the IMS to 3GPP. This avoids having different IMS versions. However, sometimes there is no clear cut between the IMS and the services on top of it, as it is the case for presence.

In its work, OMA focuses on usability and presumes the existence of a network technology (e.g., IMS) specified by outside parties. The specifications are agnostic to the particular (cellular) network technologies being used. This distinguishes the presence service specifications of OMA and 3GPP. The specifications are compatible but 3GPP includes specifications on the network layer, while OMA restricts their scope to the application layer. The 3GPP specification addresses, in contrast to OMA, the communication between functional elements of the presence service (e.g., Presence Server) and various elements of the 3GPP network architecture (e.g., Gateway GPRS Support Node (GGSN)). Since the primary focus of this thesis is on the service aspect, we concentrate on the OMA Presence SIMPLE enabler specification.

2.2.2 Presence Service Concepts

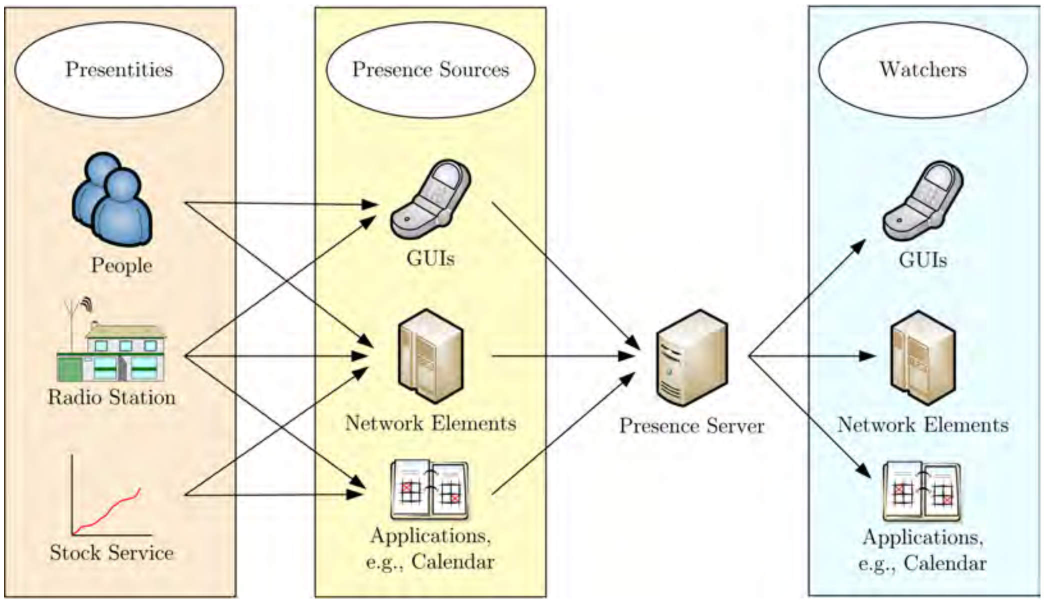


Figure 5: Presence service components

Fig. 5 shows various roles the OMA Presence SIMPLE enabler defines or, more precisely, mainly reuses from different IETF Request for Comments (RFCs) and 3GPP specifications.

A **Presentity** is a "[...] logical entity that has presence information [...] associated with it" ([48], p. 8) and is a short name for presence entity. Most commonly, a single person plays the role of a Presentity but it may also reference, for instance, a group of people, a help desk, a radio station or a conference room. Most likely, a Presentity is referenced using a *sip:*, *pres:* or *tel:* URI.

A Presentity publishes its presence information with the help of one or several **Presence Sources**. Examples for Presence Sources are a piece of software on a user equipment (e.g., on a laptop), a network element (e.g., GGSN) or a 3rd party application (e.g., an external calendar application). The Presence Sources send the presence information to a Presence Server (PS). This logical entity, in turn, makes this information available to Watchers.

A **Watcher** is "any uniquely identifiable entity that requests presence information about a Presentity, or watcher information about a Watcher, from the presence service" ([48], p. 9). Watcher information identifies information about a Watcher and may be of interest for Presentities. We distinguish three different types of Watchers: fetcher, poller, and subscribed-watcher. A fetcher asks the presence information of one or more Presentities only once. A poller requests presence information on a regular basis. A subscribed-watcher, by contrast, requests notifications about future changes in the Presentity's presence information.

The **Presence Server** has more responsibilities than just relaying presence information from Presence Sources to Watchers. First, it must compose the presence information of different sources into one single document (we detail on this topic in Section 5.1). In a next step, the Presence Server filters this raw document according to Watcher specific privacy rules. Some Watchers may be allowed to see only some parts of the presence information or some information may be adapted (e.g., show an offline status to some Watchers even if the Presentity is online). The resulting documents are transformed on per-watcher rules which include, for example, filtering and partial notification. Eventually, the Presence Server sends the final notifications to the Watchers.

2.2.3 Presence Service Architecture

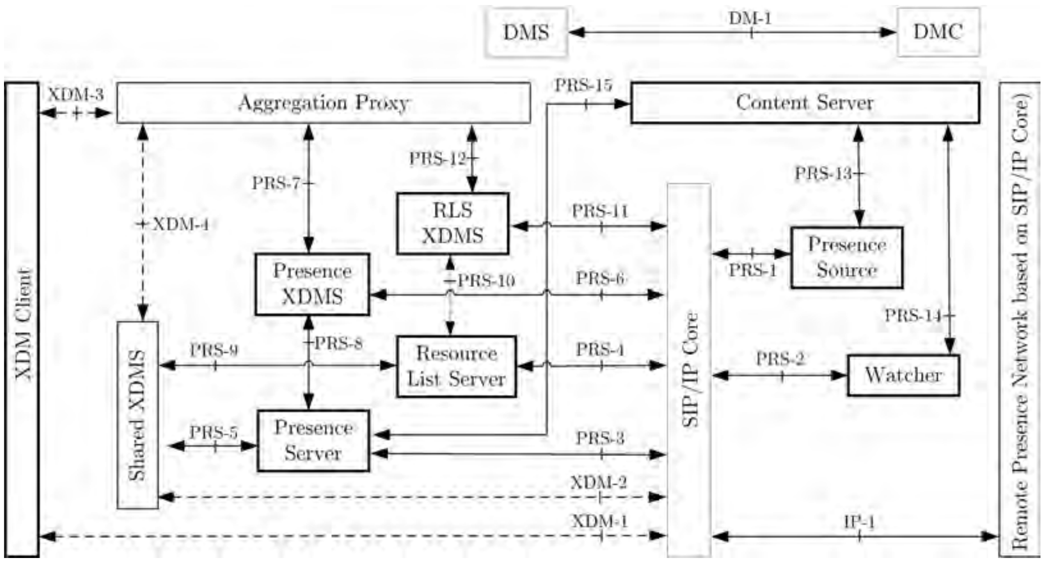


Figure 6: OMA Presence SIMPLE reference architecture (source: [47], p. 12)

Fig. 6 shows the Presence SIMPLE reference architecture as defined in the OMA Presence SIMPLE architecture document [47]. Bold elements and reference points with solid lines are defined in the OMA Presence SIMPLE architecture itself whereas the remaining elements are external to this specification. The XDM elements are specified in the XDM Enabler and discussed in Section 2.3.
The functionality of the entities Presence Source, Watcher, and Presence Server has already been discussed in Section 2.2.2. The Presence Source is either located in a user terminal or within a network entity. As mentioned

in the introduction, a Resource List Server (RLS) allows subscriptions to resource lists (see Section 2.3.5) and enables a Watcher application to get notified about the presence information of all Presentities in a presence list using one single subscription. We will discuss the concept of a RLS in Section 3. Beside the tasks mentioned in Section 2.2.2, the Presence Server is able to subscribe to changes to documents stored in the Shared and Presence XML Document Management Servers (XDMSs) and fetches XML documents from there. In an analogous manner, the RLS accesses the Shared XDMS and RLS XDMS. The Presence and RLS XDMS are special types of enabler specific XDMS. We discuss Shared and enabler specific XMDS in Section 2.3.1. The Presence XDMS stores presence authorization rules while the RLS XDMS stores Presence Lists. We discuss these XML data formats in Section 2.3.5.
The Content Server is capable of managing Multipurpose Internet Mail Extensions (MIME) objects for Presence. It allows Presence Sources and the Presence Server to store objects within the Content Server and link to those objects.
The communication between Presence Sources, Presence Server and Watchers is built on top of the SIP event notification framework (RFC 3265 [61]). However, this framework cannot be used directly but is extended by additional specifications, so-called event packages. The event package *presence* (RFC 3856 [63]) is one example. The SIP methods SUBSCRIBE and NOTIFY [61] handle event notification in SIP and are used for the communication between Presence Server and Watchers. Presence Sources report their presence information with so-called PUBLISH requests, which are defined in RFC 3903 [38].
Fig. 7 depicts an exemplary message flow. A Watcher sends a SUBSCRIBE request (1) to the Presence Server and receives a 200 OK (2) and an immediate mandatory NOTIFY request (3). The Request URI of the SUBSCRIBE request (i.e., *sip: alice@ex.org*) identifies the Presentity the Watcher wants to know the presence information of. Whenever presence information changes

the Presence Source sends a PUBLISH request. This triggers sooner or later another NOTIFY request to inform the Watcher about the change. Depending on the value of the Expires header field in the SUBSCRIBE message, the Presence Server keeps sending NOTIFY requests to the Watcher for a certain amount of time. The bodies of the respective PUBLISH and NOTIFY requests contain the presence information.

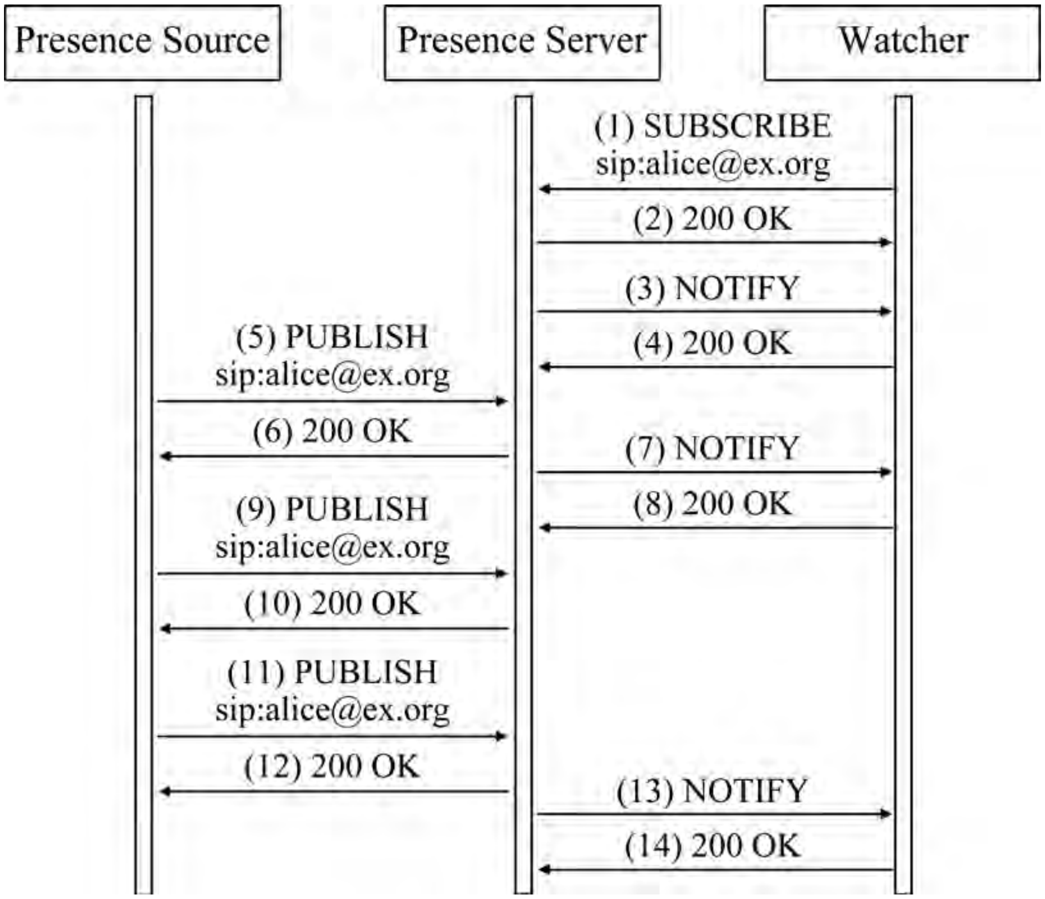


Figure 7: PUBLISH/SUBSCRIBE/NOTIFY example message flow

2.3 OMA XML Document Management Enabler

Many OMA enablers, like Presence [45] or Instant Messaging [43], need to access and manipulate certain information they require for their services. The OMA XML Document Management enabler [46] "defines a common mechanism that makes user-specific service-related information accessible to the service enabler that need them." ([53], p. 10). This information is defined in well-structured XML documents stored in the network. Two main features of XDM are:

- Principals[3] can store, access and manipulate information stored in XML files by means of the XML Configuration Access Protocol (XCAP) [72].
- Principals can be informed about changes to documents by means of the SIP SUBSCRIBE/NOTIFY mechanism.

The following subsections examine the XDM architecture (Section 2.3.1), its functionality (Section 2.3.2), security issues related to XDM (Section 2.3.3) and common application usages (Section 2.3.4). Section 2.3.5 discusses presence related application usages.

2.3.1 XDM Architecture

Fig. 8 shows the functional entities of XDM and the interfaces between them. Entities in bold boxes and reference points with solid lines are specified by the XDM enabler [46]. All other entities and reference points are subject to definition by specific enablers (e.g., Presence). The following list summarizes the functionalities of these entities. The reader can refer to [53] for the detailed specification.

[3] "An entity that has an identity, that is capable of providing consent and other data, and to which authenticated actions are done on its behalf. Examples of principals include an individual user, a group of individuals, a corporation, service enablers/applications, system entities and other legal entities" ([53], p. 8).

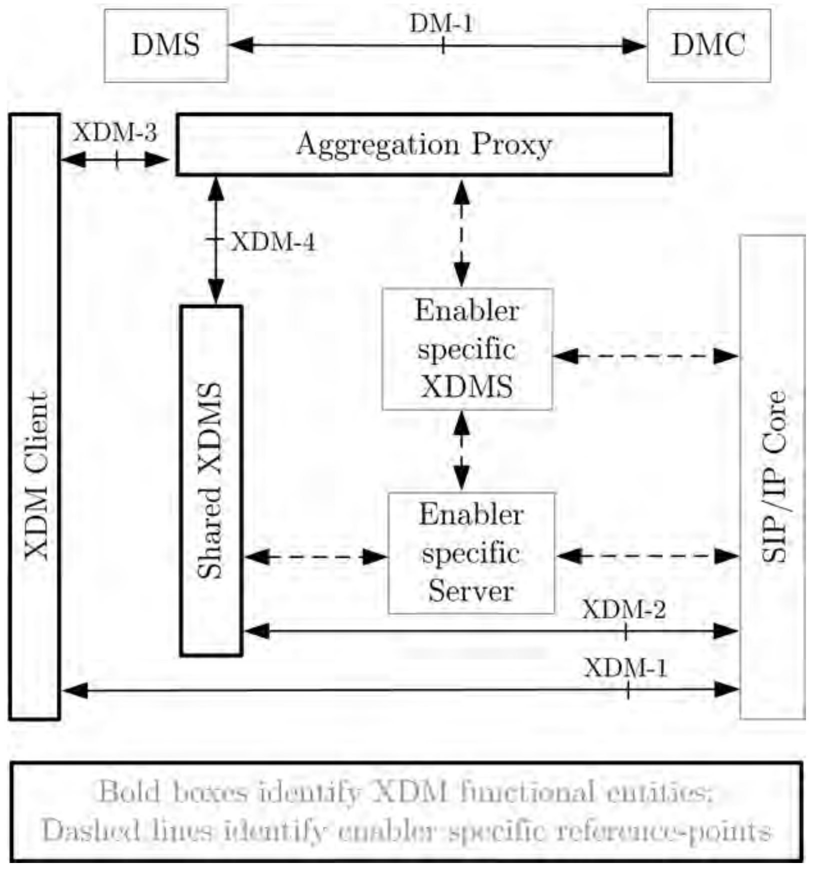


Figure 8: XML Document Management architecture (source: [53], p. 14)

XDM Client: XDM Clients (XDMCs) can be implemented in terminals or servers and provide access to XDM Servers. They support (possibly a subset) of the features listed in Section 2.3.2.

Aggregation Proxy: An Aggregation Proxy is a contact point for XDMCs implemented in a terminal. Its tasks are:

- Authentication of XDM Clients
- Routing of XCAP requests to the correct XDM Server
- Charging (optional)

- Compression/decompression of XCAP requests (optional)

Enabler specific XDMS: An enabler specific XDM Server stores documents specific to a certain service enabler and performs the following functions:

- Authorization of XCAP and SIP requests
- Managing of XML documents
- Aggregation of notifications of changes in (several) documents
- Notification of subscribers about changes in documents

The Presence XDMS and RLS XDMS (see Section 2.2.3) are examples of such enabler specific XDM Servers.

Shared XDMS: In contrast to enabler specific XDM Servers, which manage XML documents specific to a certain service enabler, a Shared XDM Server manages documents which can be reused by several other enablers. According to the OMA XDM enabler, a Shared XDMS only supports URI Lists (see Section 2.3.5).

Enabler specific Server: The functionality of an enabler specific Server is defined by the specific enabler. In the case of the OMA Presence SIMPLE enabler the Presence Server and Resource List Server (see Section 2.2.3) are enabler specific Servers.

SIP/IP Core: The XDM enabler uses IMS or 3GPP2 CDMA2000 Multimedia Domain (MMD) networks. The IMS supports the XDM Service with the following functions (quoted from [53], p. 15 et seq.):

- Routes the SIP signaling between the XDM Client and the XDM servers
- Provides discovery and address resolution services
- Supports SIP compression

- Performs a certain type of authorization of the XDM Client based on the user's service profile
- Maintains the registration state
- Provides charging information

DMS: "The Device Management Server [42] allows initializations and updates of all configuration parameters necessary for XDMC." ([53], p. 16)

DMC: The Device Management Client (DMC) communicates with the Device Management Server (DMS) to initialize and update all configuration parameters.

Tab. 1 summarizes all reference points of the XDM architecture. Since the DM-1 reference point between a DM Client and Server has a somehow separated functionality, it is not included in Tab. 1. The DM-1 reference point is specified in [42] and the necessary configuration objects are defined in [44].

2.3.2 XDM Functionalities

The main functionalities of XDM are manipulation and change notification of XML documents.

Manipulation of XML Documents: XDM supports the following operations on a XML document (quoted from [53], p. 12):

- Creating or replacing a document
- Deleting a document
- Retrieving a document
- Creating or replacing an XML element
- Deleting an XML element

Reference Point	**Protocol**	**Functionality**	**Specification/IMS-Interface**
XDM Client - SIP/IP Core (XDM-1)	SIP	Subscribe/Notify	3GPP TS 23.002 [12]/ISC interface (XDMC in a server entity), Gm interface (XDMC in a terminal)
Shared XDM - SIP/IP Core (XDM-2)	SIP	Subscribe/Notify	3GPP TS 23.002 [12]/ISC interface
XDM Client - Aggregation Proxy (XDM-3)	XCAP	XML document management, mutual authentication	3GPP TS 23.002 [12]/Ut interface
Shared XDMS - Aggregation Proxy (XDM-4)	XCAP	Shared XML document management	RFC 4825 [72]
Enabler specific XDMS - Aggregation Proxy	Enabler specific	Enabler specific XML document management	Enabler specific
Enabler specific XDMS - SIP/IP Core	SIP	Subscribe/Notify	Enabler specific
Enabler specific XDMS - Enabler specific server	Enabler specific	Transfer of enabler specific documents from XDMS to enabler specific Server	Enabler specific
Shared XDMS - Enabler specific server	XCAP	Transfer of URI Lists to enabler specific Server	Enabler specific
Enabler specific Server - SIP/IP Core	SIP	Subscribe/Notify	Enabler specific

Table 1: Reference points (XDM architecture)

- Retrieving an XML element
- Creating an XML attribute for an XML element
- Deleting an XML attribute
- Retrieving an XML attribute

It is not necessary that a certain document supports all operations.

The document tree on a XDM Server is divided into a global tree for global documents and a user tree for user-specific documents. Documents, and XML elements and attributes within a document, are addressed with XCAP URIs (see RFC 4825 [72]). A generic XCAP URI for the user tree has the following format:
`[XCAPRootURI]/[AUID]/users/[XUI]/[docPath]/~~/[nodeSelector]`
For a XDM Client implemented in a terminal a Domain Name Server (DNS) must resolve the hostname of the XCAP root to the aggregation server. Every application which makes use of XCAP defines its own application usage, which is identified by an Application Unique ID (AUID). Along with other key pieces of information, an application usage defines the XML schema for the data used by the application. The XCAP User Identifier (XUI) [72] identifies a specific user and is either a SIP URI or a tel URI, followed by the specific path to the XML document. Subsequent to the document selector an optional node selector selects an element or attribute within the document. Fig. 13 in Section 3.2 shows an example of a XCAP request.

Change Notifications of XML Documents: The second main feature of XDM is to inform entities about changes to XML documents. XDM utilizes the SIP SUBSCRIBE/NOTIFY mechanism [61] in combination with the User Agent (UA) profile change event package [57].

This event package is part of a framework "for discovery, delivery, notification and updates of user agent profile data" ([57], p. 1) and is defined in draft-ietf-sipping-config-framework [57]. The event-package token name for this package is *ua-profile*.

The draft defines several parameters, which can be added to the Event header field in SUBSCRIBE/NOTIFY requests. Tab. 2 summarizes these parameters together with the extensions defined in [58] (see below).

Parameter	**Message**	**Description**	**Value(s)**
profile-type	SUBSCRIBE	Token value of the profile type	"device","user","local-network", "application"
vendor	SUBSCRIBE	Specified by implementer of the user agent	Quoted-string, should include DNS domain name for uniqueness
model	SUBSCRIBE	Specified by implementer of the user agent	Quoted-string
version	SUBSCRIBE	Specified by implementer of the user agent	Quoted-string
network-user	SUBSCRIBE/ NOTIFY	User's Address of Record	network user address
effective-by	NOTIFY	Max. seconds before new profile becomes effective	Digit(s)
auid	SUBSCRIBE/ NOTIFY	Selects all documents under this auid for a specific user	Quoted-string
document	SUBSCRIBE/ NOTIFY	Selects a document	Quoted-string

Table 2: Parameters of the event package *ua-profile*

The draft draft-ietf-sip-xcap-config [58] defines an extension to the User Agent (UA) profile change event package. The document introduces the new profile-type *application* and two new Event header parameters: *auid* and *document*. A subscriber sets the parameter *profile type* to *ap-*

plication when it wants to get informed about document changes. If no *auid* parameter is present, the subscriber requests to be informed about changes to all documents both in the global and home tree for all application usages associated with the identity given in the Request URI of the SUBSCRIBE message. If the *auid* parameter is present, the subscription informs about changes to all documents for the given AUID. If the *document* parameter is present, notifications about changes to the specific document will be made. The *document* parameter contains a XCAP document selector to a complete document.

