D4.2.2

Final Integrated Services Design and Implementation Report

Version 1.0

WP 4.2 Mobile Network Middleware Architecture, Design & Implementation

Dissemination Level: Public

Lead Editor: Per-Oddvar Osland, Telenor

31/10/2005

Status: Final

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Information Society

Grid for complex problem solving

Proposal/Contract no.: 004293

2. Fair Use Rights. Nothing in this license is intended to reduce, limit, or restrict any rights arising from fair use, first sale or other limitations on the exclusive rights of the copyright owner under copyright law or other applicable laws.

3. License Grant. Subject to the terms and conditions of this License, Licensor hereby grants You a worldwide, royalty-free, non-exclusive, perpetual (for the duration of the applicable copyright) license to exercise the rights in the Work as stated below:

a. to reproduce the Work, to incorporate the Work into one or more Collective Works, and to reproduce the Work as incorporated in the Collective Works;
b. to distribute copies or phonorecords of, display publicly, perform publicly, and perform publicly by means of a digital audio transmission the Work including as incorporated in Collective Works.
c. For the avoidance of doubt, where the work is a musical composition:
    i. **Performance Royalties Under Blanket Licenses**. Licensor waives the exclusive right to collect, whether individually or via a performance rights society (e.g. ASCAP, BMI, SESAC), royalties for the public performance or public digital performance (e.g. webcast) of the Work.
    ii. **Mechanical Rights and Statutory Royalties**. Licensor waives the exclusive right to collect, whether individually or via a music rights society or designated agent (e.g. Harry Fox Agency), royalties for any phonorecord You create from the Work ("cover version") and distribute, subject to the compulsory license created by 17 USC Section 115 of the US Copyright Act (or the equivalent in other jurisdictions).
d. **Webcasting Rights and Statutory Royalties**. For the avoidance of doubt, where the Work is a sound recording, Licensor waives the exclusive right to collect, whether individually or via a performance-rights society (e.g. SoundExchange), royalties for the public digital performance (e.g. webcast) of the Work, subject to the compulsory license created by 17 USC Section 114 of the US Copyright Act (or the equivalent in other jurisdictions).

The above rights may be exercised in all media and formats whether now known or hereafter devised. The above rights include the right to make such modifications as are technically necessary to exercise the rights in other media and formats, but otherwise you have no rights to make Derivative Works. All rights not expressly granted by Licensor are hereby reserved.

4. Restrictions. The license granted in Section 3 above is expressly made subject to and limited by the following restrictions:

a. You may distribute, publicly display, publicly perform, or publicly digitally perform the Work only under the terms of this License, and You must include a copy of, or the Uniform Resource Identifier for, this License with every copy or phonorecord of the Work You distribute, publicly display, publicly perform, or publicly digitally perform. You may not offer or impose any terms on the Work that alter or restrict the terms of this License or the recipients' exercise of the rights granted hereunder. You may not sublicense the Work. You must keep intact all notices that refer to this License and to the disclaimer of warranties. You may not distribute, publicly display, publicly perform, or publicly digitally perform the Work with any technological measures that control access or use of the Work in a manner inconsistent with the terms of this License Agreement. The above applies to the Work as incorporated in a Collective Work, but this does not require the Collective Work apart from the Work itself to be made subject to the terms of this License. If You create a Collective Work, upon notice from any Licensor You must, to the extent practicable, remove from the Collective Work any credit as required by clause 4(b), as requested.
b. If you distribute, publicly display, publicly perform, or publicly digitally perform the Work or Collective Works, You must keep intact all copyright notices for the Work and provide, reasonable to the medium or means You are utilizing: (i) the name of the Original Author (or pseudonym, if applicable) if supplied, and/or (ii) if the Original Author and/or Licensor designate another party or parties (e.g. a sponsor institute, publishing entity, journal) for attribution in Licensor's copyright notice, terms of service or by other reasonable means, the name of such party or parties; the title of the Work if supplied; and to the extent reasonably practicable, the Uniform Resource Identifier, if any, that Licensor specifies to be associated with the Work, unless such URI does not refer to the copyright notice or licensing information for the Work. Such credit may be implemented in any reasonable manner; provided, however, that in the case of a Collective Work, at a minimum such credit will appear where any other comparable authorship credit appears and in a manner at least as prominent as such other comparable authorship credit.

5. Representations, Warranties and Disclaimer

UNLESS OTHERWISE MUTUALLY AGREED TO BY THE PARTIES IN WRITING, LICENSOR OFFERS THE WORK AS-IS AND MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND CONCERNING THE MATERIALS, EXPRESS, IMPLIED, STATUTORY OR OTHERWISE, INCLUDING, WITHOUT LIMITATION, WARRANTIES OF TITLE, MERCHANTIBILITY, FITNESS FOR A PARTICULAR PURPOSE, NONINFRINGEMENT, OR THE ABSENCE OF LATENT OR OTHER DEFECTS, ACCURACY, OR THE PRESENCE OF ABSENCE OF ERRORS, WHETHER OR NOT DISCOVERABLE. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO SUCH EXCLUSION MAY NOT APPLY TO YOU.