The body of a NOTIFY message contains a well-formed XML document in the xcap-diff document format (draft-ietf-simple-xcap-diff [68]). Such a XML document includes - for every changed document - an identifier, the former and new entity tags (XCAP uses them for versioning) and potentially the changes to the document in the XML patch format. The patch instructions follow the specifications of draft-ietf-simple-xml-patch-ops [82] and allow constructing the current document. The initial NOTIFY of a SUBSCRIBE/NOTIFY dialog informs the Watcher about the current version of the XML document(s) and is used as reference for further notifies.

2.3.3 XDM Security Issues

This section describes security mechanisms between a XDM Client in a terminal and an Aggregation Proxy. Security mechanisms between the Aggregation Proxy and a XDM Server or between an Application Server and a XDM Server are defined in the corresponding 3GPP specifications and are not part of the OMA XDM enabler.
In case of the IMS, the mutual authentication between a XDMC and an Aggregation Proxy follows the mechanism defined for Presence in TS 33.141 [15]. If present, the Generic Authentication Architecture (TS 33.222 [10]) is used and the procedures of TS 24.109 [8] are followed. The Aggregation

Proxy first verifies if the XDM Client has inserted a valid identity in the X-3GPP-Intended-Identity header field [8] and if no header is present, the proxy inserts the authenticated identity in a X-3GPP-Asserted-Identity header field [8].
If the Generic Authentication Architecture is not present, HTTP Digest authentication (RFC 2617 [29]) is used. The *username* parameter of the Authorization header is set to a Public User Identity (SIP or tel URI) which matches the XUI for XCAP. Upon a successful authentication, the Aggregation proxy inserts a X-XCAP-Asserted-Identity header field to the XCAP request. This OMA specific extension header is defined in the XDM specification [54] and contains the asserted identity.
Transport Layer Security (TLS) conforming to RFC 2818 [60] provides integrity and confidentiality protection of the exchanged messages.
The default access control policy of XCAP is also applied to XDM. Only the creator of a document is allowed to perform all actions. Application Servers of the trusted network are allowed to read documents but other entities have no access at all. Permission-based systems are subject to future releases. For global documents each application usage defines the authorization policy.

2.3.4 Common Application Usages

Section 6.7 of the OMA XDM specification [54] quotes two common application usages.

XCAP server capabilities: XCAP server capabilities enables clients to know which extensions, application usages or namespaces a XCAP server supports. For this purpose the XCAP standard [72] introduces the XCAP server capabilities application usage with the AUID *xcap-caps*. The XCAP specification [72] requires that every XCAP server supports this application usage and alike the XDM specification [54] requests that every XDM server must support this application usage.

XML Documents Directory: A new application usage with the AUID *org.openmobilealliance.xcap-directory* allows XDM Clients to fetch - for a given XUI (quoted from [54], p.24):

1. the list of all XCAP managed documents corresponding to that XUI across all XDMSs, or
2. the list of all documents for a given AUID corresponding to that XUI stored in an XDMS.

Therefore, every XDMS maintains a document named *directory.xml* for each XUI. For every AUID, this document contains a list of the documents managed by this XDMS together with some additional information (i.e., etag, last-modified, size) for each document. Upon a request from a XDM Client the Aggregation Proxy aggregates the responses from all XDM Servers before forwarding a single response to the XDM Client.

2.3.5 Presence Related Application Usages

Beside the common application usage discussed in Section 2.3.4, the OMA XDM enabler defines one application usage for resource lists (i.e., so-called URI Lists) [52]. The Shared XDM server handles this application usage. The OMA Presence SIMPLE enabler specifies two application usages: one for Presence Authorization Rules [50] and one for Presence Lists [51] used by Resource List Servers.

Presence Authorization Rules: The architecture document of the OMA Presence SIMPLE enabler [47] divides Watcher authorization into subscriber authorization and presence content authorization. The Presence XDM specification [50] defines a single application usage for both. The AUID is *org.openmobilealliance.pres-rules* and both subscriber and presence content authorization rules are defined in a XML document named *pres-rules*. Such a document exists for every user; no global documents are used.

This application usage reuses the document structure for presence rules defined in draft-ietf-simple-presence-rules [71]. Listing 1 shows an example of such a document. A document contains a set of rules and each rule has three parts:

- Conditions - conditions to apply this rule
- Actions - actions the server takes
- Transformations - transformations to presence information

The element `<identity>` defines conditions which restrict a rule to a single identity or a group of identities (e.g., everybody excepts...) [79]. Additionally, the OMA XDM specification [54] extends the allowed elements by `<external-list>`, `<anonymous request>` and `<other-identity>`. The `<external-list>` element allows the matching of all identities in a URI List. The `<anonymous request>` element allows the matching of requests identified as anonymous and the `<other-identity>` element allows the matching of all identities which do not match any other rule. This allows a default policy.

The only allowed action element is `<sub-handling>`. It defines the subscription authorization decision (e.g., confirm, polite-block). Section 3.2.1 of [71] gives the complete list of possible decisions.

Transformations define which elements of the presence information a Watcher can see. The elements `<provide-persons>`, `<provide-devices>` and `<provide-services>` grant access to the information about persons, devices and services of a certain class or instance (see Section 2.4.3). Other elements make particular elements of the presence information visible (e.g., `<provide-network-availability>`,

`<provide-mood>`). Section 5.1.2.1 of [50] gives the complete list of possible transformations.

Note that not all child elements of `<conditions>`, `<actions>` or `<transformations>` defined in [71] are defined for the the OMA application usage (e.g., `<sphere>` is not defined).

```
<?xml version="1.0" encoding="UTF-8"?>
<cr:ruleset
  xmlns:op="urn:oma:xml:prs:pres-rules"
  xmlns:pr="urn:ietf:params:xml:ns:pres-rules"
  xmlns:cr="urn:ietf:params:xml:ns:common-policy">
  <cr:rule id="ck81">
    <cr:conditions>
      <cr:identity>
        <cr:one id="tel:+43012345678"/>
        <cr:one id="sip:lisi@ibkt.tuwien.ac.at"/>
      </cr:identity>
    </cr:conditions>
    <cr:actions>
      <pr:sub-handling>allow</pr:sub-handling>
    </cr:actions>
    <cr:transformations>
      <pr:provide-services>
        <op:service-id>org.openmobilealliance:PoC-session</op:service-id>
      </pr:provide-services>
      <op:provide-willingness>true</op:provide-willingness>
      <pr:provide-status-icon>true</pr:provide-status-icon>
    </cr:transformations>
  </cr:rule>
</cr:ruleset>
```

Listing 1: Example of Presence Authorization Rules

Presence Lists: The RLS XDM specification [51] defines an application usage for Presence List documents. The structure of such a document is identical with the structure of a RLS-services document defined in Section 4 of draft-ietf-simple-xcap-list-usage[4] [65].

[4]The latest version is RFC 4826 [70] but the studied version of the RLS XDM specification [51] cites the draft in version 5.

The AUID of this application usage is *rls-services* and the Presence List for a particular user is stored under his/her directory in a file named *index*. There also exists an index file in the global tree which is an aggregation of all index files in the user tree. This makes it easier for a RLS to find the `<service>` element for a particular URI.

Listing 2 shows an example of a Presence List. A Presence List document consists of a sequence of `<service>` elements. Each `<service>` element has a *uri* attribute, which contains the identifier a client can subscribe to in order to use this service. A `<service>` element includes either a `<resource-list>` element, which contains a link to a resource list, or directly a list of URIs with the help of the `<list>` element (see next paragraph). In addition, each service definition includes a `<packages>` element. The child elements of `<packages>` are `<package>` elements, which specify SIP event packages supported by this service.

The only differences between the definition of a RLS-service document in draft-ietf-simple-xcap-list-usage [65] and a Presence-List document defined in the RLS XDM specification [51] are that the latter must include at least one `<package>` element as child of `<packages>` and must at least support the presence event package.

```
<?xml version="1.0" encoding="UTF-8"?>
  <rls-services xmlns="urn:ietf:params:xml:ns:rls-services"
      xmlns:rl="urn:ietf:params:xml:ns:resource-lists"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
   <service uri="sip:mybuddies@ibkt.tuwien.ac.at">
     <resource-list>
       http://ibkt.tuwien.ac.at/resource-lists/users/sip:skytale@ibkt.tuwien
           .ac.at/index/~~/resource-lists/list%5b@name=%22friends%22%5d
     </resource-list>
     <packages>
      <package>presence</package>
     </packages>
   </service>
  </rls-services>
```

Listing 2: Example of a Presence List

URI Lists and Group Usage Lists: The URI List application usage with the AUID *resource-list* is defined by the Shared XDM specification [52]. This application usage is not enabler specific but used whenever there is the need to refer to a group of resources. Therefore, the documents are stored on a special XDM Server named Shared XDMS (see Section 2.3.1). For each user, there is a single file for all shared URI Lists in his/her directory called *index*.

The document format is based on resource-list documents which are specified in Section 3 of draft-ietf-simple-xcap-list-usage [65]. A resource-list document contains one or more `<list>` elements, one for each resource list. A list can recursively contain other lists which allows a hierarchical structuring. Listing 3 shows an example of a URI List.

Allowed child elements of `<list>` are `<entry>`, `<entry-ref>` and `<external>`. A particular resource is identified by an `<entry>` element. It has a single mandatory attribute *uri* which names the URI of the resource. The other two elements, `<external>` and `<entry-ref>`, include a reference. An `<entry-ref>` element contains a relative URI to an entry of another list within the same XCAP root, while an `<external>`

element refers to a complete list on any XCAP server. Optionally, the child element `<display-name>` specifies a name for a resource or a list.

Moreover, the OMA specification [52] makes the optional *name* attribute of a `<list>` entry compulsory. The name has the format oma_xyz and the Shared XDM specification lists four allowed names: *oma_allcontacts*, *oma_buddylist*, *oma_pocbuddylist* and *oma_blockedcontacts*. Other enablers can define new names. The Open Mobile Naming Authority (OMNA) URI-List Usage Name Registration [41] keeps a list of all valid names.

Additionally, the Shared XDM specification [52] defines a new child element of `<list>` with the name `<appusages>`. “The `<appusages>` element contains the node URI and the AUID value of those application usages referring to the `<list>` element” ([52], p. 8).

The Shared XDM specification [52] also defines a second application usage which may be supported. It is called Group Usage List with the AUID *org.openmobilealliance.group-usage-list* and allows a client to maintain a list of group names or service URI it knows. The structure again complies with the structure of a resource-list document specified in [65]. Additionally, the specification introduces the `<uriusage>` element as new child element of `<entry>`. This element indicates what the *uri* attribute of the `<entry>` element is used for. It is an abstract type and every application usage defines the `<uriusage>` element. Furthermore, the elements `<external>` and `<entry-ref>` are not allowed in order to reduce complexity.

In this context, a common extension which is specified in Section 6.6 of the OMA XDM specification [54] is worth mentioning . The specification defines the `<external>` element which allows various application usages to refer to URI Lists stored in the Shared XDMS. The element contains either a XCAP document URI pointing to a resource-list document or a XCAP node URI pointing to a `<list>` element within a resource-list document.

```
<?xml version="1.0" encoding="UTF-8"?>
  <resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
      <list name="oma_buddylist">
        <display-name>colleagues</display-name>
        <entry uri="sip:alice@ibkt.tuwien.ac.at">
          <display-name>Alice</display-name>
        </entry>
        <entry uri="sip:christoph@ibkt.tuwien.ac.at"/>
        <external anchor="http://xcap.ibkt.tuwien.ac.at/resource-lists/users
            /sip:marco@ibkt.tuwien.ac.at/index/~~/resource-lists/list%5
            b@name=%22support%22%5d">
            <display-name>Technical Support</display-name>
        </external>
        <entry-ref ref="resource-lists/users/sip:marco@ibkt.tuwien.ac.at/
            index/~~/resource-lists/list%5b@name=%22list1%22%5d/entry%5b@uri
            =%22sip:bernhard@ibkt.tuwien.ac.at%22%5d"/>
       </list>
   </resource-lists>
```

Listing 3: Example of a URI List

2.4 Presence Data Formats

In the previous section we have discussed the architecture and protocols to manage presence information. But common protocols alone are not sufficient. In order to achieve interoperability between different presence systems, it is necessary to agree on common data formats for presence information. This section presents such presence data formats.

2.4.1 Data Formats for 3GPP and OMA Presence

The requirement document [48] of the OMA Presence SIMPLE enabler lists building blocks of presence information that must be supported. In Section 10.3 and Section 10.4 of the OMA Presence SIMPLE specification [49] these building blocks are mapped to elements of several presence formats. Table

3 lists the supported data formats. If no mapping to existing elements is possible, OMA-specific extensions to the Presence Information Data Format (PIDF) are used. The complete list of OMA-specific PIDF extensions is available from [40].
The OMA specification [49] allows Presence Sources to publish elements from other PIDF extensions as long as a Watcher can ignore the extensions it does not understand and the meaning of the elements the Watcher understands does not change. To allow easy registration of new presence information packages, the Open Mobile Naming Authority (OMNA) [40] maintains a registry of presence information packages.

Name	**Standard**	**OMA Support**	**3GPP Support**
Presence Information Data Format (PIDF)	RFC 3863 [81]	m	m
Presence Data Model	RFC 4479 [66]	m	m
Rich Presence Information Data Format (RPID)	RFC 4480 [77]	m	m
Location Information	RFC 4119 [55]	m	o
OMA-specific extensions to PIDF	OMNA [40]	m	-
Contact Information for the PIDF (CIPID)	RFC 4482 [75]	-	o
SIP User Agent Capabilities	draft-ietf-simple-prescaps-ext [34]	-	o

Table 3: Supported presence data formats (m: mandatory, o: optional, -: unmentioned)

Table 3 also lists the presence data formats supported by 3GPP. 3GPP TS

24.141 [16] mentions in Section 5.3.1.2 the formats a presence source[5] must implement. Additional extensions can also be used but their usage is out of the scope of TS 24.141 [16].
If a Presence Source wants to support partial publication of presence information, it reverts to the following specifications:

- Publication of Partial Presence Information (draft-ietf-simple-partial-publish [39])
- PIDF Extension for Partial Presence (draft-ietf-simple-partial-pidf-format [35])

The following subsections give an overview of the mentioned presence formats.

2.4.2 Presence Information Data Format (PIDF)

The intention of the PIDF [81] is to provide a minimal set of presence status values to enable interoperability between different presence systems and to provide a framework for extensions. All other presence formats we discuss in the following subsections are defined as extensions to the PIDF. Extensions must be defined in a way that, if a client does not support these extensions, it can still understand the minimal mandatory set of presence values defined in the PIDF.
RFC 3863 [81] introduces the new content type *application/pidf+xml* for a XML MIME entity that contains presence information. A PIDF object itself is a well-formed XML document which contains zero or more `<tuple>` elements. Each `<tuple>` element contains (without optional extensions) a `<status>` element, an optional `<contact>` element, storing a URL of the

[5]In 3GPP and IETF SIMPLE specifications, the term *Presence User Agent* is used instead of *Presence Source*. A Presence User Agent is one type of Presence Source. We use the term *Presence Source* in this thesis.

contact address, an optional `<node>` element for human readable information and an optional `<timestamp>` element.
To describe the status of a Presentity the PIDF only defines the `<basic>` element that allows differentiating between open (i.e., online) and closed (i.e., offline). Other status values may be included through extensions. Listing 4 shows an example of a PIDF document.

```
<?xml version="1.0" encoding="UTF-8"?>
 <presence xmlns="urn:ietf:params:xml:ns:pidf"
           entity="pres:alice@ibkt.tuwien.ac.at">
    <tuple id="sg89ae">
      <status>
        <basic>open</basic>
      </status>
      <contact priority="0.8">tel:+09012345678</contact>
      <note>My cell phone</note>
    </tuple>
 </presence>
```

Listing 4: Example of a PIDF document

2.4.3 A Data Model for Presence

The PIDF models presence information as a series of tuples but it gives no guidance how to map real-world communication systems into a presence document. The data model for presence (RFC 4479 [66]) closes this gap.
Fig. 9 shows the data model. A Presentity is identified by a URI, which is, in case of SIP, often the Address of Record of the user or, in case of IMS, the Public User Identifier. The data model has three central elements: **person**, **service** and **device**. Each attribute is attached to a person, a service or a device element because it describes a person, a service or a device. The attributes of these components are divided into characteristics and status. Characteristics describe properties which usually do not change over time while status describes dynamic information.
A **person** component models information about the end-user (which could be a conference room or a call center in reality). There can be only one person

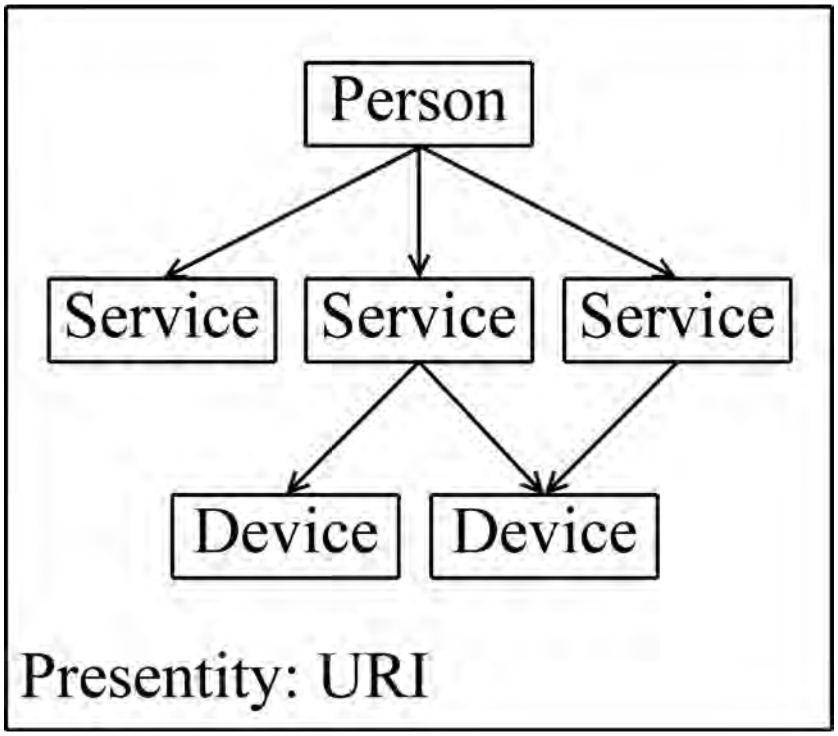


Figure 9: The SIP presence data model (source: [22], p. 300)

component per Presentity. A characteristic for a person is, for instance, his birthday. His mood (e.g., happy) or his current activity (e.g., on the phone) are examples for his status.

A **service** component represents a point of reachability for communication (e.g., telephony, instant messaging, push-to-talk). Characteristics describe the nature and capabilities of a service. Examples of characteristics include used media or supported SIP extensions. A helpful characteristic is the device ID, which links to the device the service is running on. The reach information for a service gives information on how to contact this service. This might be a simple URI or more complex. The reach information also helps to decide what to model as service. A service which can be reached independently from other services should be modeled as a separate service. The status

of a service is in the simplest form the basic status (open or closed). All information about a service should help a Watcher to decide how likely a communication will succeed or when it should start a communication, if not now.
A **device** component models the physical environment a service runs on (e.g., a cell phone, a laptop, a PC). Examples for characteristics of a device are physical dimension, size of its display or speed of its CPU. Examples for status are whether a device is turned on or off, the amount of battery power left or the geographic location of a device.
RFC 4479 [66] also defines some guidelines which extensions of the data model must follow. For instance, extensions must define to which data components new attributes are applicable.

2.4.4 Mapping between the Data Model and the PIDF

Since the PIDF is the minimum standard all presence systems agree on, there is a need to map the data model to the PIDF. For it, the data model uses existing elements and extends the PIDF when necessary. Therefore, RFC 4479 [66] introduces the new `<person>` and `<device>` elements. If a Watcher does not understand these extensions, it will ignore them and only process the PIDF information. Service components are mapped to the `<tuple>` element, which is defined in the PIDF. Fig. 10 shows the mapping of the data model to the PIDF.

2.4.5 Rich Presence Information Data Format (RPID)

The Rich Presence Information Data Format (RPID), specified in RFC 4480 [77], defines additional presence attributes to describe the person, service and data elements defined in RFC 4479 [66]. The main goals are:

- to be able to describe information comparable to current commercial presence systems

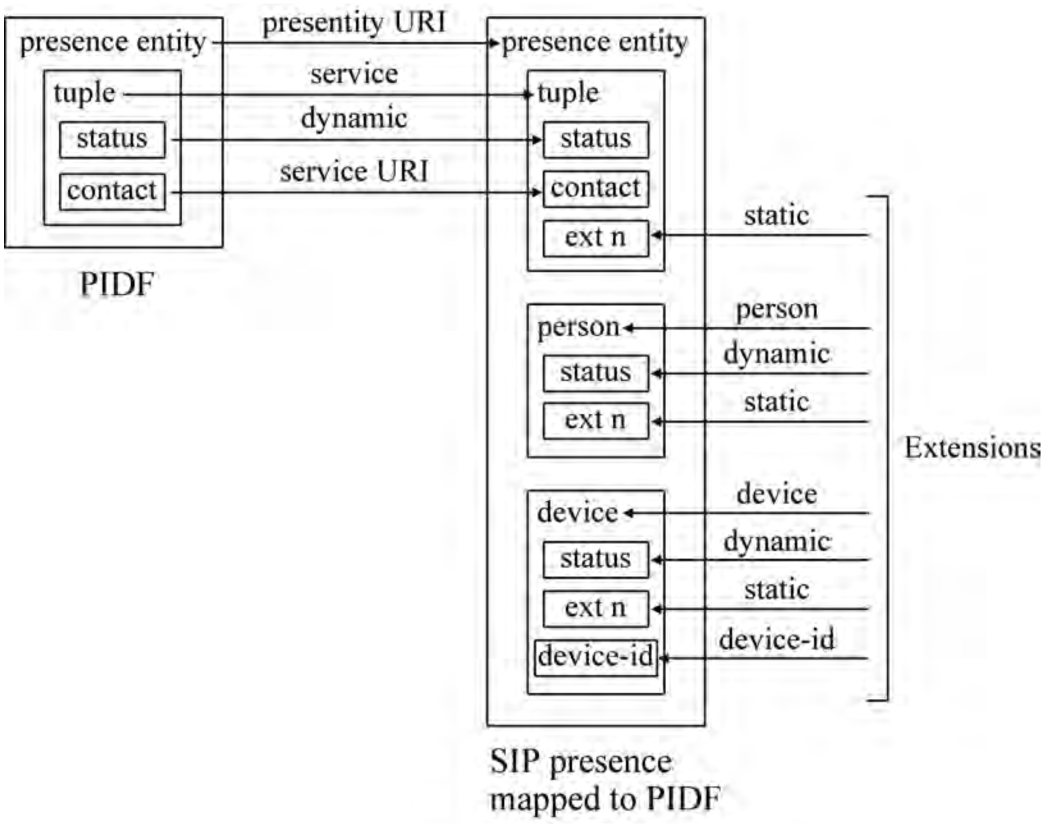


Figure 10: The SIP presence data model mapped to the PIDF (source: [22], p. 302)

- the possibility to set presence information automatically (e.g., from a calendar or user activity)
- backward compatibility to the PIDF (the content-type *application/pidf+xml* is also used for the RPID).

One example of a new element is the `<activities>` element, which gives information about the activities a Presentity is currently doing. The specification defines a list of possible activities (e.g., on the phone, dinner, holiday, travel) and allows a user-defined string enclosed in an `<other>` element. The list of possible values can also be extended with Internet Assigned Numbers

Authority (IANA)-registered values from other namespaces.
Another example is `<status-icon>`, which includes a URI pointing to an icon representing the current status of a person or service.
Some elements can have *from* and *to* attributes which reveal when the information is expected to be valid. These elements allow to give information about a future status.
Tab. 4 lists all elements defined by RFC 4480 [77].

Element	**Description**
`<activities>`	Activities a person is doing
`<class>`	Groups similar person elements, devices or services
`<deviceID>`	A device identifier
`<mood>`	The mood of a person
`<place-is>`	Properties of the place a person is currently at
`<place-type>`	Type of place the person is located in
`<privacy>`	Distinguishes whether the communication service is likely to be observable by other parties
`<relationship>`	Indicates how a person who is likely to be reached (e.g., an assistant) is related to the person who is associated with the Presentity
`<service-class>`	Describes the type of service in categories as electronic, post, etc.
`<sphere>`	Characterizes the overall current role of the Presentity (e.g., home)
`<status-icon>`	Depicts the current status
`<time-offset>`	Quantifies the timezone the person is in
`<user-input>`	Based on human user input, this element records the usage state of a device or service

Table 4: Elements of the RPID

2.4.6 Other Data Formats

Besides the RPID some other extensions to the PIDF exist. This section will shortly summarize extensions for contact information, location information and user agent capabilities since either the relevant 3GPP or OMA specifications mention these extensions (see Section 2.4.1).

RFC 4482 [75] specifies **contact information for the PIDF** (CIPID). The elements standardized by this data format describe contact information of a Presentity. Some of this information is usually found on business cards and includes references to the homepage, display name, an icon, a map and a representative sound, for instance. In contrast to the RPID, the CIPID focuses on the static part of a Presentity's presence information.

The presence-based **GEOPRIV Location Object Format** (RFC 4119 [55]) defines elements which describe a geographical position in the world. The XML schema extends the `<status>` element of the PIDF with a complex type named `<geopriv>`. This type has two mandatory child elements: `<location-info>` for location information and `<usage-rules>` for usage rules. Usage rules limit the retransmission and retention of location information. Optional child elements indicate the method by which a location was determined and describe the entity that supplied the location information.

The draft draft-ietf-simple-prescaps-ext [34] defines elements which describe **SIP user agent capabilities**. It specifies service capabilities as well as device capabilities. Service capabilities are used to describe characteristics, like which media type(s) a service support(s) (e.g., audio, data, text) or if the service represents an automata or a human. A device capability is, for instance, whether the device is fixed or mobile.

2.4.7 Resolving Ambiguity

The data model for Presence [66] takes ambiguity explicitly into account by allowing multiple descriptions of a single person, single service, or single device.

In some cases a compositor at the Presence Server can resolve this ambiguity in an automated way (see Section 5.1.2). But in many cases, as RFC 4479 [66] claims, the Watcher of the presence information can solve ambiguity best. A Watcher can have more information than the compositor or can present the ambiguity to the human user, who can resolve ambiguity often better than an automaton.
In general it is important that some attributes exist which help to resolve ambiguity. For an automaton any element can be useful; for a human user the `<note>` element can give useful hints in many cases. Often very helpful for both is a timestamp to determine the time of the most recent change.

3 Resource List Server Design

In this section we propose a generic design of a Resource List Server. The term Resource List Server (RLS) and its functionality is defined in RFC 4662 [62], which is an extension to the SIP-specific event notification framework (RFC 3265 [61]). The extension allows subscribing to a homogeneous list of resources instead of to a single resource. This helps to substantially reduce the network traffic. A Watcher need not handle subscriptions to many single resources but only a single subscription to a resource list. To save further traffic, a RLS can easily throttle the rate of notifications and adapt it to the capacity of the channel and/or terminal.
First, Section 3.1 gives an overview of the functionalities of a RLS and Section 3.2 shows an exemplary message flow. Section 3.3 defines the main components of the proposed design and Section 3.4 the interfaces between them. Section 3.5 presents the different components more in detail. We conclude with some design decisions in Section 3.6. The formats of presence and URI Lists have been discussed already in Section 2.3.5.

3.1 Overview of RLS Operation

This section gives a comprehensive overview of the functionality of a Resource List Server (RLS). Fig. 11 shows the involved network entities and the interactions between these entities. Note that the interactions do not directly map to any standardized messages.
The messages exchanged between the Watcher and the Resource List Server follow the SIP SUBSCRIBE/NOTIFY mechanism as defined in [61]. The Watcher acts as subscriber and the Resource List Server plays the role of a notifier.
There is no difference between a subscription to a single resource or to a list. If a client supports the event list extension, the target of a SUBSCRIBE request can either be a single resource or a list. The support is indicated

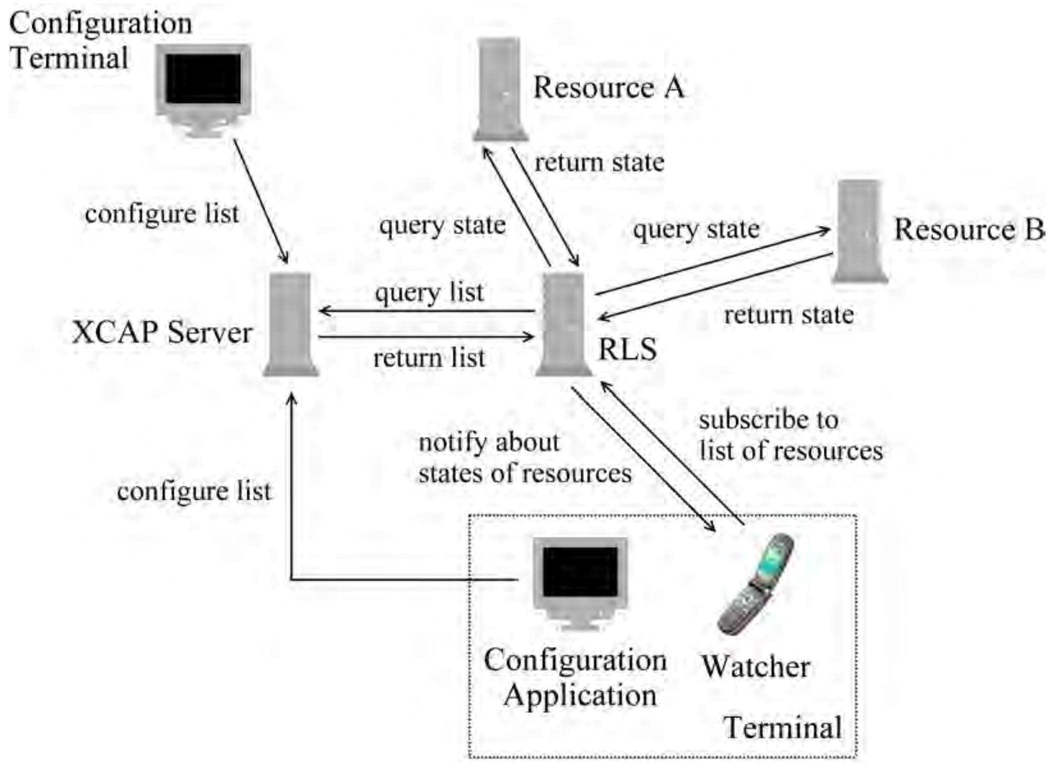


Figure 11: RLS network diagram

by including an option tag *eventlist* in a Supported header field of every SUBSCRIBE message. If the subscription is to a list, the RLS indicates this by including the tag *eventlist* in a Require header field of the response to a SUBSCRIBE message and in all following NOTIFY messages.
The returned NOTIFY body is of MIME type multipart/related. The first section of the multipart/related document is always of type application/rlmi+xml and contains meta-information about each resource in the list. The other sections provide information about the state of individual resources. Listing 18 in Section 5.5.1 shows such a multipart/related document including a RLMI document.
The root element of a RLMI document is `<list>` which contains, among other information, the URI identifying the list. A `<list>` can have several

`<resource>` child elements, which describe a single list entry. A `<resource>` element must have an *uri* attribute, which specifies the URI identifying this resource. Finally, a `<resource>` element contains one or several `<instance>` elements which inform about the state of a virtual subscription[6] (e.g., active) and link to a presence document which is included in the multipart/related body. The format of the individual state information is determined by the event package of the initial SUBSCRIBE request to the RLS (e.g., presence).
When the RLS receives a SUBSCRIBE request, it must obtain the rls-service definition (i.e., Presence List in terms of the OMA Presence SIMPLE enabler) which corresponds to the Request-URI in the SUBSCRIBE message. As described in Section 2.3.5, the service definition either directly includes a resource list (i.e., URI List in terms of the OMA XDM enabler) or links to such a list. In the latter case, the RLS follows the link to obtain the resource list. Generally spoken, a XCAP server stores the rls-service and resource-list definitions and the RLS queries them over the XCAP protocol. In context of the OMA Presence SIMPLE enabler, a RLS XDMS stores the Presence Lists and a Shared XDMS the URI Lists.
In the end, the RLS has a list of URIs which represents resources the RLS should learn the state of. How the RLS learns the state of a resource is up to the RLS. The RLS can be the authority of the state and access the state directly. Alternatively, the RLS can subscribe to another entity and learn resource states from this entity. In Fig. 11, Resource A and Resource B are such entities (e.g., Presence Server) which manage the state of resource A and resource B. Any subscription that a RLS creates to learn the state of a resource the RLS is not the authority of is called back-end subscription. Back-end subscriptions can be a SIP subscription or of any other type.
The resource lists stored on a XCAP server (i.e., a Shared XDMS in terms of the OMA XDM enabler) can be created and maintained by sending suit-

[6] "A Virtual Subscription is a logical construct within an RLS that represents subscriptions to the resources in a resource list." ([62], p.3)

able XCAP requests to the server. The configuration entity (i.e., a XDM client in terms of the OMA XDM enabler) which allows a user to manage her/his resource lists can be integrated in the same terminal as the Watcher or be an independent application on a separate terminal. For instance, the configuration can be done over a website.

3.2 Example Message Flow

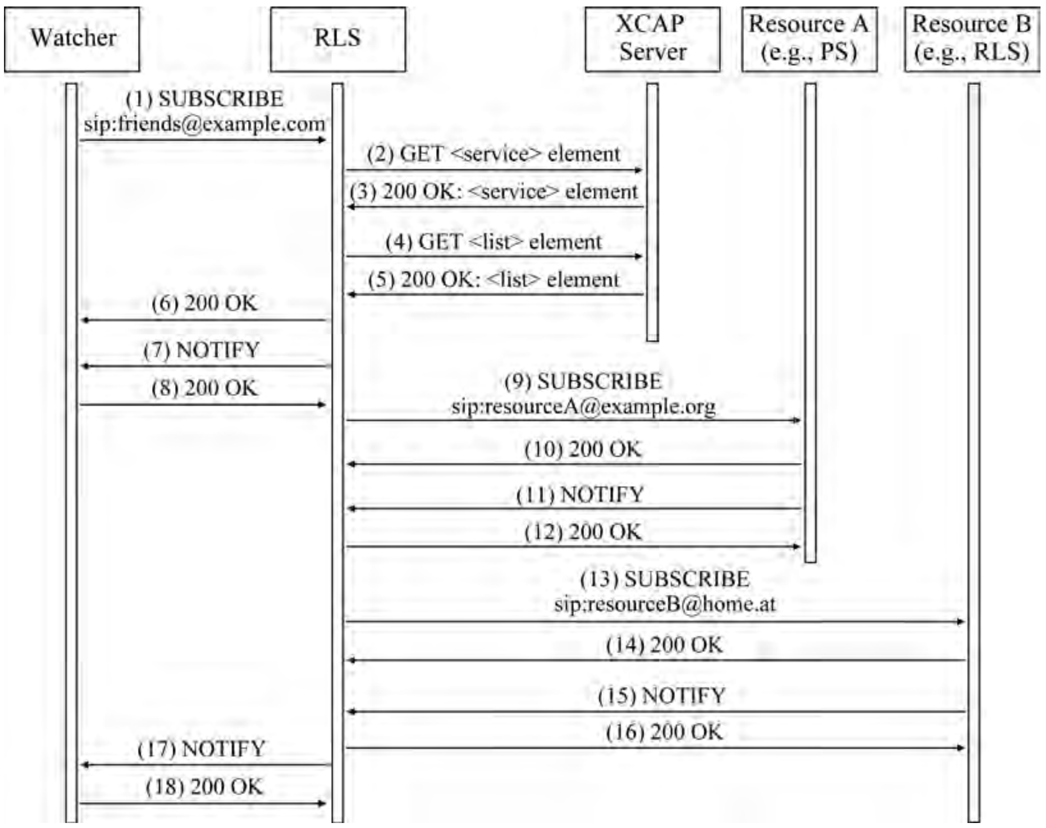
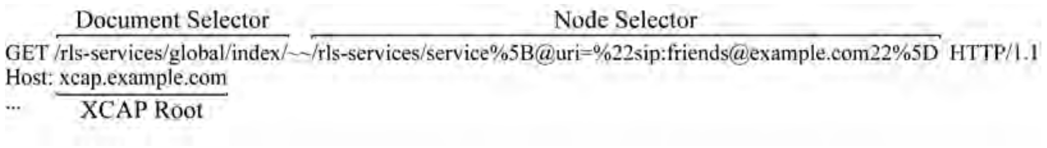


Figure 12: Exemplary RLS message flow

Fig. 12 illustrates an exemplary message flow. In this example, the back-end subscriptions, which are sent by the RLS to learn the state of a resource, are SIP subscriptions. If we assume *presence* as event package, the state of

resource A could be controlled by a Presence Server. Resource B could again be a list whose state is controlled by another RLS. Security mechanisms are intentionally omitted in Figure 12 for simplicity reasons.
Let us assume the Watcher wants to subscribe to *sip:friends@example.com*. Therefore, the Watcher sends a SUBSCRIBE message (1), which terminates at the RLS. The RLS acknowledges the message with a 200 OK (6). RFC 3265 [61] requires an immediate NOTIFY message. This NOTIFY message contains a RLMI document describing the complete buddy list. Therefore, the RLS must first fetch the service definition from a XCAP server.



```
GET /rls-services/global/index/~~/rls-services/service%5B@uri=%22sip:friends@example.com22%5D HTTP/1.1
Host: xcap.example.com
...
```

Document Selector
Node Selector
XCAP Root

Figure 13: XCAP `<service>` request example

```
HTTP/1.1 200 OK
Etag: "xyzz"
Content-Type: application/xcap-el+xml

<service uri="sip:friends@example.com">
 <resource-list>http://xcap.example.com/resource-lists/users/bob/index/~~/
     resource-lists/list%5b@name=%22acd2r%22%5d</resource-list>
  <packages>
    <package>presence</package>
  </packages>
</service>
```

Listing 5: XCAP `<service>` response example

The RLS sends an XCAP GET request (2) to the XCAP server and receives as reply a `<service>` element (3). Fig. 13 shows the XCAP request for the `<service>` element associated with the URI *sip:friends@example.com* stored in the file *index* on the host *xcap.example.com*. The request consists of a document selector, which selects the XML document, and a node selector, which selects an element or attribute within this document. In this case, the node selector references the `<service>` element with the mandatory *uri* attribute set to *sip:friends@example.com*. This URI identifies the service. The response to the request is shown in Listing 5. The `<service>` element defines the event packages which are allowed for this service and includes a resource list or a link to a resource list. In our scenario, the latter case is true which causes the RLS to fetch the list definition in a second query (4). A request to the URI specified in the service definition and shown in Listing 5 will return the list definition starting with the opening bracket of `<list>` until the closing bracket of `</list>`. Listing 6 gives an example of a list definition (also compare Section 2.3.5).

```
<?xml version="1.0" encoding="UTF-8"?>
<resource-lists xmlns="urn:ietf:params:xml:ns:resource-lists"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

 <list name="acdr2">
   <entry uri="sip:bill@example.com">
     <display-name xml:lang="en">Bill Doe</display-name>
   </entry>
   <entry uri="sip:bob@example.com">
     <display-name>Bob Buse</display-name>
   </entry>
   <list name="close-friends">
     <display-name>Close Friends</display-name>
     <entry uri="sip:joe@example.com">
       <display-name>Joe Smith</display-name>
     </entry>
     <entry uri="sip:nancy@example.com">
       <display-name>Nancy Lu</display-name>
     </entry>
   </list>
 </list>
</resource-lists>
```

Listing 6: Resource list example

Now the RLS has all necessary data in order to send the required immediate NOTIFY message (7). In this NOTIFY request the RLS can also include the state information it has knowledge about already (i.e., in case the RLS

is the authority for the state). In order to obtain the state of the remaining resources, the RLS sends SUBSCRIBE messages to Resource A (9) and Resource B (13). The incoming NOTIFY message (11) contains the state of resource A. The RLS could now send this new information to the Watcher but it decides to buffer this information and reduce the rate of notifications. Next the RLS receives a state information of resource B in another NOTIFY message (15).
The RLS decides that there is sufficient information available for justifying a subscriber notification. It sends a NOTIFY message with the updated state informations to the Watcher (17). It is left to the implementor of a RLS to define a policy when a list subscriber is informed about state changes.

3.3 Main Components

After a general overview of the RLS functionality, we now introduce our RLS design. In the following we discuss external entities which interact with the RLS, the main components of the RLS design and the interfaces between them.
Fig. 14 depicts the external entities the RLS interacts with and which are explained in the following list:

- A **Configuration** is a file which contains all configuration parameters.
- A **Watcher** subscribes to a resource list.
- An **Authority of local resources** is an application which manages the statuses of Presentities and runs on the same physical node as the RLS (e.g., a Presence Server).
- A **Notifer** is any entity which is able to inform the RLS about the status of a Presentity using a SUBSCRIBE/NOTIFY dialog (e.g., a Presence Server).

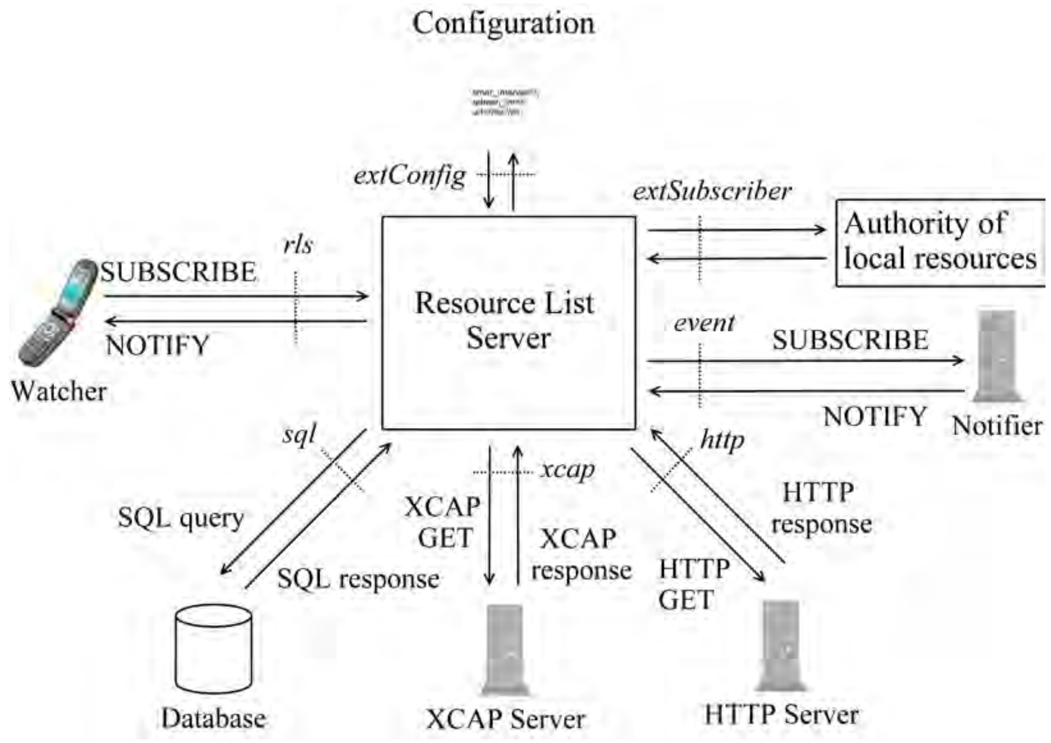


Figure 14: External Entities

- **Database**, **XCAP server** and **HTTP server** store RLS-service definitions and resource lists.

Fig. 15 shows the main components of the design and the interfaces between them, which we discuss in the following.

Notifier: This component acts as a notifier as defined in [61]. It handles incoming SUBSCRIBE messages and generates NOTIFY messages. It interacts with the Resource List Fetcher component to obtain a list of URIs the RLS should subscribe to. The task of determining the state of a single resource is delegated to the Virtual Subscription Handler component.

We assume that the subscriber has been authenticated at some point before the request is forwarded to the RLS. If this is not the case, the

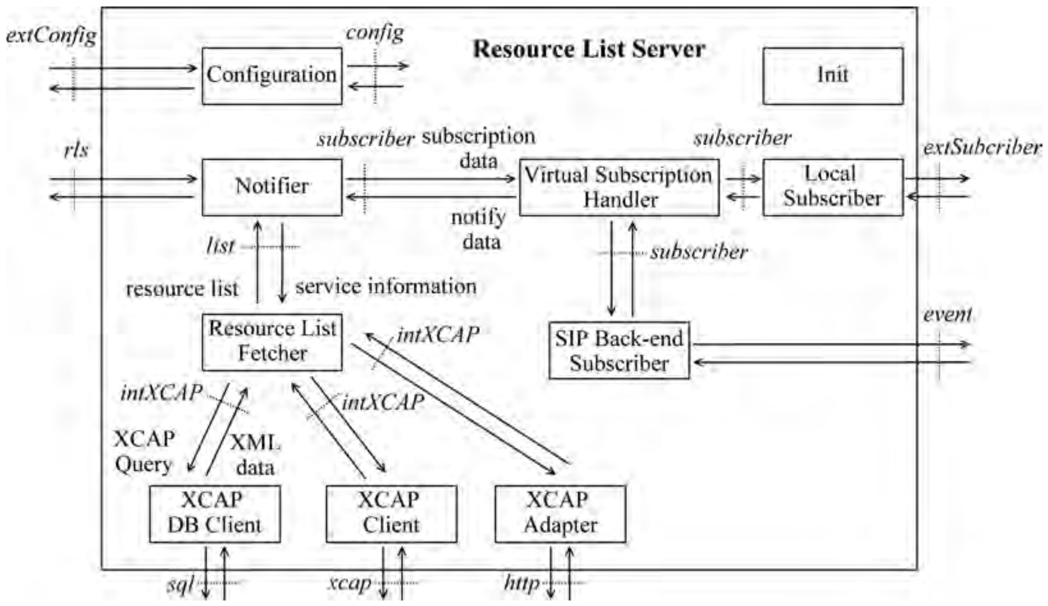


Figure 15: RLS block diagram

Notifier component's responsibility is also subscriber authentication. A subscriber authentication is mandatory because the SIP Back-end Subscriber component (see Section 3.5.4) relies on it. The Notifier component is also responsible to authorize the subscriber if she/he is allowed to subscribe to the desired list.

Resource List Fetcher: This component is responsible for fetching the appropriate resource list from a XCAP source. Depending on the subscriber and the R-URI in the SUBSCRIBE request, this component retrieves the appropriate service definition. The service definition is part of a rls-service document and either contains directly a resource list or links to a resource list in a resource-list document. In the latter case, this component also fetches the resource list. Both document types are defined in [65] (compare Section 2.3.5). The elements can be fetched from different sources. In this design the components XCAP Client, XCAP Adapter and XCAP DB Client are responsible for fetching elements. Since all components offer the same interface the location is transparent for the Resource List Fetcher. The component transforms a resource-list definition to a list of URIs the RLS can subscribe to and returns this list to the Notifier component.

XCAP Client: This component fetches required XML documents from a XCAP server. The functionality of a XCAP client is defined in [72]. It is sufficient that the client supports the GET operation.

XCAP Adapter: Instead of a fully-fledged XCAP server, a common HTTP server serves as source of XML documents. The XCAP Adapter fetches complete XML documents from an HTTP server and extracts the necessary elements locally according to the passed XCAP query. It is sufficient that the component only understands the queries needed by the Resource List Fetcher.

XCAP DB Client: This component offers the same functionality as the XCAP Adapter component but queries the XML documents from a database.

Virtual Subscription Handler: This component decides how the RLS learns the state of a certain single resource by dispatching the subscription information to the right component. This flexible design enables two options. If the RLS is collocated with a program which is the authority for the resource, the Local Subscriber component interacts with this program and handles the subscription. Otherwise the task is passed to the SIP Back-end Subscriber component.

Local Subscriber: The Local Subscriber learns the state of a resource from another external program or software module.

SIP Back-end Subscriber: This component learns the state of a resource due to a SIP subscription. It plays the role of a subscriber as defined in [61].

Configuration: The Configuration component encapsulates all configurable parameters and makes them available via the config interface.

Init: This component does all necessary initializations on startup and cleans up on shutdown of the RLS.

3.4 Interfaces

This section describes the external and internal interfaces of the RLS module. A dashed line together with a name in italic marks the interfaces in Fig. 15.

3.4.1 External Interfaces

rls: This interface allows a user agent to subscribe to a resource list and obtain presence information. It is defined in RFC 4662 [62].

sql: This interface allows querying/saving information from/to a database.

xcap: This interface allows querying information from an XCAP server. It is defined in the XCAP specification [72] whereas the RLS acts as XCAP client. Only a subset of the XCAP interface must be implemented for a RLS.

http: This common interface is defined in RFC 2616 [28] and allows fetching information from a web server.

extSubscriber: This interface allows subscribing to state information provided by an other program/software module. The interface can be equal to the internal interface subscriber (see Section 3.4.2) but need not be.

event: This interface allows subscribing to the presence information of a particular resource. It corresponds to the SIP Event Notification defined in RFC 3265 [61] plus an event package (e.g., the presence event package [63]).

extConfig: This interface provides the values of all RLS configuration parameters.

3.4.2 Internal Interfaces

list: The list interface allows querying a list of resources. The R-URI of a SUBSCRIBE request, which identifies the RLS service, together with the identity of the subscriber and the event package of a subscription to a resource list are passed to the Resource List Fetcher. The Resource List Fetcher returns a hierarchical list of resources. A list can consist of several sub-lists and each list consists of zero or more resources. Zero or more names can be assigned to both a list and a resource.

intXCAP: A XCAP query is passed to one of the XCAP handling components and the queried xml data is returned.

subscriber: This interface allows establishing and canceling subscriptions to a single resource. The following information must be passed over the interface in order to establish a subscription:

- URI which identifies the resource
- identity of the list subscriber
- event-type
- accepted MIME-types of the list subscription.

The subscription handling component is not allowed to reuse a subscription to a certain resource for several users because this would circumvent authorization policies. A back-end subscription is always done in the name of the list subscriber.

The subscription handling component informs the calling component about changes of the resource state (e.g., a received NOTIFY) it has successfully subscribed to. It gives the calling component access to the following information:

- status of the subscription
- reason (if state is *terminated*)
- MIME content-type
- NOTIFY body (i.e., resource state).

config: This interface exposes all configuration parameters to the other components. Since almost all components make use of parameter settings, the information flow to every single component is not shown in the Figures.

3.5 Component Descriptions

While the previous section has presented an overview of the RLS design, this section illustrates design details specific to some of the components.

3.5.1 Notifier

As shown in Fig. 16, the Notifier is divided in the following sub-blocks:

Subscription Handler: The handler accepts incoming SUBSCRIBE messages. It retrieves a list of resources from the Resource List Fetcher component and initiates a virtual subscription for each resource in the list.

Notify Generator: This component is responsible for creating and sending NOTIFY messages. It creates the RLMI document and merges it together with the resource state information into a single message. This component controls the rate of notifications.

Subscriber Authorization: This component determines if the subscriber is authorized to subscribe to the desired list.

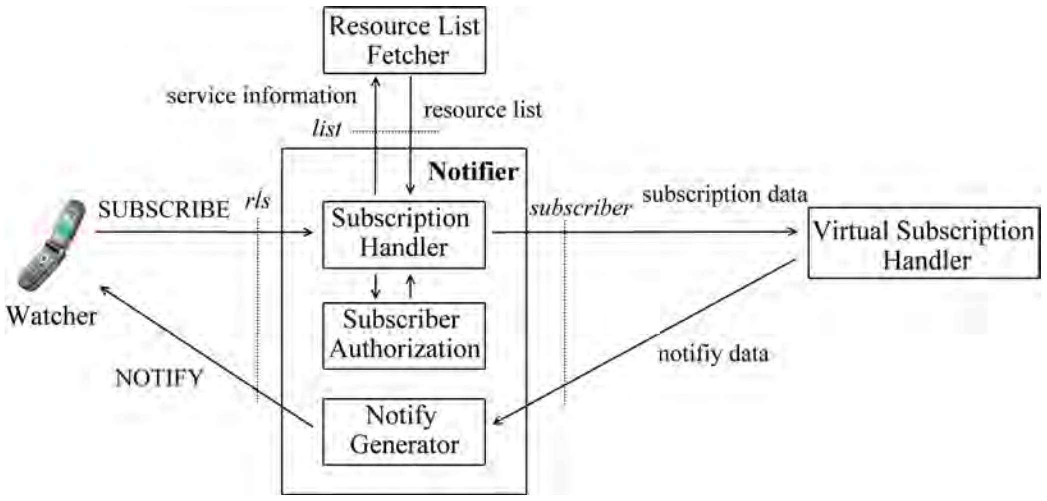


Figure 16: Notifier component

3.5.2 Resource List Fetcher

As shown in Fig. 17, the Resource List Fetcher is subdivided into the following components:

Service Fetcher: The Service Fetcher fetches a rls-service definition [62] from one of the possible sources. It chooses the XCAP Client component if the documents must be fetched from a XCAP server. The XCAP Adapter component is used if the documents are saved on an HTTP server and the XCAP DB Client component if they are saved in a database. The source is determined by a module parameter. The list definition is either part of the service definition or stored in a separate document. In the latter case, this component also fetches the list definition.

List Creator: The List Creator reads the XML list definition from the Service Fetcher and creates a (hierarchical) list of URIs plus some additional information (i.e., names) which is necessary for the later processing. The Notifier component uses this list later on to create virtual

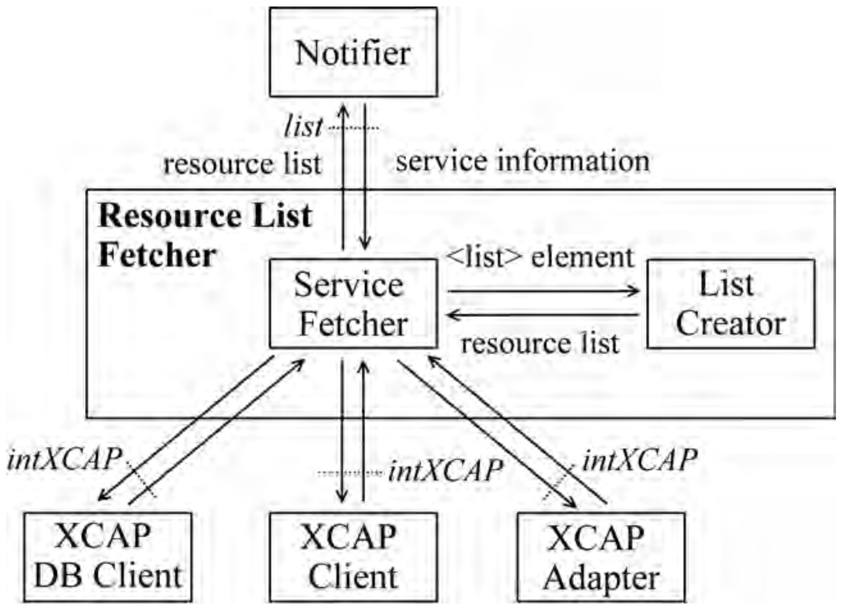


Figure 17: Resource List Fetcher component

subscriptions and a RLMI document.

3.5.3 Virtual Subscription Handler

The sole task of the Virtual Subscription Handler is to decide on base of subscription parameters for a specific resource which component takes care of reporting the resource state. This could be an external program/module if this module is the authority for the state of this resource, the SIP Back-end Subscriber component for SIP subscriptions or another component which maybe will be defined in the future.
In a first release, every resource state will be obtained by SIP back-end subscriptions. This means that even for locally known states a new SIP subscription is initiated.
In later releases, the decision procedure could be the following: If the domain part of the URI of a resource is equal to the local domain, the resource state is obtained by another local module. Otherwise the SIP Back-end

Subscriber component initiates a back-end subscription. A module parameter determines if every resources should lead to a back-end subscription or not.

3.5.4 SIP Back-end Subscriber

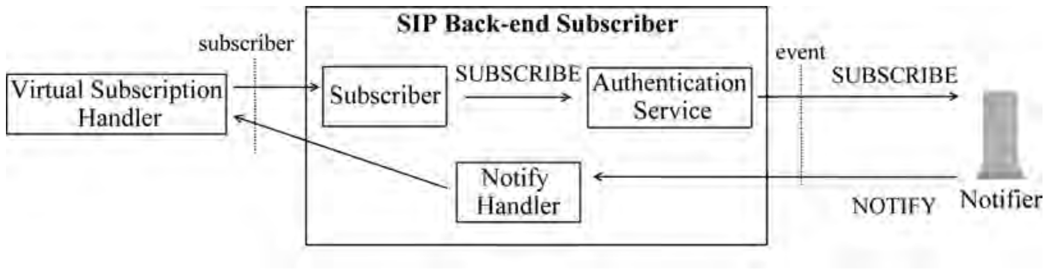


Figure 18: SIP Back-End Subscriber component

Fig. 18 shows the sub-blocks of the SIP Back-End Subscriber component.

Subscriber: The Subscriber component implements the subscriber interface. It generates and sends SUBSCRIBE messages and is in charge of refreshing existing subscriptions.

Authentication Service: The Authentication Service calculates a hash value over some header fields, signs this hash value and includes the signature in the header of a back-end subscription. Thereby the Authentication Service ensures other servers that the user in the From header is authenticated. This requires that the user is first authenticated in the home domain.

The Authentication Service enables the RLS to receive the same presence information as if the list subscriber itself subscribes to the resource. RFC 4474 [56] defines the role of an authentication service.

Notify Handler: The Notify Handler receives incoming NOTIFY messages as response to the SUBSCRIBE messages sent by the Subscriber component. It stores the state information in the database and informs other components about the state change.

3.6 Design Decisions

This section summarizes some design decisions concerning rate of notifications, full/partial state and subscriber authorization.

3.6.1 Rate of Notifications

One of the purposes of a RLS is to buffer notifications and reduce the traffic to the Watcher. The RLS can be configured to send at maximum each n^{th} second a NOTIFY message to the Watcher. If this value is set to 0, every state change results immediately in a new NOTIFY message.

3.6.2 Full/Partial State

RFC 4662 [62] allows NOTIFYs which contain either full or partial state. If partial state is indicated, only state information about resources with an updated state are included. The type of state (full/partial) does not refer to any underlying event package, which potentially allows some kind of partial state information for a single resource.
The RLS module can be configured to either only send NOTIFY messages with full state or allow NOTIFY messages with partial state. In the latter case, a full state is only sent when this is forced by RFC 4662 [62] (i.e, for immediate NOTIFY requests upon the reception of a SUBSCRIBE request).

3.6.3 Subscriber Authorization

RFC 4826 [70] suggests the presence authorization rules defined in [71] (see Section 2.3.5) for a subscriber authorization. But this specification defines several conditions and transformations which are not useful in case of a resource list. Generally, the RLS cannot rely on reading the state information of resources because the bodies of incoming NOTIFY requests may be encrypted. The RLS only knows the Resource List Meta Information for sure.

Basically, the information worth to protect is the membership of a resource in a list and not the state of resources. A client can always learn the state of a single resource due a subscription to this single resource instead of a subscription to the list. The implemented prototype only allows a subscription to a certain resource list if the resource list is stored in the user tree of the subscriber on the XCAP/HTTP server or stored in the database for this particular subscriber (see Section 4.4.1).

4 Resource List Server Implementation

Based on the RLS design presented in Section 3, we have implemented a RLS prototype which integrates with the modular concept of the OpenSER SIP proxy [5]. Therefore, we give an overview of OpenSER in Section 4.1. Section 4.2 establishes a connection between the design of Section 3 and source-code files. Subsequently, Section 4.3 identifies the dependencies of the implemented module on functionalities of the OpenSER core, other OpenSER modules and external libraries. We conclude with selected implementation aspects in Section 4.4. Appendix A summarizes all module parameters, exported module functions and database table definitions.

4.1 OpenSER

OpenSER [5] is a flexible open-source SIP proxy, which, depending on configuration, can act as SIP Proxy, SIP Registrar or SIP Application Server. The code is written in pure C for Unix/Linux-like systems in order to support a wide range of (embedded) architectures.
OpenSER includes its own scripting language for configuration files. Among others, this scripting language allows loading and configuring modules, manipulating SIP messages and determining the Proxy's routing logic.
OpenSER is deployed as part of commercial systems (e.g., Cisco Service Node servers [3]) and included in many official Unix-based distributions (e.g., Ubuntu).
The OpenSER project is a spin-off of the SIP Express Router (SER) project. SER has been mainly developed by the FhG FOKUS research institute in Berlin and had its first release in autumn 2002. Due different views in the management and development of the SER project, two core developers started the OpenSER project in June 2005. The latest major release was OpenSER v1.3.0 in December 2007.

4.1.1 OpenSER Modules

One of the main features of OpenSER is a plug&play module interface which is similar to the one implemented by the Apache Web Server [1]. It allows keeping the core highly stable and of small size. Functionality is added by modules without a need to change the core. This is a powerful tool to deploy flexible and custom SIP services.
Each OpenSER module defines a structure which exports the following information:

- Module name
- Initialization, per-child initialization (called by all processes after the fork) and destroy module functions
- Exported functions which can be called from the configuration script
- Module parameters
- Exported statistics
- Exported management interface (MI) functions
- Exported pseudo-variables
- Additional processes required by the module
- `dlopen()` flags (`dlopen()` loads the dynamic library file)
- Function which handles response messages

The Resource List Server is implemented as an OpenSER module. The rls module exports module parameters and functions which are accessible from the configuration script but no other functions or pseudo-variables.

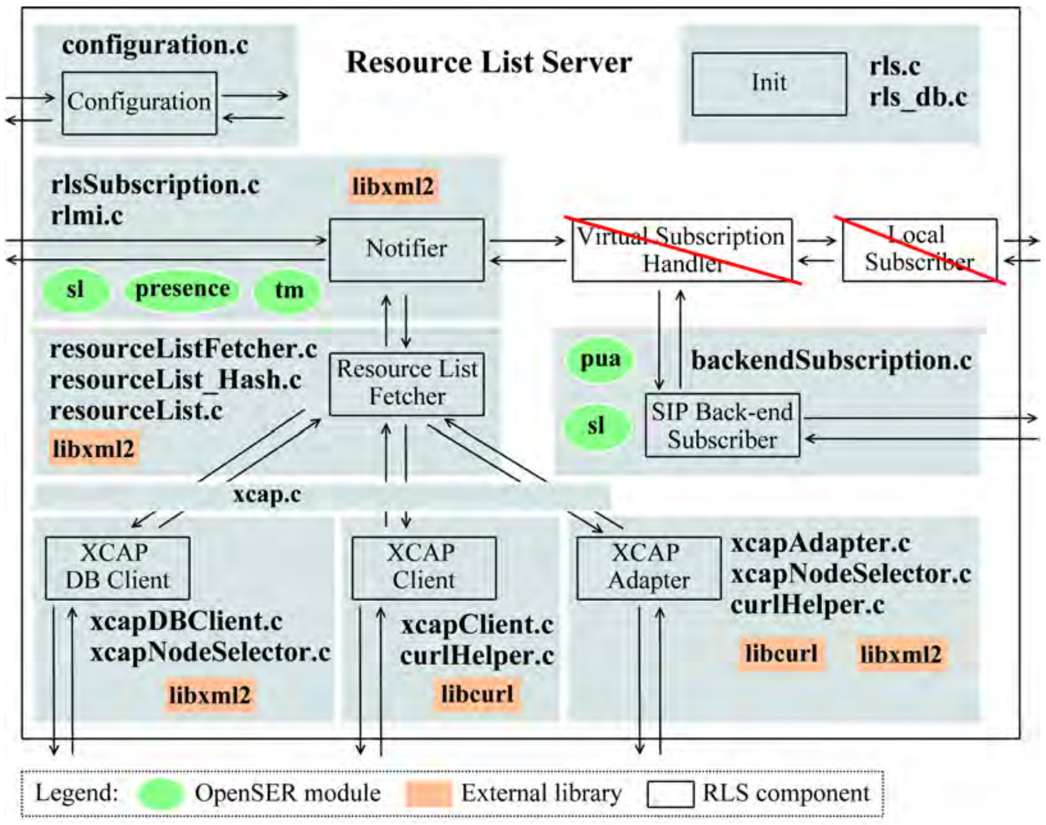


Figure 19: Source-code files and dependencies on OpenSER modules and external libraries

4.2 Mapping the Design to Source Code

Fig. 19 shows which source-code files (in bold letters) implement which parts of the design described in Section 3.

The RLS prototype does not implement the components Virtual Subscription Handler and Local Subscriber. Even if the Presence Server and RLS are collocated and run within a single OpenSER instance, the RLS sends a back-end subscription for every resource. It has the advantage that the Presence Server and RLS can be easily split into two independent servers and the load distributed at anytime. On the other hand, this solution is slower and

increases the network traffic. However, a Virtual Subscription Handler can be added at any time. Additional coding of the Local Subscriber component is needed to access presence information locally without sending a SUBSCRIBE request. Section 4.5 discusses some potential problems in doing so.
Table 5 briefly presents the individual source-code files.

4.3 Dependencies

The prerequisites for the rls module are OpenSER itself, along with the modules listed in 4.3.1, the libraries listed in 4.3.2 and a database of a type OpenSER supports.

4.3.1 OpenSER Core and Modules

Since the RLS is implemented as an OpenSER module, it highly makes use of functionalities offered by the OpenSER core and other OpenSER modules. The rls module uses the following OpenSER core APIs:

Module interface: It allows setting the module parameters in the configuration file and calling module functions upon the reception of SIP messages.

Memory management: OpenSER has its own memory management to allocate and free private and shared memory.

Database interface: OpenSER offers a generic interface to query and store data from/to a database. In version 1.3.0, OpenSER supports MySQL, PostgreSQL, Berkeley, flat files, all database types which have unixodbc drivers, and a Perl virtual database.

Message parsing: OpenSER includes its own message parser designed for speed.

Timer: Timers allow the execution of periodical tasks.

Source file	**Description**
rls.c	Implements OpenSER's module interface
configuration.c	Repository of all module parameters
rls_db.c	Encapsulates the initialization and termination of a database connection; the corresponding header file defines all column names of the used database tables
rlsSubscription.c	Handles incoming subscriptions to a resource list and sends appropriate responses and NOTIFY requests
rlmi.c	Creates multipart/related documents composed of RLMI document(s) and the presence information of individual resources
backendSubscription.c	Starts and stops back-end subscriptions with the help of the pua module, handles corresponding incoming NOTIFY requests and stores dialog information to the database
resourceListFetcher.c	Fetches service-definitions/resource-lists from the desired source and parses the necessary information into an internal structure
resourceList.c	Offers methods to create resource-list structures as used internally, add different attributes and copy complete lists
resourceList_Hash.c	Hash table which stores resource lists
xcap.c	Offers a common interface for all XCAP sources and binds functions depending on the used source
xcapClient.c	Fetches (partial) XML documents from a XCAP server with the help of curlHelper.c
xcapAdapter.c	Fetches complete XML documents from an HTTP server with the help of curlHelper.c and processes the node selector locally with the help of xcapNodeSelector.c
xcapDBClient.c	Fetches complete XML documents from a database and processes the node selector locally with the help of xcapNodeSelector.c
xcapNodeSelector.c	Processes a XCAP node selector and returns the desired part of a XML document
curlHelper.c	Uses the libcurl library to fetch a document from a XCAP or HTTP server

Table 5: Source-code files

Locking: OpenSER includes its own locking library.

Logging: OpenSER has logging facilities for debug and error messages.

Helper functions: Some useful definitions and helper functions are used from the core (e.g., string definition, hash-code generation).

Besides some functionalities offered by the core, the rls module depends on the OpenSER modules presence, pua, sl, tm and one database module. Fig. 19 shows these modules as green ellipses. Only the database module is not shown because it depends on the used database type.

presence: The presence module evolved from, as the name states, presence handling module, into a more generic module which handles PUBLISH and SUBSCRIBE messages and generates NOTIFY messages in a general, event independent way. However, it is not possible with the provided API to handle incoming SUBSCRIBE messages and send NOTIFY messages. The rls module uses the presence module to save/delete resource-list subscriptions to/from a hash table and save/restore this hash table to/from the database (rlsSubscription.c). Moreover, the API allows extracting dialog information from message headers and copying stored dialog information. Last but not least, the module manages a list of the event types which the RLS supports.

pua (presence user agent): The pua module offers the functionality of a presence user agent (i.e., Presence Source) client and enables other modules to send SUBSCRIBE and PUBLISH messages. The rls module uses the pua module to send back-end subscriptions (backendSubscription.c).

sl (stateless replier): The sl module allows generating SIP replies to SIP requests without keeping a state. The rls module makes use of this functionality when sending replies to incoming SUBSCRIBE messages (rlsSubscription.c) or incoming NOTIFY messages (backendSubscription.c).

tm (Transaction (stateful) module): The tm module enables stateful processing of SIP transactions and allows processing transaction state as opposed to individual messages. This is needed when the rls module sends NOTIFY messages within an established dialog (rlsSubscription.c).

database module: The generic database interface is part of the core but implemented by modules. A separate module exists for each supported database type. For instance, the module for MySQL connectivity is named mysql.

4.3.2 External Libraries

In Fig. 19, the pastel-colored rectangles indicate external libraries. The rls module depends on the libraries libxml2 and libcurl.

libxml2: Libxml2 [7] is a XML C parser and toolkit, which is known to be very portable. We use libxml2 for:

- parsing XML documents when processing a XCAP node selector (xcapNodeSelector.c)
- parsing service definitions and resource lists (resourceListFetcher.c)
- creating RLMI documents (rlmi.c).

libcurl: Libcurl [4] is a free and easy-to-use client-side URL transfer library, supporting a various number of protocols and is highly portable. We use libcurl for:

- fetching documents from an HTTP server (curlHelper.c)
- fetching documents from a XCAP server (curlHelper.c).

4.4 Implementation Aspects

In this section we highlight some implementation aspects. We discuss subscriber authentication and authorization, how the prototype caches data and stores them in a database. We explain how the prototype fetches rls-service definitions, how these definitions relate to a RLMI document and how the prototype handles back-end subscriptions.

4.4.1 Authentication and Authorization

The rls module itself does not authenticate subscribers. In the OpenSER design other modules handle authentication before the routing logic calls the appropriate rls module function. The auth module provides common functions which are needed by other more advanced authentication related modules, namely auth_db (database-backend authentication module), auth_diameter (Diameter-backend authentication module) and auth_radius (Remote Authentication Dial In User Service (RADIUS)-backend authentication module). However, if the RLS is an Application Server in an IMS network, authentication is anyway the task of the S-CSCF and not of the RLS.
Authorization is done insofar that the rls module only fetches resource lists which are saved under the identity of the resource-list subscriber in the database or in the XCAP user tree on an HTTP/XCAP server. Therefore, users can only subscribe to their own resource lists.
Furthermore, the RLS supports Simple and Digest HTTP authentication when fetching documents from an HTTP/XCAP server. Section 4.5 suggests some improvements on this issue for a next release.

4.4.2 Database Tables and Caching

The rls module utilizes two database tables. A third one is necessary if the module fetches the XML documents from a database.

- The table *rls_rl_subscriptions* stores the dialog information of resource-list subscriptions.
- The table *rls_backend_subscriptions* (partly) stores the dialog information of back-end subscriptions (some information is stored by the pua module).
- The table *xcap* stores rls-service and resource-list XML documents.

The *xcap* table need not be in the same database as the other two tables. A separate database can be specified via the *xcap_root* module parameter. The database table structures are defined in Appendix A.3.

The information about ongoing resource-list subscriptions is copied from the database table *rls_rl_subscriptions* into a hash-table on start-up and modified entries are regularly written back to the database. This caching is mainly handled by the presence module.

The Resource List Fetcher fetches the service and resource-list XML documents from a XML source and keeps the parsed resource lists in memory. After a configurable period of time (*resource_list_cache_time* module parameter), the resource list is deleted from memory and need to be fetched and parsed again (if still needed). This removes unneeded resource lists from memory but also allows noticing changes in resource lists. However, the terminal also needs to send a new SUBSCRIBE request to recognize these changes correctly. Section 4.5 suggests some improvements on this issue for a next release.

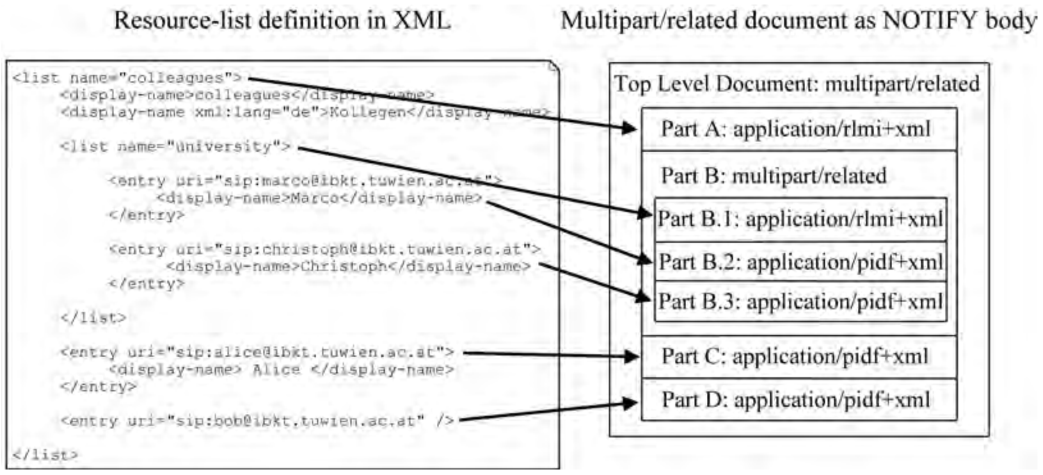
By contrast, the information about back-end subscriptions is not cached because the entries only change when a back-end subscription is started or stopped, a notification is received or an unsuccessful subscription is renewed. If the notification and retry intervals for unsuccessful subscriptions are not too small, the frequency of a database access is similar to the frequency the data in a cache would be saved to the database. In this case, a cache does not improve the performance. However, an optional cache is a possible feature of a next release.

4.4.3 Fetching of RLS-Service Definitions

In order to fetch a service definition, the Resource List Fetcher component requests the following URL from one of the XCAP components:
`/rls-services/users/[RL-SUBSCRIBER]/index/~~/rls-services/`
`service[@uri="[R-URI]"]`
`[RL-SUBSCRIBER]` is replaced by the SIP URL in the From header of the SUBSCRIBE request to the resource list. `[R-URI]` is replaced by the Request URL of the SUBSCRIBE request. The XCAP Adapter and XCAP Client components insert the configured XCAP root in the front to complete the URL.
The XCAP Client component sends the full URL to the XCAP server. The XCAP Adapter removes the node selector, which is separated from the document selector by two tilde characters, and fetches the file *index* stored at `/rls-services/users/[RL-SUBSCRIBER]/` from an HTTP server. The HTTP server is specified by the XCAP root. The node selector is processed locally.
The XCAP DB Client parses the node selector locally as well but fetches the rls-service document that is stored for the particular user `[RL-SUBSCRIBER]` in the database table *xcap* (see Appendix A.3 for the table definition).
If the service definition does not contain a list but a reference to a list, this reference is resolved by the Resource List Fetcher component. However, the reference must contain the same XCAP root as where the service definition was fetched from and point to the same user directory of `[RL-SUBSCRIBER]`.
Section 4.5 lists missing features of the prototype and suggests some improvements for a next release.

4.4.4 Modeling of Resource Lists

RFC 4826 [70] defines a XML format for representing resource lists (compare Section 2.3.5) and RFC 4662 [62] defines a XML format for RLMI. The RLMI is a part of the multipart/related document which the RLS sends to a subscriber in NOTIFY messages. It is the task of the Resource List Fetcher Component to extract the necessary information from the XML resource-list document and to cache it in an internal structure. The Notifier (more precisely rlmi.c) uses this data to create the RLMI document.


Resource-list definition in XML
Multipart/related document as NOTIFY body
Top Level Document: multipart/related
Part A: application/rlmi+xml
Part B: multipart/related
Part B.1: application/rlmi+xml
Part B.2: application/pidf+xml
Part B.3: application/pidf+xml
Part C: application/pidf+xml
Part D: application/pidf+xml

```
<list name="colleagues">
     <display-name>colleagues</display-name>
     <display-name xml:lang="de">Kollegen</display-name>

     <list name="university">

          <entry uri="sip:marco@ibkt.tuwien.ac.at">
               <display-name>Marco</display-name>
          </entry>

          <entry uri="sip:christoph@ibkt.tuwien.ac.at">
                <display-name>Christoph</display-name>
          </entry>

     </list>

     <entry uri="sip:alice@ibkt.tuwien.ac.at">
          <display-name> Alice </display-name>
     </entry>

     <entry uri="sip:bob@ibkt.tuwien.ac.at" />

</list>
```

Figure 20: Relation between a resource list and multipart/related document

The rls module can handle hierarchical resource lists. In other words, a list can contain several other lists which can contain other lists and so forth. Fig. 20 shows the relation between a XML resource-list definition and the multipart/related document. For each sub-list the top level document embeds another multipart/related document. If a resource has an active back-end subscription, the resource state is added to the multipart/related document (in this case in form of application/pidf+xml documents). Unfortunately, eyeBeam 1.1[7] does not support hierarchical multipart/related documents as

[7]The rls module was tested with eyeBeam 1.1 because the latest version 1.5 does not support subscriptions to resource lists anymore.

described here.

4.4.5 Back-end Subscriptions

RFC 4662 [62] states that “implementations MUST NOT present the result of one back-end subscription to more than one user, unless [...]”([62], p. 32) certain constraints are fulfilled which is considered as difficult and not recommended. The utilization of one back-end subscription for several users circumvents any authorization policies which the notifier maintains for this resource.

In other words, if a user subscribes with the same event package to several resource lists (or to the same resource list several times) and these lists contain a common resource, a single back-end subscription is sufficient for the common resource. However, a user could send subscriptions from different devices which support different data types. In this case, back-end subscriptions either only accept data types which must be supported or the module always starts a new back-end subscription, which supports the intersection of all data types supported by the different subscriptions to the resource list. The advantage of fewer back-end subscriptions (i.e, less traffic) contrasts with a higher complexity of the RLS.

Therefore, the RLS prototype starts a new back-end subscription for every resource-list subscription and every resource. One back-end subscription is assigned to one and only one resource-list subscription.

4.5 Limitations and Future Work

Apart from Authentication Service, TLS support and references in resource lists (see below), the first prototype fulfills all requirements stated in the relevant RFCs. In this section, we summarize the limitations of the first prototype and propose enhancements for future releases.

The RLS design comprises the Authentication Service as part of the SIP

Back-end Subscriber component (see Section 3.5.4). RFC 4662 [62] demands that a "RLS that uses SIP back-end subscriptions to acquire information about the resources in a resource list MUST be able to act as an authentication service [. . .], provided that local administrative policy allows it to do so" ([62], p. 31). The role of an Authentication Service is defined in RFC 4474 [56] and not implemented in the prototype.

Since resource lists contain sensitive information, the XCAP standard (RFC 4825 [72]) makes all XCAP clients implement TLS [60]. TLS is not supported by the rudimental XCAP client of the RLS prototype.

The authorization of resource-list subscribers has also space for improvements. As discussed in Section 4.4.1, a subscriber can only access service definitions, respectively resource lists, which are saved under her/his identity in a database or in the XCAP user tree on an HTTP/XCAP server. The files on the HTTP/XCAP server can be protected by a password, which is set by module parameters (see Appendix A.1). But instead of using a single password, as it is the case for the prototype, credentials for every subscriber could be stored and fetched in/from a database table. For this purpose, the rls module could reuse the OpenSER subscriber table. Moreover, we could extend the authorization policy and access common resource lists in the XCAP global tree or implement more advanced authorization policies.

As discussed in Section 2.3.2, the XDM architecture allows subscribing to changes in documents. The RLS prototype does not implement this feature. If implemented, it would allow a faster detection of changes in resource lists than it is possible by now with the caching mechanism.

Section 4.2 has outlined already that a Local Subscriber component, as suggested in the design, is not implemented yet. However, this is not easily done because the presence_xml module, which implements the functionality of a Presence Server, does not offer an appropriate interface. It is not sufficient to simply take the presence information stored in the database because the Presence Server does more than just relaying presence information (compare

Section 2.2.2).
Furthermore, an optional caching of back-end subscription information could be considered for a performance improvement.
Resource lists may include references to other lists or to entities of other lists (see Section 2.3.5). The prototype is not able to resolve these references.
The RLS prototype supports hierarchical resource lists (see Section 4.4.4) but eyeBeam, the SIP User Agent (UA) which we have used for testing, does not. It could be of interest to implement a client which supports hierarchical resource lists.

5 Aggregation of Presence Information

The introduction of a RLS can significantly improve performance of low-bandwidth links and mobile terminals as shown in the last two sections. A useful extension to standardized presence service concepts is the aggregation of presence information and therefore this section focuses on aggregation aspects.

There are two main scenarios to consider when analyzing the aggregation of presence information. The first scenario refers to the aggregation of presence information of one single Presentity which uses distinct Presence Sources (e.g., a mobile phone and a notebook). The different Presence Sources may send different views of the world which need to be aggregated into a single view. Section 5.1 discusses this aspect.

The second scenario concerns the aggregation of presence information of several Presentities. The aim is to have presence information about a group of Presentities instead of having just the presence information of all particular Presentities of the group. For instance, a user wants to know if the group (i.e., all members) is available for a phone call. For this purpose, the presence information of all group members must be aggregated. Section 5.2 discusses this issue.

Following this, we discuss aggregation using the examples of a call center and a video conference call.

The following subsections rely on the architecture of the OMA Presence SIMPLE enabler [45] as presented in Section 2.2.3.

5.1 Single Presentity State Aggregation

5.1.1 Architecture

Fig. 21 depicts one part of the OMA Presence SIMPLE enabler architecture (see Section 2.2.3). The respective reference points are defined in the OMA Presence SIMPLE architecture [47]. We assume that all shown Presence

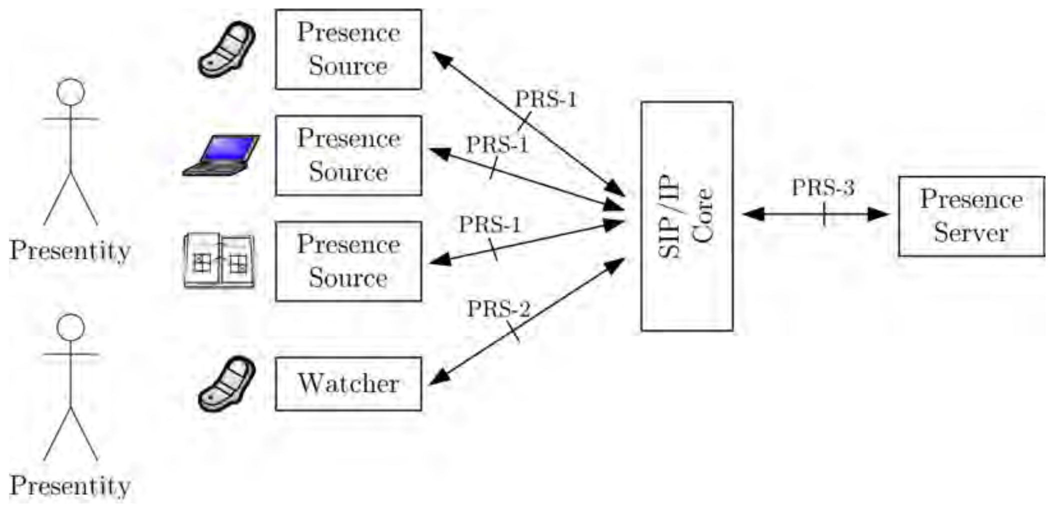


Figure 21: Aggregation of multiple Presence Sources

Sources publish presence information of the same Presentity. It means that the entity attribute of the `<presence>` element is the same for all published PIDF documents. The OMA Presence SIMPLE specification [49] demands that the URI of the entity attribute in a PIDF document equals the request-URI of the PUBLISH request which contains the PIDF document. In the IMS, this URI is known as Public User Identity.
A Watcher subscribes to the presence status of the Presentity and receives presence information composed of the presence information published by the different Presence Sources. The fact that all Presence Sources provide presence information of the same Presentity does not preclude that several people (e.g., call-center agents) use the respective terminals. The registration of several devices with the same Public User Identity (i.e., Presentity) is only possible since (full) 3GPP IMS Release 6 because Release 5 does not allow that a single Public User Identity is associated with several Private User Identities (see Section 2.1.3).

5.1.2 Composition

The Presence Server needs to combine the presence documents provided by different Presence Sources into one single raw presence document. This function is called composition.

The **OMA Presence SIMPLE specification** [49] defines a default composition policy. It specifies under which conditions and how a Presence Server should aggregate person, service or device elements (see Section 2.4.3). For instance, two `<tuple>` elements are aggregated into one `<tuple>` element if they contain identical `<contact>` elements, identical `<class>` elements, identical `<service-id>` and `<version>` elements of `<service-description>` elements and if there are "no conflicting elements (i.e., same elements with different values) or attributes under the `<tuple>` elements" ([49], p. 28). Different timestamps are not considered as conflicting elements. Similar rules apply for the aggregation of `<device>` and `<person>` elements. The specification allows local policies which augment the default composition policy but these policies may not violate the default policy. Consequently, additional policies to the default policy of the OMA Presence SIMPLE specification [49] are needed if, for instance, different services or different statuses should be composed.

The (expired) drafts "A Processing Model for Presence" (draft-rosenberg-simple-presence-processing-model [67]) and "Composing Presence Information" (draft-schulzrinne-simple-composition [78]) discuss presence composition. **A Processing Model for Presence** introduces a model for processing presence information. The model divides the different steps at the Presence Server into SIP subscription processing and presence document processing whereas the latter is further divided into collection, composition, privacy filtering, watcher filtering and post-processing composition.

For this thesis the step composition is of primary interest. The applied composition policy is selected by the presence authorization rules. According to [67], composition consists of the following steps:

1. Correlation
2. Conflict resolution
3. Merging

The first step correlates existing presence information to new presence information. For instance, provided that a Presentity represents a single person, the Presence Server can deduce that the person is on the phone if a telephone service reports that it is busy. Correlation identifiers (e.g., device-id which links a service to a device) can support here.
The second step resolves the conflict of multiple person/service/device elements for the same person/service/device. After a conflict has been detected the conflict resolution itself can be based on the likelihood how accurate the information of each source is (e.g., if one device reports user activity, it probably is a more accurate source). Another possibility is to take the identity of the source (if known) into account.
The last step is merging. It allows merging different services or devices into a single service or device. Therewith it is possible, for instance, to merge multiple telephony services reported by multiple sources into a single telephony service. One way to choose the services/devices that should be merged is to define a particular attribute as selector (e.g., status online). The draft [67] calls this element a pivot element. How the different attributes of the services are merged is a matter of local policy. If the services have different SIP URIs as contact address, the draft [67] proposes that the Presence Server creates a URI which represents the collection of services. “Requests made to that URI could fork to the set of services that were combined together” ([67], p. 14).
The draft **Composing Presence Information** [78] also discusses composition and partly overlaps with [67]. As goal it defines to remove information that is either stale, contradictory or redundant. The draft assumes that a Presentity is a single human being and does not represent a group of people.

The composition process is divided into the following steps:

1. Discarding
2. Derivation
3. Conflict resolution
4. Merging

Discarding removes tuples with stale or redundant information. This is the case if services have the status closed, tuples are older than a certain threshold or devices are not referenced by any service.
Derivation provides useful new data and corresponds to the step correlation of [67].
Conflict resolution also corresponds to a step of the same name in [67]. For instance, conflicts can be found by comparing values of distinct elements (e.g., sleeping and steering) whereas the focus is mainly on the `<person>` element. To decide which tuples of the conflicting elements should be included in the final presence document, the draft suggests several heuristics, namely recent tuple, trustworthy tuple based on type or identity of a source, or the value of another element (e.g., user input).
Merging has a different meaning than in [67]. "Merging combines several tuples that logically represent the same information" ([78], p. 11). Therefore, only `<person>` elements or services with the same contact URI are merged.
In contrast to [67], the draft [78] specifies a composition policy format in XML. Several XML elements are defined for the different composition steps. The XML patch format [82] is used for the insertion of new information whereas XPath [20] expressions define where the new contents should be inserted. The format has limited possibilities but may serve as a basis for a more comprehensive format which allows more complex compositions. The authors of the draft [78] present their approach also in [80].

The paper **A Script-based Approach to Distributed Presence Aggregation** [21] tries to solve the problem of composition with a script-based approach. The authors have chosen JavaScript to ease development for people who have already experience in web programming. The JavaScript-based Presence Aggregation Language (PAL) is introduced and used to combine presence information from different sources into a single document. Additionally, PAL scripts can modify, add, or remove information and force or block sending of status notifications.
Provided that a Presentity represents a group of people, the composition process becomes more complex. For instance, if one telephony service is busy this need not mean that the group is busy.
In conclusion, no standard exists which allows advanced composition of presence information like Boolean operations on entities. The drafts discussed in this section and Section 2.4.7 can help in defining new composition policies and composition policy formats.

5.1.3 Integration of XDM

If the composition policy format is defined in XML, the respective documents can be stored on XDM Servers. For it, not only a XML schema but also a new application usage needs to be defined. This document introduces the name *composition-policy* for this non-existing application usage.
The architecture discussed in this section makes already use of the Presence XDM (see Fig. 6) in order to apply presence authorization rules. In the same way, we add a new composition-policy XDMS which stores the composition policy to the architecture depicted in Fig. 6.

5.2 Multiple Presentities State Aggregation

The task of merging presence information from several Presentities is similar to the task of merging presence information from different Presence Sources of a single Presentity. The main difference is that, a priori, no unique identifier connects the different presence information documents. This connection can be established using a URI List [52]. A list has a unique identifier and the presence information of all Presentities in the list can be merged. We introduce the term Presence Aggregation Server as name for the entity where the aggregation of presence information happens.

5.2.1 An Architecture based on a Presence Server

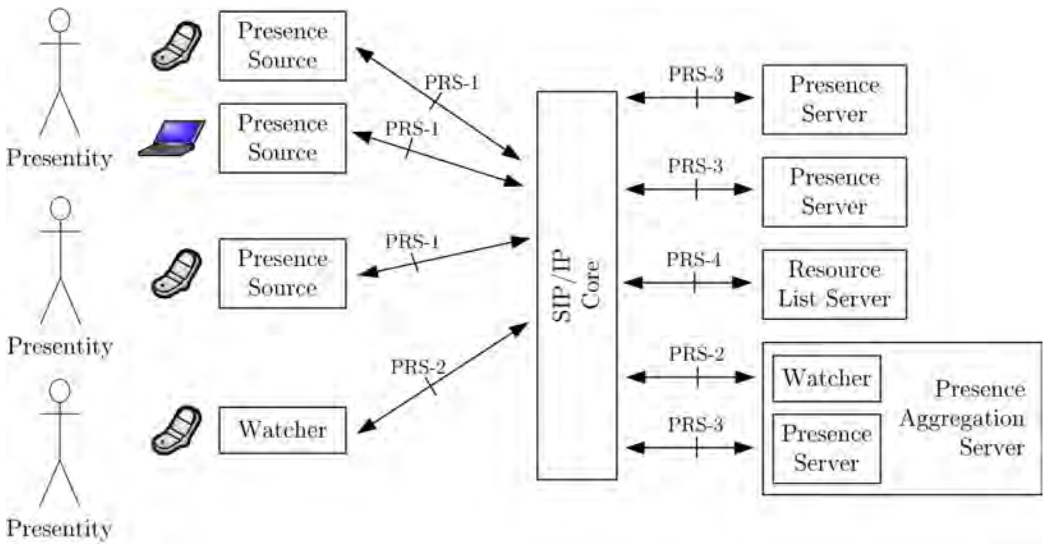


Figure 22: Aggregation solution with a Presence Server

Fig. 22 models the Presence Aggregation Server as extended Presence Server with integrated Watcher (i.e., subscriber) functionality. A stand-alone Presence Server as defined by the OMA Presence SIMPLE specification [49] is not sufficient. As per the OMA Presence SIMPLE specification, a Presence Server must be able to receive PUBLISH requests but does not need to support subscriptions to presence information stored on other Presence Servers. If a Presence Aggregation Server wants to aggregate not only locally stored presence information, it needs to subscribe to the presence information managed by different Presence Servers. Moreover, the OMA Presence SIMPLE specification states that a Presence Server "handles publications from one or multiple Presence Source(s) of a certain Presentity" ([49], p. 24) and not of several Presentities like in this case.

The integrated Watcher has two possibilities to collect the presence information of the different Presentities. The Watcher can fetch the appropriate URI List from a Shared XDMS via XCAP and subscribe to all the URIs in the list. Alternatively, the Watcher delegates this work to a RLS. The Watcher subscribes to the list and receives the collected presence information of all Presentities. In the first case the Watcher must support XCAP in order to fetch the URI List; in the latter case the Watcher must support the event list extension [62] in order to subscribe to the RLS. In any case, the integrated Watcher does the subscription with the identity of the watcher of the aggregate information.

A client which is interested in the aggregate presence information subscribes to a URI which identifies a URI List. The SUBSCRIBE request is routed to the Presence Aggregation Server whereon the integrated Watcher collects the presence information of all Presentities on the list. Following a composition policy for this list, the Presence Aggregation Server composes the presence documents of all Presentities into a single document. The integrated Presence Server can process this raw document like any presence document for a single Presentity and apply authorization rules, event notification filtering and partial notification (see Section 2.2.2).

5.2.2 An Architecture based on a RLS

Fig. 23 shows a slightly different approach. The Presence Aggregation Server is directly integrated into a Resource List Server. According to to the OMA Presence SIMPLE specification [49], a RLS "SHALL support list subscriptions to the presence event package, according to the RLS procedures described in [RFC4662]" ([49], p. 32). A NOTIFY message from a RLS contains

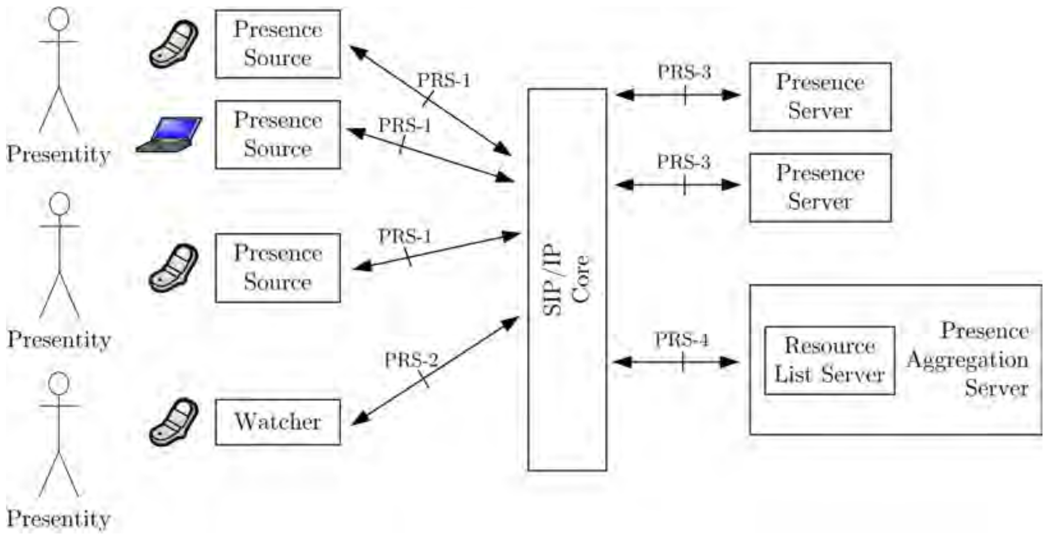


Figure 23: Aggregation solution with a RLS

a multipart/related [33] body. The root document of the multipart/related body must be a RLMI document [62] as discussed in Section 3.1.

The key for presence aggregation is that the top-level `<list>` element of a RLMI document itself may refer to "a section within the multipart/related body that contains aggregate state information for the resources contained in the list" ([62], p.12). The definition of such aggregate information is not part of the RLS specification [62] and is defined on a per-package basis. In other words, the RLMI format allows inserting aggregate information about the complete list. This aggregate information is derived from the received presence information of the individual resources in the list according to some composition policy.

If now a subscriber should only be notified about the aggregate state and the presence status of the particular resources must be hidden, no `<resource>` elements are included in the RLMI document. Then the multipart/related document only consists of the RLMI document and the presence document containing the aggregate state. At any time, the RLS can include detailed presence information about a particular Presentity. Hence, a `<resource>`

element is added to the RLMI document and the presence document for the particular Presentity is included in the multipart/related document.
RFC 4462 [62] mentions that, in general, a RLS should not rely on being able to read the MIME bodies received from any back-end subscription because they can be encrypted. However, the same problem exists if a Presence Aggregation Server implemented as Presence Server receives encrypted MIME bodies.
It is questionable how Watchers handle aggregate information of a complete list and how they render this information if no information about individual resources is included in a RLMI document. Additionally, a RLS requires more features of a terminal because the terminal must support the eventlist option tag and be able to accept the application/rlmi+xml MIME type plus the application/pidf+xml MIME type.
If a client only wants to subscribe to aggregate information, the RLS could also return just a NOTIFY with the aggregate presence information like a normal Presence Server. This equals the architecture described before when the integrated Watcher subscribes to all resources[8]. The crucial point is that a particular subscription is either a subscription to a list or to a single resource but cannot be mixed. "In particular, this means that RLSs MUST NOT send NOTIFY messages that do not contain RLMI for a subscription if they have previously sent NOTIFY messages in that subscription containing RLMI. Similarly, RLSs MUST NOT send NOTIFY messages that do contain RLMI for a subscription if they have previously sent NOTIFY messages in that subscription which do not" ([62], p. 7).

5.2.3 Composition

The conclusions of 5.1.2 also apply for the architecture we have discussed in this section. Additionally, a composition policy can consider the different

[8]In contrast to the first architecture a RLS retrieves a Presence List from a RLS XDMS instead of a URI List from a Shared XDMS. The Presence List can include a link to a URI List.

identifiers of the Presentities.
Resource Lists as specified in [65] can be nested. This should be taken into account when defining the composition policies.

5.2.4 Integration of XDM

The proposed Presence Aggregation Server uses four different XDM Servers (see Fig. 6): a Shared XDMS, a RLS XDMS, a Presence XDMS and a new composition-policy XDMS.
A Presence Aggregation Server implemented as RLS aggregates the presence information of resources in a particular Presence List which is stored on a RLS XDMS. The RLS-service definitions stored on a RLS XDMS contain these lists directly or include a link to a URI List stored on a Shared XDMS. If no RLS is used, the integrated Watcher of a Presence Aggregation Server retrieves a URI List from a Shared XDMS. The Shared XDM specification [52] defines only four types of lists (see Section 2.3.5) but new types can be registered.
A Presence Aggregation Server applying presence authorization rules fetches them from a Presence XDMS. The composition policy is retrieved from a composition-policy XDMS, which is not part of a standard yet.

5.3 Comparison

To conclude the discussion on presence aggregation, we outline some strength and drawbacks of the proposed architectures. Independent of the particular architecture, there is no difference for a Watcher if it subscribes to aggregate information or presence information of a single Presentity as there is no difference between a subscription to a single resource and to a resource list. Comparing the composition of presence information of a single Presentity to the aggregation of presence information of several Presentities, the problem is somehow just moved to another level. In the first case, group members

have a common (IMS) subscription and a common Public User Identity, but different Private Identities. In the latter case, group members have a common resource list but nothing else in common. Every group member has its own Public User Identity.
Tab. 6 summarizes some strengths and drawbacks of all discussed architectures.

Single Presentity State Aggregation
+ All reference points, except between PS and composition-policy XDMS, follow existing standards. + Forking of a request to all group members is simple. - Requires (full) 3GPP IMS Rel. 6 or later (if several devices are used) - It is harder to differentiate between particular group members because the all have the same Public User Identity. - All group members must have the same IMS subscription.

Multiple Presentities State Aggregation (Presence Aggregation Server as PS)	
+ All reference points, except between PS and composition-policy XDMS, follow existing standards. + There are no constraints for group members (e.g., same subscription). + Due the different Public User Identities of group members, they are easier specifiable. - It is necessary to maintain and access resource lists. - No simple forking of a request to all group members is possible.	
Watcher subscribing to all resources	Watcher utilizing a RLS
+ Easy integration of locally known presence information.	+ The RLS manages all subscriptions to particular group members.
+ no RLS is needed.	- The main advantages of a RLS (reducing the traffic between Watcher and PS and throttle the rate of notifications) are not true anymore if both Watcher and PS are in the core network.
+ Support of subscriptions to presence information is sufficient.	- A Presence Aggregation Server cannot use locally known presence information.
- The Watcher must manage a dialog for each group member.	- A Watcher must support eventlist.

Multiple Presentities State Aggregation (Presence Aggregation Server as RLS)
+ All reference points, except between PS and composition-policy XDMS, follow existing standards. + Easy integration of complete presence information about particular group members. - Watcher in the terminal must support eventlist. - It is necessary to maintain and access resource lists. - No simple forking of a request to all group members is possible.

Table 6: Comparison of aggregation architectures

5.4 Call-Center Scenario

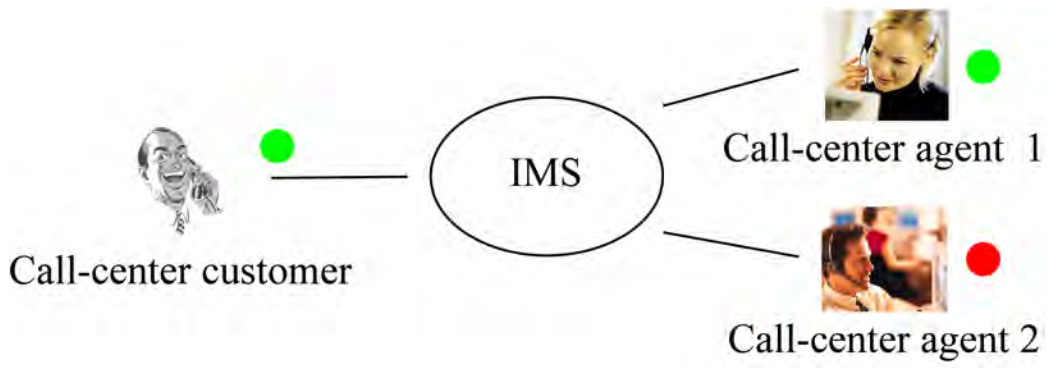


Figure 24: Call-center scenario

This scenario is based on a simplified call center. On the right side, Fig. 24 shows some call-center agents. Their terminals are connected to an IMS network and they publish the presence status of the agents. In Fig. 24 the green dot for agent 1 indicates that he is currently available; the red dot for agent 2 indicates that the agent is currently busy. The customer on the left side is not interested in the presence statuses of single agents. He wants to know if the call center is available (i.e., if one of the agents is available). Since one agent is available, the customer sees a green dot.
The following Subsections model the call center first as a single Presentity and then as a group of Presentities.

5.4.1 Single Presentity State Aggregation

The architecture presented in Section 5.1 serves as basis for the discussions in this section. The call center is modeled as a single Presentity. Every user agent has its own Private User Identity but registers the same Public User Identity with the IMS network. This is only possible since (full) 3GPP IMS Release 6.
Each terminal registers the Public User Identity *sip:callcenter@ex.org* with

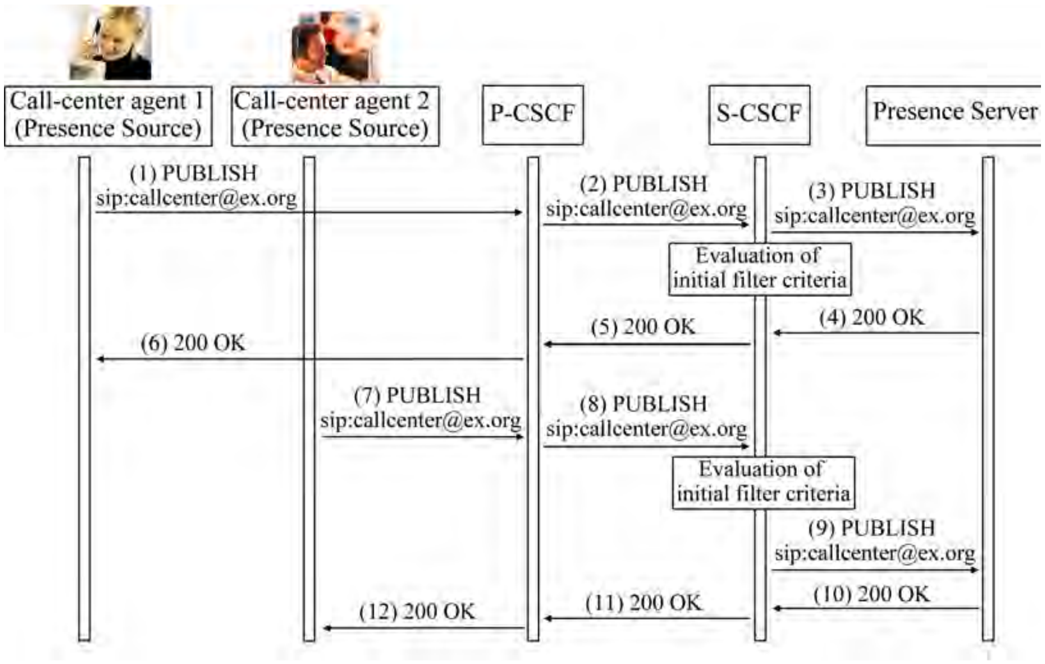

Call-center agent 1 (Presence Source)
Call-center agent 2 (Presence Source)
P-CSCF
S-CSCF
Presence Server
(1) PUBLISH sip:callcenter@ex.org
(2) PUBLISH sip:callcenter@ex.org
(3) PUBLISH sip:callcenter@ex.org
Evaluation of initial filter criteria
(4) 200 OK
(5) 200 OK
(6) 200 OK
(7) PUBLISH sip:callcenter@ex.org
(8) PUBLISH sip:callcenter@ex.org
Evaluation of initial filter criteria
(9) PUBLISH sip:callcenter@ex.org
(10) 200 OK
(11) 200 OK
(12) 200 OK

Figure 25: The Presence Sources of the agents publishing presence information

the IMS core. Additionally, a second Public User Identity, which is exclusively associated with the Private User Identity used with this device, is registered implicitly. For instance, *sip:agent1@ex.org* is registered for the terminal of agent 1 and *sip:agent2@ex.org* for the terminal of agent 2.

Upon a successful registration with the IMS core, the Presence Source applications which run on the terminals of the call-center agents start to publish their presence information. The message flow is illustrated in Fig. 25. First, agent 1 sends a PUBLISH request (1) with the Request-URI set to *sip:callcenter@ex.org*. The request is routed to a S-CSCF where one of the initial filter criteria indicates that this request should be forwarded to the Presence Server (3). The Presence Server handles the request as any other PUBLISH request following the procedure described in Section 5.4.1 of the OMA Presence SIMPLE specification [49].

The terminal of agent 2 also sends a PUBLISH request (7) with the Request-URI set to *sip:callcenter@ex.org*. The message could be routed to a different P-CSCF and a different S-CSCF (not shown) but the S-CSCF forwards the request to the same Presence Server (9).
The message flow corresponds to the usual publication of presence information as described in the OMA Presence SIMPLE specification [49].
Minimalistic examples of the PIDF documents sent in the bodies of the PUBLISH requests are displayed in Listing 7 for agent 1 and Listing 8 for agent 2. Both indicate one service but for agent 2 the status of its service is closed. The Public User Identity registered additionally to *sip:callcenter@ex.org* is used as contact address for a service. The `<person>` element in Listing 8 includes an `<activity>` element that indicates that the person is currently on the phone.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:callcenter@ex.org">
  <tuple id="a1232">>
    <status>
      <basic>open</basic>
    </status>
    <contact>sip:agent1@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>
  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
  </pdm:person>
</presence>
```

Listing 7: Published PIDF document of agent 1 (body of messages (1)-(3) in Fig. 25)

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:callcenter@ex.org">
 <tuple id="a4222">>
   <status>
    <basic>closed</basic>
   </status>
   <contact>sip:agent2@ex.org</contact>
   <timestamp>2007-05-29T09:15:16Z</timestamp>
 </tuple>
 <pdm:person id="a1233">
   <rpid:class>agent</rpid:class>
    <rpid:activities>
       <rpid:on-the-phone/>
    </rpid:activities>
 </pdm:person>
</presence>
```

Listing 8: Published PIDF document of agent 2 (body of messages (7)-(9) in Fig. 25)

According to our scenario, a call-center customer is interested in the presence status of the call center. As depicted in Fig. 26, the terminal sends a SUBSCRIBE request (1) with the Request-URI set to *sip:callcenter@ex.org* to the P-CSCF. The SUBSCRIBE request is routed over several hops to the Presence Server which successfully accepts the request. The Presence Server sends a XCAP request (7) to the composition-policy XDMS and fetches the composition policy document for *sip:callcenter@ex.org*. Furthermore, the Presence Server gets the presence authorization rules from the Presence XDMS[9] (9-10) like for every other subscription. According to the composition policy, the Presence Server creates a raw presence document out of all received presence documents. Afterward, the Presence Server can apply content rules, event notification filtering and partial notification on this raw document. Eventually, the transformed presence document is included in a NOTIFY message

[9]The Presence Server can also fetch the presence authorization rules before any composition policy.

and sent to the Watcher (11-13).

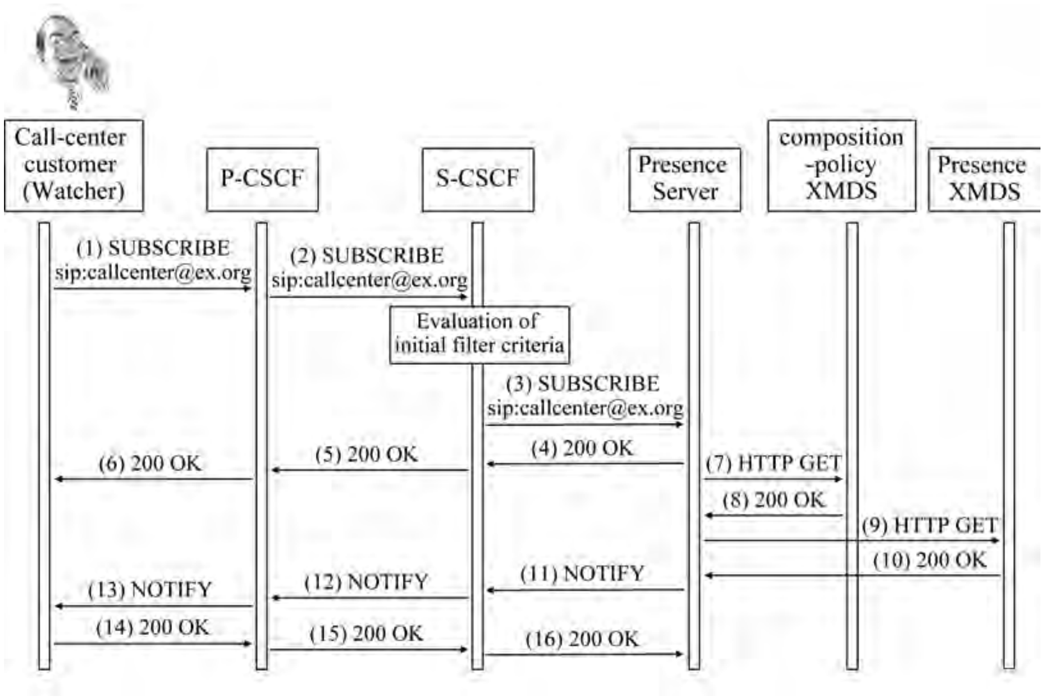


Figure 26: Subscription to aggregate presence information (single Presentity)

Since the aggregation of presence information does not affect the interfaces, the message flow corresponds to the usual SUBSCRIBE/NOTIFY procedure as described in the OMA Presence SIMPLE specification [49]. The only difference is the download and application of an advanced composition policy. Listing 9 shows one possibility how the two received presence documents of Listing 7 and Listing 8 can be aggregated. The raw presence document includes the `<person>` element without the `<on-the-phone/>` element because one agent is available. The service with the basic status open is included too.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:callcenter@ex.org">

  <tuple id="a1232">>
    <status>
      <basic>open</basic>
    </status>
    <contact>sip:agent1@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>

  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
  </pdm:person>

</presence>
```

Listing 9: Aggregate PIDF document with the contact address of an agent (body of messages (11)-(13) in Fig. 26)

It is a matter of policy if all available services are included or only one of the available services. If all available services are included, the terminal of the Watcher must decide to which service it should try to connect to. If this service is not available anymore, the terminal can try immediately another service.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:callcenter@ex.org">

  <tuple id="a1232">>
    <status>
      <basic>open</basic>
    </status>
    <contact>sip:callcenter@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>

  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
  </pdm:person>

</presence>
```

Listing 10: Aggregate PIDF document with the contact address of the call center (body of messages (11)-(13) of Fig. 26)

Instead of the Public User Identity which is specific for a user agent, the common identity *sip:callcenter@ex.org* can be included as contact address for the service. Listing 10 depicts the PIDF document with the SIP URI of the call center as contact address. The decision which agent should be called is then up to the S-CSCF or an AS on the path. An easy and reasonable guideline would be to fork the request to all agents. If an agent is available he/she answers the call. This approach has several advantages. The agents do not need to have an extra Public User Identity and a customer only gets to know one Public User Identity - *sip:callcenter@ex.org*. Moreover, there is a high chance in a call center that the services reported as available in the presence information are not available anymore when the call is initiated. But still, the presence information that the call center is available is correct as long as one arbitrary service is available. Therefore, it is reasonable to postpone the decision which agent(s) to call until the session initiation.

5.4.2 Multiple Presentity State Aggregation

In this scenario each call center agent has its own IMS subscription and does not share a common Public User Identity with all agents. The common identity is the identity of a URI List where the Public User Identities of all agents are in.
Since we are not interested in complete presence documents of a single agent, the Presence Aggregation Server does not act as RLS but rather as Presence Server.
It is quite possible that the published documents of all call-center agents end up at the same PS which is a Presence Aggregation Server at the same time. Then the Presence Aggregation Server does not need to subscribe to presence information managed by other Presence Servers. But still the Presence Aggregation Server needs to carry out local subscriptions. It needs to know when one of the Presentities in the list has changed its status, possibly composite the presence documents of a single Presentity and apply presence authorization rules dependent on the watcher.
We assume that the presence information we want to aggregate is spread over several Presence Servers. For instance, this will probably be the case if volunteers with different backgrounds run a call center for a joint project. Further we assume that the Presence Aggregation Server itself subscribes to all resources and does not utilize a RLS.
Fig. 27 depicts the message flow for publishing presence information. In contrast to Fig. 25, the terminals send PUBLISH requests with different Public User Identifiers as Request-URI. The PUBLISH requests terminate at different Presence Servers.
Fig. 28 shows the message flow for subscribing to aggregate presence information. The message flow between the Watcher and the Presence Aggregation Server corresponds to the message flow in Fig. 26. Upon the reception of a SUBSCRIBE request, the Presence Aggregation Server downloads the corresponding URI List from the Shared XDM Server (4-5). Subsequently, it

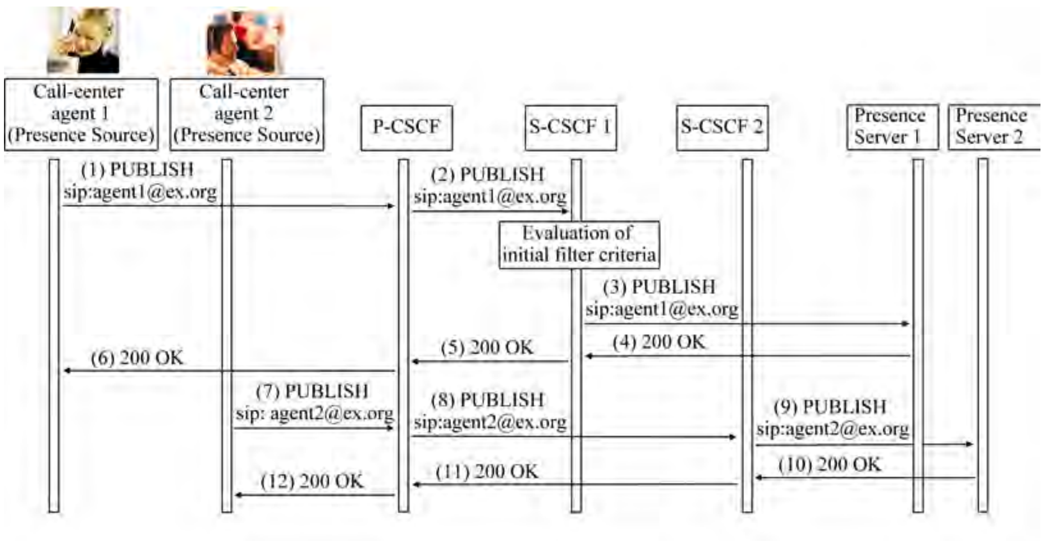

Call-center agent 1 (Presence Source)
Call-center agent 2 (Presence Source)
P-CSCF
S-CSCF 1
S-CSCF 2
Presence Server 1
Presence Server 2
(1) PUBLISH sip:agent1@ex.org
(2) PUBLISH sip:agent1@ex.org
Evaluation of initial filter criteria
(3) PUBLISH sip:agent1@ex.org
(4) 200 OK
(5) 200 OK
(6) 200 OK
(7) PUBLISH sip: agent2@ex.org
(8) PUBLISH sip:agent2@ex.org
(9) PUBLISH sip:agent2@ex.org
(10) 200 OK
(11) 200 OK
(12) 200 OK

Figure 27: The Presence Sources of the agents (Presentities) publishing presence information

subscribes to all URIs in the list and receives NOTIFY requests containing the presence information of a particular Presentity (9-16). For this part, Fig. 28 omits the CSCFs on the path. In the next step, the Presence Aggregation Server downloads the composition policy for this URI List from the composition-policy XDMS (17-18). Now the Presence Aggregation Server has all necessary information to aggregate the presence documents into one. The aggregate presence document is handled like any other presence document. Presence authorization rules are transfered from the Presence XDMS (19,20) and applied. Eventually, the aggregate presence document is sent to the Watcher in a NOTIFY request (21-23).
Listing 11 and Listing 12 show the PIDF documents published by agent 1, respectively agent 2. These PIDF documents only differ from the PIDF documents of Section 5.4.1 in the entity attribute. In the first place, the two PIDF documents are not related to each other. The aggregate document in Listing 13 uses the URI identifying the URI List for the entity attribute.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:agent1@ex.org">
  <tuple id="a1232">>
    <status>
      <basic>open</basic>
    </status>
    <contact>sip:agent1@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>
  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
  </pdm:person>
</presence>
```

Listing 11: Published PIDF document of agent 1 (body of messages (1)-(3) of Fig. 27)

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:agent2@ex.org">
  <tuple id="a4222">>
    <status>
      <basic>closed</basic>
    </status>
    <contact>sip:agent2@ex.org</contact>
    <timestamp>2007-05-29T09:15:16Z</timestamp>
  </tuple>
  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
    <rpid:activities>
      <rpid:on-the-phone/>
    </rpid:activities>
  </pdm:person>
</presence>
```

Listing 12: Published PIDF document of agent 2 (body of messages (7)-(9) of Fig. 27)

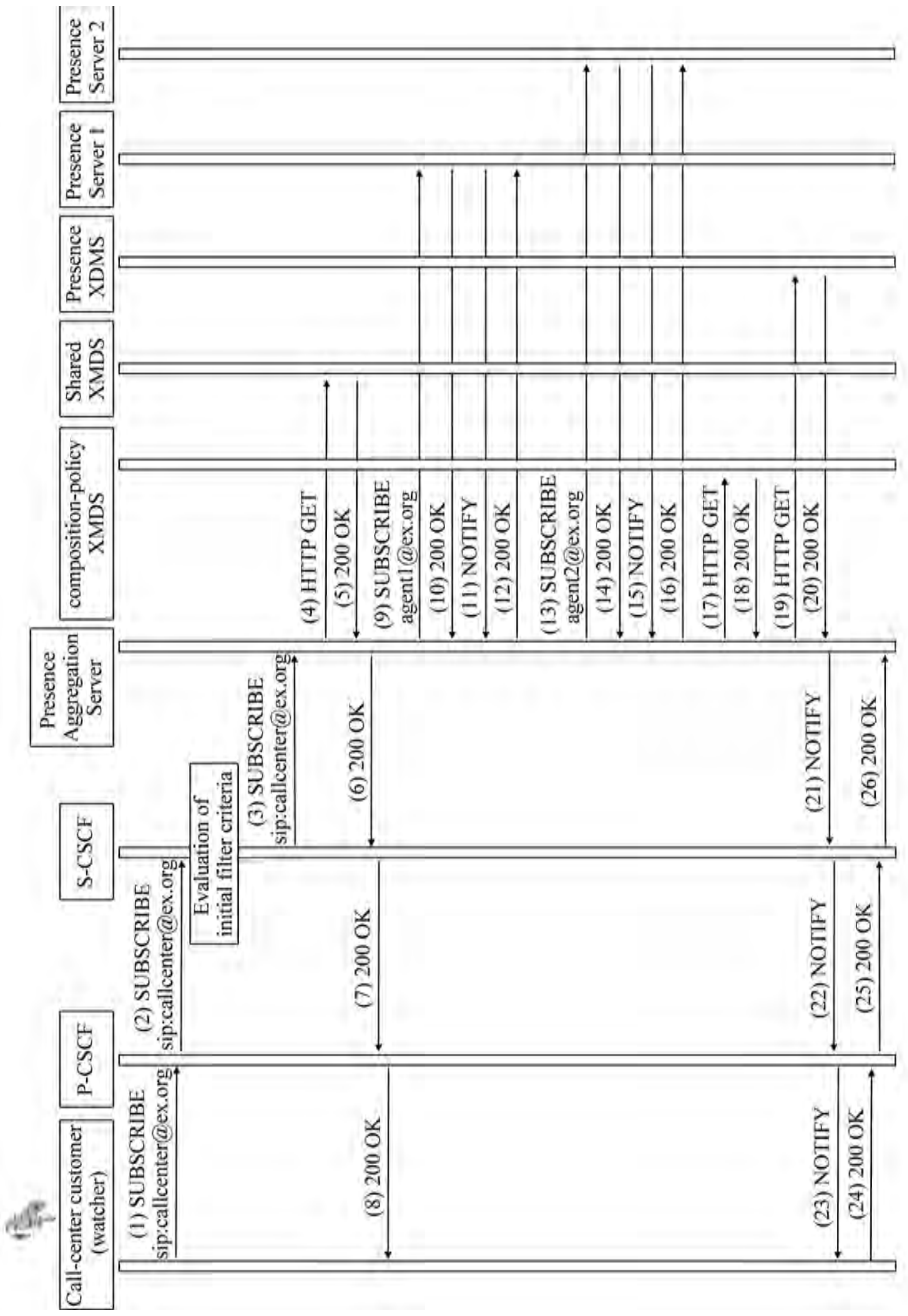


Figure 28: Subscription to aggregate presence information (multiple Presentities)

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          entity="sip:callcenter@ex.org">

  <tuple id="a1232">>
    <status>
      <basic>open</basic>
    </status>
    <contact>sip:callcenter@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>

  <pdm:person id="a1233">
    <rpid:class>agent</rpid:class>
  </pdm:person>

</presence>
```

Listing 13: Aggregate PIDF document ((body of messages (21)-(23) of Fig. 28)

Again the Presence Aggregation Server can choose between two addresses to include as contact address of a service. The Public User Identity of one of the agents or the identity of the URI List. The latter does not unfold the identity of an agent but makes the call processing more difficult. In the scenario of Section 5.4.1, the call could be forked to all terminals because they have been registered with the same Public User Identity. In this scenario an AS must handle the call and rely it to some agent(s) in the list.

5.5 Video-Call Scenario

This scenario is basically an enhancement of the call-center scenario. The discussion leader of a video conference wants to initiate the conference only if all mandatory members of the group can send/receive video on one of their currently registered devices. Optional members are not relevant for the presence status. In this scenario, the group members are Alice, Bob and Carol. Alice and Carol are mandatory members; Bob is an optional member.

5.5.1 Presence Information Documents

The message flows of this scenario correspond to the message flows of the call-center scenario in Section 5.4.2. But in this example the published presence information is richer. As consequence, the aggregation of this information is more difficult. This section shows examples how terminals can indicate video support and different levels of group membership and how the aggregate presence information could look like. The details of the aggregation process, how the aggregation rules look like and how they can be modeled, are again left for further studies.

Draft draft-ietf-simple-prescaps-ext [34] defines elements to model service capabilities (see Section 2.4.6). According to Section 5.3.1.2 of 3GPP TS 24.141 [16], a Presence Source must implement this draft "if it wants to make use of SIP user agent capabilities in the presence document" ([16], p. 11). The OMA Presence SIMPLE enabler does not mention this draft in its specification [49] but it allows Presence Sources to publish elements from other PIDF extensions as long as these extensions do not violate other extensions.

Listing 14, Listing 15 and Listing 16 show exemplary PIDF documents for the group members Alice, Bob and Carol. The `<video>` element, which is a child element of `<servcaps>`, indicates whether a service supports video or not. The `<class>` element shows if this person is a mandatory or optional member of the group. The RPID (see Section 2.4.5) defines the `<class>` element, which describes the class of a service, device, or person. The naming of the class is left to the Presentity. This makes automatic checks more complicated but using this element is more straightforward than defining a new extension to PIDF.

However, if the Presence Aggregation Server does not receive presence information from all group members, it cannot know if all mandatory members are available. Fortunately, the `<entry>` element of URI Lists [52] can contain elements from other namespaces. A new element could be defined which

indicates if a member is optional or mandatory.
The service included in the presence information of Carol (listing 16) supports video but does not support audio. Moreover, the contact address is a tel URI and not a SIP URI. Is it sufficient for a video conference to support only video? Does the contact address of the service must be a SIP address? The example shows that a reasonable aggregation of presence information can become very complex.

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
          entity="sip:alice@ex.org">

  <tuple id="a8098a.67236">>
    <status>
      <basic>open</basic>
    </status>
    <pcp:servcaps>
      <pcp:video>true</pcp:video>
      <pcp:audio>true</pcp:audio>
    </pcp:servcaps>
    <contact>sip:alice@ex.org</contact>
    <timestamp>2007-05-29T09:14:56Z</timestamp>
  </tuple>

  <pdm:person id="a1233">
    <rpid:class>mandatory member</rpid:class>
  </pdm:person>

</presence>
```

Listing 14: Published PIDF document of Alice

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
          entity="sip:bob@ex.org">

  <tuple id="a8098a.67236">>
    <status>
      <basic>closed</basic>
    </status>
    <pcp:servcaps>
      <pcp:video>true</pcp:video>
      <pcp:audio>true</pcp:audio>
    </pcp:servcaps>
    <contact>sip:bob@ex.org</contact>
    <timestamp>2007-05-29T09:14:59Z</timestamp>
  </tuple>

  <pdm:person id="a1233">
    <rpid:class>optional member</rpid:class>
  </pdm:person>

</presence>
```

Listing 15: Published PIDF document of Bob

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
          xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
          xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
          entity="sip:carol@test.com">

  <tuple id="a8098a.67111">>
    <status>
      <basic>open</basic>
    </status>
    <pcp:servcaps>
      <pcp:video>true</pcp:video>
      <pcp:audio>no</pcp:audio>
    </pcp:servcaps>
    <contact priority="0.9">tel:+1-212-555-1234</contact>
    <timestamp>2007-05-29T09:15:03Z</timestamp>
  </tuple>

  <pdm:person id="s007">
    <rpid:class>mandatory member</rpid:class>
    </pdm:person>

</presence>
```

Listing 16: Published PIDF document of Carol

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
          entity="sip:conference-group@ex.org">

  <tuple id="a8012r.67111">>
    <status>
      <basic>open</basic>
    </status>
    <pcp:servcaps>
      <pcp:video>true</pcp:video>
    </pcp:servcaps>
    <contact>sip:conference-group@ex.org</contact>
    <timestamp>2007-05-29T09:15:03Z</timestamp>
  </tuple>

</presence>
```

Listing 17: Aggregate PIDF document

```
 --50UBfW7LSCVLtggUPe5z
   Content-Transfer-Encoding: binary
   Content-ID: <nXYxAE@ps.ex.org>
   Content-Type: application/rlmi+xml;charset="UTF-8"

   <?xml version="1.0" encoding="UTF-8"?>
    <list xmlns="urn:ietf:params:xml:ns:rlmi"
          uri="sip:conference-group@ex.org"
          version="1" fullState="true"
          cid="aURBsM@ps.ex.org">
      <name xml:lang="en">Video conference group</name>

      <resource uri="sip:bob@ex.org">
        <name>Bob Smith</name>
        <instance id="juwigmtboe" state="active"
                  cid="bUZBsM@ps.ex.org"/>
      </resource>

      <resource uri="sip:alice@ex.org">
        <name>Alice Anders</name>
        <instance id="hqzsuxtfyq" state="active"
                  cid="ZvSvkz@ps.ex.org"/>
      </resource>
     <resource uri="sip:carol@test.com">
       <name>Carol Evol</name>
        <instance id="pozcuxttyq" state="active"
                 cid="U5Gvkr@ps.ex.org"/>
        </resource>
    </list>

   --50UBfW7LSCVLtggUPe5z
   Content-Transfer-Encoding: binary
   Content-ID: <aURBsM@ps.ex.org>
   Content-Type: application/pidf+xml;charset="UTF-8"

    <?xml version="1.0" encoding="UTF-8"?>
      <presence xmlns="urn:ietf:params:xml:ns:pidf"
          xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
          entity="sip:conference-group@ex.org">

        <tuple id="a8012r.67111">>
          <status>
            <basic>open</basic>
          </status>
          <pcp:servcaps>
            <pcp:video>true</pcp:video>
          </pcp:servcaps>
          <contact>sip:conference-group@ex.org</contact>
          <timestamp>2007-05-29T09:15:03Z</timestamp>
        </tuple>

      </presence>

 --50UBfW7LSCVLtggUPe5z
  Content-Transfer-Encoding: binary
  Content-ID: <bUZBsM@ps.ex.org>
  Content-Type: application/pidf+xml;charset="UTF-8"

  <?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
              xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
              xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
              xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
              entity="sip:bob@ex.org">

      <tuple id="a8098a.67236">>
        <status>
          <basic>closed</basic>
        </status>
```

```
        <pcp:servcaps>
          <pcp:video>true</pcp:video>
          <pcp:audio>true</pcp:audio>
        </pcp:servcaps>
        <contact>sip:bob@ex.org</contact>
        <timestamp>2007-05-29T09:14:59Z</timestamp>
      </tuple>

      <pdm:person id="a1233">
        <rpid:class>optional member</rpid:class>
      </pdm:person>

    </presence>

--50UBfW7LSCVLtggUPe5z
  Content-Transfer-Encoding: binary
  Content-ID: <ZvSvkz@ps.ex.org>
  Content-Type: application/pidf+xml;charset="UTF-8"

    <?xml version="1.0" encoding="UTF-8"?>
    <presence xmlns="urn:ietf:params:xml:ns:pidf"
              xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
              xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
              xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
              entity="sip:alice@ex.org">

      <tuple id="a8098a.67236">>
        <status>
          <basic>open</basic>
        </status>
        <pcp:servcaps>
          <pcp:video>true</pcp:video>
          <pcp:audio>true</pcp:audio>
        </pcp:servcaps>
        <contact>sip:alice@ex.org</contact>
        <timestamp>2007-05-29T09:14:56Z</timestamp>
      </tuple>

      <pdm:person id="a1233">
        <rpid:class>mandatory member</rpid:class>
      </pdm:person>

    </presence>

--50UBfW7LSCVLtggUPe5z
  Content-Transfer-Encoding: binary
```

```
Content-ID: <U5Gvkr@ps.ex.org>
Content-Type: application/pidf+xml;charset="UTF-8"

 <?xml version="1.0" encoding="UTF-8"?>
   <presence xmlns="urn:ietf:params:xml:ns:pidf"
           xmlns:pdm="urn:ietf:params:xml:ns:pidf:data-model"
           xmlns:rpid="urn:ietf:params:xml:ns:pidf:rpid"
           xmlns:pcp="urn:ietf:params:xml:ns:pidf:caps"
           entity="sip:carol@test.com">

     <tuple id="a8098a.67111">>
       <status>
         <basic>open</basic>
       </status>
       <pcp:servcaps>
         <pcp:video>true</pcp:video>
         <pcp:audio>no</pcp:audio>
       </pcp:servcaps>
       <contact priority="0.9">tel:+1-212-555-1234</contact>
       <timestamp>2007-05-29T09:15:03Z</timestamp>
     </tuple>

     <pdm:person id="s007">
       <rpid:class>mandatory member</rpid:class>
     </pdm:person>

 </presence>
```

Listing 18: Multipart/related body sent by an RLS

Listing 17 shows how the aggregate document could look like. It only includes one `<tuple>` element and indicates that the service is available and supports video. The contact address is set to the group identifier *sip:conference-group@ex.org*. A S-CSCF could route a call to an AS which manages the conference.

In this scenario, it can be useful to know the presence information of particular Presentities. For instance, if the conference leader wants to add certain optional members to a video conference, presence information about these Presentities can come in handy. A Presence Aggregation Server implemented as RLS could provide this. Listing 18 shows an example of a multipart/related document. This document is the body of a NOTIFY message received from a RLS. It includes the RLMI document, the aggregate presence information and the presence information of the particular resources.

6 Summary

The IP Multimedia Subsystem simplifies the deployment of IP-based services in the cellular world. Ongoing IMS releases add interworking with WLAN and support for fixed and cable networks, which brings the IMS closer to the goal of access independence. Apart from new services, cost savings are a strong argument for the IMS.

The presence service for the IMS is more than a simple service; it serves as an enabler for many different services (e.g., Instant Messaging). The presence service provides information about points of reachability for communication and their status, location information, current activities and so forth. Depending on the context, presence service can be perceived either as integral part of the IMS or as a service enabler on top of the IMS. This is the reason why 3GPP as well as OMA have their own, compatible, presence service specifications.

The OMA Presence SIMPLE enabler strongly interacts with the OMA XDM enabler. XDM allows access and manipulation of user-specific service-related information in terms of XML files stored in the network. XCAP is the protocol to access and manipulate XML files and the SUBSCRIBE/NOTIFY mechanism permits subscriptions to changes in XML files. Presence Authorization Rules, Presence Lists, URI Lists and Group Usage Lists are the application usages supported by the OMA Presence SIMPLE enabler. Other enablers can introduce new application usages.

In Section 2.4, we have discussed different data formats for presence information. The least common denominator of all presence data formats is PIDF. Whenever a client cannot understand certain extensions, it understands at least the presence information encoded in PIDF. Both the IMS specifications for Presence and the OMA Presence SIMPLE enabler support several extensions to PIDF and both are open for new extensions. The discussed presence data formats are applicable for a broad range of services.

A RLS helps to save network traffic and reduces the load on terminals. We have proposed a generic RLS design, which utilizes, next to a XCAP server, a HTTP server or database as source of XML files. The presence information of a single resource is either fetched from a Presence Server using SIP subscriptions or queried from a local authority of resources. The modular design allows adding new sources of XML files in a transparent way and extending the possibilities to query the presence information of a single resource. Section 4 has highlighted some implementation details of the RLS prototype, which has been implemented as module for the open-source OpenSER SIP server.

In Section 5, we have proposed several architectures for a Presence Aggregation Server. A Watcher does not perceive any difference between a subscription to aggregate or plain presence information. Apart from a subscription to a RLS, the Watcher does not even know if the presence information is aggregated or not.

Aggregation of presence information has a wide field of application but the existing concepts for aggregation are often limited. The aggregation can become very complex and it is to question whether an aggregation policy can be efficiently modeled in XML.

The question how presence aggregation could be implemented in detail and how to model advanced aggregation rules is left for further studies.

A RLS Module User Guide

A.1 Exported Module Parameter

This section summarizes all possible rls module configuration parameters. Please note that the module is named *rls_rb* in order to avoid conflicts with the existing OpenSER code base.

db_url (str): The database URL. This parameter is compulsory.
Example:
modparam("rls_rb","db_url", "mysql://openser:openserrw@192.168.10.35/openser")

backend_subs_table (str): The name of the database table which stores back-end subscription information. The default is "rls_backend_subscriptions".
Example: modparam("rls_rb","backend_subs_table", "backend_subs")

rl_subs_table (str): The name of the database table which stores information about resource-list subscriptions. The default is "rls_rl_subscriptions".
Example: modparam("rls_rb","rl_subs_table", "rl_subs")

supported_event (str): Adds an event to the list of events which the RLS handles. At least one event type must be supported.
Example: modparam("rls_rb","supported_event", "presence")

xcap_source (str): The source of the rls-service and resource-list definitions. The valid values are "XCAP" (XCAP server), "HTTP" (HTTP Server) and "DB" (database). The default value is "DB".
Example: modparam("rls_rb","xcap_source", "HTTP")

xcap_root (str): The URL pointing to the XCAP source. Depending on the value of the xcap_source parameter the value must be a URL to a XCAP, HTTP or database server. This parameter is compulsory.
Example: modparam("rls_rb","xcap_root", "http://example.org/")

xcap_table (str): The name of the database table which stores XML files. The default value is "xcap".
Example: modparam("rls_rb","xcap_table", "xml_table")

xcap_auth_type (str): The type of authentication used for the HTTP or XCAP server. The valid values are "digest" for HTTP Digest authentication and "simple" for HTTP Simple authentication. The default value is "digest".
Example: modparam("rls_rb","xcap_auth_type", "simple")

xcap_username (str): The authentication user name. The default value is NULL.
Example: modparam("rls_rb","xcap_username", "marlene")

xcap_password (str): The authentication password. The default value is NULL.
Example: modparam("rls_rb","xcap_password", "ab12cd")

retry_interval (int): The time in seconds before an unsuccessful back-end subscription is, at the earliest, retried. The default value is 120.
Example: modparam("rls_rb","retry_interval", 480)

retry_timer_interval (int): The timer interval in seconds before a timer retries all expired back-end subscriptions and back-end subscriptions with an expired retry_interval. The default value is 100.
Example: modparam("rls_rb","retry_timer_interval", 30)

notify_interval (int): The timer interval when the module sends notifications for active resource-list subscriptions (only if presence information has changed). If the value is 0, a NOTIFY is sent immediately with every presence information update of a resource in the list. The default value is 40.
Example: modparam("rls_rb","notify_interval", 0)

resource_list_cache_time (int): The time in seconds how long a resource list should be kept in the cache. The default value is 1800.
Example: modparam("rls_rb","rl_cache_timer", 560)

no_rl_subs_return_code (int): The return code of the function rls_handle_subscribe if no resource list corresponds to the requested URI. The default value is 1.
Example: modparam("rls_rb","no_rl_subs_return_code", 42)

always_reply (int): If this parameter is set to 1, the module sends an appropriate response message to SUBSCRIBE requests, even if no resource list corresponds to the requested URI. The default value is 0.
Example: modparam("rls_rb","always_reply", 1)

max_expires (int): The maximum admissible expires value of resource-list subscriptions. The default value is 3600.
Example: modparam("rls_rb","max_expires", 120)

send_partial_state (int): If this value is set to 1, the module sends notifications with partial state. The default value is 1.
Example: modparam("rls_rb","send_partial_state", 0)

server_address (str): The RLS address which will become the value of Contact header fields. This parameter is obligatory.
Example: modparam("rls_rb","server_address", "sip:ps01.ibkt.tuwien.ac.at:5060")

outbound_proxy (str): The outbound proxy URL to be used when sending back-end subscribe requests. The default value is NULL.
Example: modparam("rls_rb","outbound_proxy", "sip:10.10.0.1")

rl_subs_hash_size (int): The size of the hash table which stores resource-list subscriptions. The default value is 512.
Example: modparam("rls_rb","rl_subs_hash_size", 1024)

rl_hash_size (int): The size of the hash table which stores resource lists. The default value is 512.
Example: modparam("rls_rb","rl_hash_size", 1024)

A.2 Exported Functions

The module exports two functions which can be called from the configuration script.

rls_handle_subscribe(): This function handles incoming SUBSCRIBE requests. If the SUBSCRIBE request targets a resource list, this function stores/updates the dialog information, initiates back-end subscriptions (in the case of a new subscription), sends a reply message and an immediate NOTIFY request. If the parameter "always_reply" is set to 1, this function sends a reply message even if the SUBSCRIBE request is not for a resource list. This function can be used from REQUEST_ROUTE. The return values are:

- 0 if no error has occurred
- -1 if an error has occurred
- the value specified by the parameter "no_rl_subs_return_code" if the SUBSCRIBE request is not for a resource list and the parameter "always_reply" is not set to 1.

rls_handle_notify(): This function handles incoming NOTIFY requests of back-end subscriptions. It stores/updates the information to the database and sends an appropriate reply message. This function can be used from REQUEST_ROUTE. The return values are:

- 0 if no error has occurred.
- -1 if an error has occurred.

A.3 Database Table Definitions

The module depends on two database tables. One stores the dialog information of resource-list subscriptions (*rls_rl_subscriptions*) and a second one (partly) the dialog information of back-end subscriptions (*rls_backend_subscriptions*). A third table (*xcap*) is necessary if the module fetches the rls-service, respectively resource list, definitions from the database.
Tab. 7 outlines the layout of table *rls_rl_subscriptions*. Since the presence module is in charge of caching resource-list subscriptions, the presence module forces this layout.
Tab. 8 shows the layout of table *rls_backend_subscriptions*. The dialog information of back-end subscriptions is partly stored in the table *pua* used by the pua module.
Tab. 9 shows the layout of table *xcap*. The valid values for doc_type are:

- 1 - pres-rules
- 2 - resource-list
- 3 - rls-services

Key	**Type**	**Description**
id	int(10)	Unique ID, primary key
presentity_uri	varchar(128)	Presence URI
to_user	varchar(64)	User part of To header URI
to_domain	varchar(64)	Domain part of To header URI
watcher_username	varchar(64)	The watchers's user name
watcher_domain	varchar(64)	The watchers's domain
expires	int(11)	The time at which the subscription expires; Expires header field value + time()
event	varchar(64)	Event string
event_id	varchar(64)	Event ID parameter value
callid	varchar(64)	Call-ID header field value
to_tag	varchar(64)	Tag value from the To header field
from_tag	varchar(64)	Tag value from the From header field
local_cseq	int(11)	Local CSeq header field value
remote_cseq	int(11)	Remote CSeq header field value
record_route	text	Record-Route header field
socket_info	varchar(64)	Socket information
contact	varchar(64)	Contact header field
local_contact	varchar(64)	Local contact
version	int(11)	Version; used for sending NOTIFYs
status	int(11)	Subscription status
reason	varchar(64)	Reason parameter value

Table 7: rls_rl_subscription table

Key	Type	Description
id	int(10)	Unique ID, primary key
resource_uri	varchar(128)	Resource URI
watcher_uri	varchar(128)	Watcher URI
event	varchar(64)	Event String
supported_mime_types	varchar(255)	Comma separated string of supported MIME types
state	varchar(64)	Subscription status
reason	varchar(64)	Reason parameter value
body	binary	NOTIFY body
content_type	varchar(64)	Content type of NOTIFY body
retry_time	datetime	Point in time when an inactive subscription is retried
expires	datetime	Point in time when an active subscription expires
rl_subs_id	varchar(512)	Dialog ID of corresponding resource-list subscription
dirty	int(1)	1 if updated status has not been reported to the resource-list subscriber yet

Table 8: rls_backend_subscriptions table

Key	Type	Description
id	int(10)	Unique ID, primary key
username	varchar(64)	Owner's user name
domain	varchar(54)	Owner's domain
doc	binary	XML document
doc_type	int(11)	Type of the XML document

Table 9: xcap table

B List of Abbreviations

2G	Second Generation
3G	Third Generation
3GPP	3rd Generation Partnership Project
3GPP2	3rd Generation Partnership Project 2
AoR	Address of Record
AS	Application Server
AUID	Application Unique ID
BGCF	Breakout Gateway Control Function
CAMEL	Customized Applications for Mobile network Enhanced Logic
CIPID	Contact Information for the PIDF
CS	Circuit Switched
CSCF	Call/Session Control Function
DB	Database
DMC	Device Management Client
DMS	Device Management Server
DNS	Domain Name Server
DSL	Digital Subscriber Line
ETSI	European Telecommunications Standards Institute
GGSN	Gateway GPRS Support Node
GPRS	General Packet Radio Service
HSS	Home Subscriber Server
HTTP	Hypertext Transfer Protocol
IANA	Internet Assigned Numbers Authority
I-CSCF	Interrogating-CSCF
IETF	Internet Engineering Task Force
IMPI	IP Multimedia Private Identities
IMPU	IP Multimedia Public Identities
IMS	IP Multimedia Subsystem
IM-SSF	IP Multimedia Service Switching Function

MIME	Multipurpose Internet Mail Extensions
MGCF	Media Gateway Control Function
MGW	Media Gateway
MMD	Multimedia Domain
MRF	Media Resource Function
MRFC	Media Resource Function Controller
MRFP	Media Resource Function Processor
NAI	Network Access Identifier
OMA	Open Mobile Alliance
OMNA	Open Mobile Naming Authority
OSA-SCS	Open Service Access-Service Capability Server
PAL	Presence Aggregation Language
P-CSCF	Proxy-CSCF
PIDF	Presence Information Data Format
PNA	Presence Network Agent
PS	Presence Server
PSTN	Public Switched Telephone Network
PUA	Presence User Agent
QoS	Quality of Service
RADIUS	Remote Authentication Dial In User Service
RFC	Request for Comments
RLMI	Resource List Meta Information
RLS	Resource List Server
RPID	Rich Presence Information Data Format
S-CSCF	Serving-CSCF
SER	SIP Express Router
SGW	Signaling Gateway
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SLF	Subscription Locator Function

TISPAN	Telecoms and Internet converged Services and Protocols for Advanced Networks
TLS	Transport Layer Security
UA	User Agent
UE	User Equipment
UICC	Universal Integrated Circuit Card
UMTS	Universal Mobile Telecommunications System
WLAN	Wireless Local Access Network
XCAP	XML Configuration Access Protocol
XDM	XML Document Management
XDMC	XML Document Management Client
XDMS	XML Document Management Server
XML	eXtensible Markup Language
XUI	XCAP User Identifier

List of Figures

List of Tables

Listings

References

[1] Apache HTTP Server Project. Available at `http://httpd.apache.org`, Last retrieved Jan. 15, 2008.

[2] CounterPath Homepage. Available at `http://www.counterpath.com`, Last retrieved Jan. 15, 2008.

[3] Data sheet Cisco Service Node for Linksys One. Available at `http://www.cisco.com/en/US/products/ps7194/products_data_sheet0900aecd805c3cc1.html`, Last retrieved Jan. 15, 2008.

[4] Libcurl - the multiprotocol file transfer library. Available at `http://curl.haxx.se/libcurl/`, Last retrieved Jan. 15, 2008.

[5] OpenSER. Available at `http://www.openser.org`, Last retrieved Jan. 15, 2008.

[6] SIP for Instant Messaging and Presence Leveraging Extensions (simple) Charter. Available at `http://www.ietf.org/html.charters/simple-charter.html`, Last retrieved Jan. 15, 2008.

[7] The XML C parser and toolkit of Gnome. Available at `http://xmlsoft.org/`, Last retrieved Jan. 15, 2008.

[8] 3GPP. Bootstrapping interface (Ub) and network application function interface (Ua); Protocol details. TS 24.109 V6.9.0, 3rd Generation Partnership Project (3GPP).

[9] 3GPP. Customized Applications for Mobile network Enhanced Logic (CAMEL); Service description; Stage 1. TS 22.078 V6.9.0, 3rd Generation Partnership Project (3GPP).

[10] 3GPP. Generic Authentication Architecture (GAA); Access to network application functions using Hypertext Transfer Protocol over Transport Layer Security (HTTPS). TS 33.222 V6.6.0, 3rd Generation Partnership Project (3GPP).

[11] 3GPP. IP Multimedia Subsystem (IMS); Stage 2. TS 23.228 V6.16.0, 3rd Generation Partnership Project (3GPP).

[12] 3GPP. Network architecture. TS 23.002 V6.10.0, 3rd Generation Partnership Project (3GPP).

[13] 3GPP. Open Service Access (OSA); Stage 2. TS 23.198 V6.0.0, 3rd Generation Partnership Project (3GPP).

[14] 3GPP. Presence service; Architecture and functional description; Stage 2. TR 23.141 V6.9.0, 3rd Generation Partnership Project (3GPP).

[15] 3GPP. Presence service; Security. TS 33.141 V6.2.0, 3rd Generation Partnership Project (3GPP).

[16] 3GPP. Presence service using the IP Multimedia (IM) Core Network (CN) subsystem;Stage 3. TS 24.141 V6.8.0, 3rd Generation Partnership Project (3GPP).

[17] ABI Research. IP Multimedia Subsystem Industry Survey Results, 2005. Available at `http://www.abiresearch.com/whitepaperDL.jsp?id=15`, Last retrieved Jan. 15, 2008.

[18] B. Aboba and M. Beadles. The Network Access Identifier. RFC 2486, Internet Engineering Task Force, Jan. 1999.

[19] V. Beltran, X. Sanchez-Loro, J. Paradells, and J. Casademont. Optimization of Presence Enabled Services over Cellular Networks based on a Personal Proxy. *European Internet and Multimedia Systems and Applications , Proceedings of*, Mar. 2007.

[20] A. Berglund, S. Boag, D. Chamberlin, M. F. Fernández, M. Kay, J. Robie, and J. Siméon. XML Path Language (XPath) 2.0. Technical report, W3C, June 2006. Available at `http://www.w3.org/TR/2006/CR-xpath20-20060608/`, Last retrieved Jan. 15, 2008.

[21] O. Bergmann, J. Ott, and D. Kutscher. A Script-based Approach to Distributed Presence Aggregation. *Wireless Networks, Communications and Mobile Computing, 2005 International Conference on*, pages 1168– 1174 Vol.2, 2005.

[22] G. Camarillo and M. A. Garcia-Martin. *The 3G IP Multimedia Subsystem (IMS). Merging the Internet and the Cellular Worlds.* John Wiley & Sons Ltd, second edition, Dec. 2005.

[23] I. Cooper, I. Melve, and G. Tomlinson. Internet Web Replication and Caching Taxonomy. RFC 3040, Internet Engineering Task Force, Jan. 2001.

[24] M. Day, J. Rosenberg, and H. Sugano. A Model for Presence and Instant Messaging. RFC 2778, Internet Engineering Task Force, Feb. 2000.

[25] T. Dierks and C. Allen. The TLS Protocol Version 1.0. RFC 2246, Internet Engineering Task Force, Jan. 1999.

[26] E. E. Burger. A Mechanism for Content Indirection in Session Initiation Protocol (SIP) Messages. RFC 4483, Internet Engineering Task Force, May 2006.

[27] Ericsson. Introduction to IMS, Mar. 2007. Available at `http://www.ericsson.com/technology/whitepapers/8123_Intro_to_ims_a.pdf`, Last retrieved Jan. 15, 2008.

[28] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, June 1999. Updated by RFC 2817.

[29] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617, Internet Engineering Task Force, June 1999.

[30] J. Galvin, S. Murphy, S. Crocker, and N. Freed. Security Multiparts for MIME: Multipart/Signed and Multipart/Encrypted. RFC 1847, Internet Engineering Task Force, Oct. 1995.

[31] Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518, Internet Engineering Task Force, Feb. 1999.

[32] J. C. Han, S. O. Park, S. G. Kang, and H. H. Lee. A Study on SIP-based Instant Message and Presence. *Advanced Communication Technology, The 9th International Conference on*, pages 1298 – 1301, Feb. 2007.

[33] E. Levinson. The MIME Multipart/Related Content-type. RFC 2387, Internet Engineering Task Force, Aug. 1998.

[34] M. Lonnfors and K. Kiss. Session Initiation Protocol (SIP) User Agent Capability Extension to Presence Information Data Format (PIDF). Internet Draft draft-ietf-simple-prescaps-ext-07, Internet Engineering Task Force, July 2006.

[35] M. Lonnfors, E. Leppanen, H. Khartabil, and J. Urpalainen. Presence Information Data format (PIDF) Extension for Partial Presence. Internet Draft draft-ietf-simple-partial-pidf-format-08, Internet Engineering Task Force, Nov. 2006.

[36] I. Miladinovic. Presence and event notification in UMTS IP multimedia subsystem. *3G Mobile Communication Technologies, 2004. 3G 2004. Fifth IEE International Conference on* , pages 44 – 48, 2004.

[37] I. Morris. IMS Under Scrutiny, Nov. 2006.

[38] A. Niemi. Session Initiation Protocol (SIP) Extension for Event State Publication. RFC 3903, Internet Engineering Task Force, Oct. 2004.

[39] A. Niemi, M. Lonnfors, and E. Leppanen. Publication of Partial Presence Information. Internet Draft draft-lonnfors-simple-partial-publish-06, Internet Engineering Task Force, Feb. 2007.

[40] OMA. Open Mobile Naming Authority. Available at `http://www.openmobilealliance.org/tech/omna/`, Last retrieved Jan. 15, 2008.

[41] OMA. Open Mobile Naming Authority URI-List Usage Name Registry,.

[42] OMA. OMA Device Management. Technical Report v1.1.2, Open Mobile Alliance Ltd. (OMA), Jan. 2004. Available at `http://www.openmobilealliance.org/release_program/dm_v112.html`, Last retrieved Jan. 15, 2008.

[43] OMA. OMA Instant Messaging and Presence Service [IMPS]. Technical Report v1.2.1, Open Mobile Alliance Ltd. (OMA), Oct. 2005. Available at `http://www.openmobilealliance.org/release_program/imps_v1_2_1.html`, Last retrieved Jan. 15, 2008.

[44] OMA. OMA Management Object for XML Document Management. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[45] OMA. OMA Presence SIMPLE. Technical Report v1.0, Open Mobile Alliance Ltd. (OMA), Aug. 2006. Available at `http://www.openmobilealliance.org/release_program/Presence_simple_v1_0.html`, Last retrieved Jan. 15, 2008.

[46] OMA. OMA XML Document Management. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006. Available at `http://www.openmobilealliance.org/release_program/xdm_v1_0_1.html`, Last retrieved Jan. 15, 2008.

[47] OMA. Presence SIMPLE Architecture Document. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[48] OMA. Presence SIMPLE Requirements. Technical Report v1.0, Open Mobile Alliance Ltd. (OMA), July 2006.

[49] OMA. Presence SIMPLE Specification. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[50] OMA. Presence XDM Specification. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[51] OMA. Resource List Server (RLS) XDM Specification. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[52] OMA. Shared XDM Specification. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[53] OMA. XML Document Management Architecture. Technical Report v1.0, Open Mobile Alliance Ltd. (OMA), June 2006.

[54] OMA. XML Document Management (XDM) Specification. Technical Report v1.0.1, Open Mobile Alliance Ltd. (OMA), Nov. 2006.

[55] J. Peterson. A Presence-based GEOPRIV Location Object Format. RFC 4119, Internet Engineering Task Force, Dec. 2005.

[56] J. Peterson and C. Jennings. Enhancements for Authenticated Identity Management in the Session Initiation Protocol (SIP). RFC 4474, Aug. 2006.

[57] D. Petrie. A Framework for Session Initiation Protocol User Agent Profile Delivery. Internet Draft draft-ietf-sipping-config-framework-09, Internet Engineering Task Force, Oct. 2006.

[58] D. Petrie. Extensions to the Session Initiation Protocol (SIP) User Agent Profile Delivery Change Notification Event Package for the Extensible Markup Language Language Configuration Access Protocol (XML). Internet Draft draft-ietf-simple-xcap-00, Internet Engineering Task Force, Oct. 2006.

[59] B. Ramsdell. S/MIME Version 3 Message Specification. RFC 2633, Internet Engineering Task Force, June 1999.

[60] E. Rescorla. HTTP over TLS. RFC 2818, Internet Engineering Task Force, May 2000.

[61] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, June 2002.

[62] A. B. Roach, B. Campbell, and J. Rosenberg. A Session Initiation Protocol (SIP) Event Notification Extension for Resource Lists. RFC 4662, Internet Engineering Task Force, Aug. 2006.

[63] J. Rosenberg. A Presence Event Package for the Session Initiation Protocol (SIP). RFC 3856, Internet Engineering Task Force, Aug. 2004.

[64] J. Rosenberg. A Watcher Information Event Template-Package for the Session Initiation Protocol (SIP). RFC 3857, Internet Engineering Task Force, Aug. 2004.

[65] J. Rosenberg. Extensible Markup Language (XML) Formats for Representing Resource Lists. Internet Draft draft-ietf-simple-xcap-list-usage-05, Internet Engineering Task Force, Aug. 2005. Replaced by RFC 4826.

[66] J. Rosenberg. A Data Model for Presence. RFC 4479, Internet Engineering Task Force, July 2006.

[67] J. Rosenberg. A Processing Model for Presence. Internet Draft draft-rosenberg-simple-presence-processing-model-02, Internet Engineering Task Force, June 2006.

[68] J. Rosenberg. An Extensible Markup Language (XML) Document Format for Indicating A Change in XML Configuration Access Protocol (XCAP) Resources. Internet Draft draft-ietf-simple-xcap-diff-04, Internet Engineering Task Force, Oct. 2006.

[69] J. Rosenberg. Coexistence of P-Asserted-ID and SIP Identity. Internet Draft draft-rosenberg-sip-identity-coexistence-00, Internet Engineering Task Force, June 2006.

[70] J. Rosenberg. Extensible Markup Language (XML) Formats for Representing Resource Lists. RFC 4826, May 2007.

[71] J. Rosenberg. Presence Authorization Rules. Internet Draft draft-ietf-simple-presence-rules-10, Internet Engineering Task Force, July 2007.

[72] J. Rosenberg. The Extensible Markup Language (XML) Configuration Access Protocol (XCAP). RFC 4825, Internet Engineering Task Force, May 2007.

[73] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. RFC 3261, Internet Engineering Task Force, June 2002.

[74] H. Schulzrinne. The tel URI for Telephone Numbers. RFC 3966, Internet Engineering Task Force, Dec. 2004.

[75] H. Schulzrinne. CIPID: Contact Information for the Presence Information Data Format. RFC 4482, Internet Engineering Task Force, July 2006.

[76] H. Schulzrinne. The SIMPLE Presence and Event Architecture. *Communication System Software and Middleware, 2006. Comsware 2006. First International Conference on* , pages 1 – 9, Jan. 2006.

[77] H. Schulzrinne, V. Gurbani, P. Kyzivat, and J. Rosenberg. RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF). RFC 4480, Internet Engineering Task Force, July 2006.

[78] H. Schulzrinne, R. Shacham, W. Kellerer, and S. Thakolsri. Composing Presence Information. Internet Draft draft-schulzrinne-simple-composition-02, Internet Engineering Task Force, June 2006.

[79] H. Schulzrinne, H. Tschofenig, J. Morris, J. Cuellar, J. Polk, and J. Rosenberg. Common Policy: A Document Format for Expressing Privacy Preferences. RFC 4745, Internet Engineering Task Force, Feb. 2007.

[80] R. Shacham, H. Schulzrinne, W. Kellerer, and S. Thakolsri. Composition for Enhanced SIP Presence. *Computers and Communications, 2007. ISCC 2007. IEEE Symposium on* , pages 203 – 210, July 2007.

[81] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, and J. Peterson. Presence Information Data Format (PIDF). RFC 3863, Internet Engineering Task Force, Aug. 2004.

[82] J. Urpalainen. An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors. Internet Draft draft-ietf-simple-xml-patch-ops-02, Internet Engineering Task Force, Mar. 2006.

[83] F. Wegscheider. Minimizing Unnecessary Notification Traffic in the IMS Presence System. *Wireless Pervasive Computing, 2006 1st International Symposium on*, page 6 pp., Jan. 2006.

[84] B. Zhao and C. Liu. Efficient SIP-Specific Event Notification. *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006. ICN/ICONS/MCL 2006. International Conference on*, page 1, Apr. 2006.