6. Limitation on Liability. EXCEPT TO THE EXTENT REQUIRED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR BE LIABLE TO YOU ON ANY LEGAL THEORY FOR ANY SPECIAL, INCIDENTAL, CONSEQUENTIAL, PUNITIVE OR EXEMPLARY DAMAGES ARISING OUT OF THIS LICENSE OR THE USE OF THE WORK, EVEN IF LICENSOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

7. Termination

a. This License and the rights granted hereunder will terminate automatically upon any breach by You of the terms of this License. Individuals or entities who have received Collective Works from You under this License, however, will not have their licenses terminated provided such individuals or entities remain in full compliance with those licenses. Sections 1, 2, 5, 6, 7, and 8 will survive any termination of this License.
b. Subject to the above terms and conditions, the license granted here is perpetual (for the duration of the applicable copyright in the Work). Notwithstanding the above, Licensor reserves the right to release the Work under different license terms or to stop

distributing the Work at any time; provided, however that any such election will not serve to withdraw this License (or any other license that has been, or is required to be, granted under the terms of this License), and this License will continue in full force and effect unless terminated as stated above.

8. Miscellaneous

a. Each time You distribute or publicly digitally perform the Work, the Licensor offers to the recipient a license to the Work on the same terms and conditions as the license granted to You under this License.
b. If any provision of this License is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this License, and without further action by the parties to this agreement, such provision shall be reformed to the minimum extent necessary to make such provision valid and enforceable.
c. No term or provision of this License shall be deemed waived and no breach consented to unless such waiver or consent shall be in writing and signed by the party to be charged with such waiver or consent.
d. This License constitutes the entire agreement between the parties with respect to the Work licensed here. There are no understandings, agreements or representations with respect to the Work not specified here. Licensor shall not be bound by any additional provisions that may appear in any communication from You. This License may not be modified without the mutual written agreement of the Licensor and You.

Context

Activity 4	Detailed Architecture, Design and Implementation
WP 4.2	Mobile Network Middleware Architecture, Design and Implementation
Task 4.2.1	Integrated service platform and middleware for NGG
Dependencies	Based on work from WP 3.1, WP 4.1, WP 4.3, WP 4.4 Integration work in WP 5.1 may depend on this deliverable

Contributors:	*Reviewers:*
Dirk Haage (UPM) Vicente Olmedo (UPM) Víctor A. Villagrá (UPM) Jose I. Moreno (UPM) Per-Oddvar Osland (Telenor) Brynjar Viken (Telenor) Gaute Nygreen (Telenor) Fredrik Solsvik (Telenor) David Hausheer (University of Zurich) Cristian Morariu (University of Zurich) Peter Racz (University of Zurich) Patric Mandic (USTUTT) Ruth del Campo (USTUTT) Isabel Alonso (TID) Arantxa Toro (TID)	Internal review by WP4.2 participants. Review by Akogrimo partners external to WP 4.2: Giuseppe Laria (CRMPA) Francesco Verdino (CRMPA) Brian Ritchie (CCLRC) Nuno Inacio (IT Aveiro) Víctor A. Villagrá – as 5.1 WPL (UPM) Alfonso Sánchez-Macián (UPM)

Approved by: QM

Version	Date	Authors	Sections Affected
0.1	6/7/05	Per-O Osland	All (document created)
0.2	12/9/05	Per-O Osland	Section 2 Interfaces Sect 5 Context Manager design
0.3	22/9/05	Peter Racz, Per-O	Included contributions on LSDS and

Version	Date	Authors	Sections Affected
		Osland, Patric Mandic, Brynjar Viken, Cristian Morariu	A4C
0.4	27/9/05	Vicente Olmedo, Víctor A. Villagrá, Patric Mandic, Isabel Alonso, Arantxa Toro, Per-O Osland	Included contributions on SIP and GrSDS
0.5	21/10/05	All	Final version for review. Updated description of components: Interfaces, Design, Implementation. New template. Checked spelling, cross references, etc.
0.6	28/10/05	All	Document updated after review by Akogrimo partners
1.0	31/10/05	Per-O Osland	Status update (Draft -> Final), version update (-> 1.0) after feedback from project management

Executive summary

This document describes the functionality implemented as part of the network middleware layer of the Akogrimo architecture. The network middleware layer provides a set of functions to the upper layers, allowing Akogrimo to present several enhancements to the standard GRID architecture (i.e. OGSA). Specifically, the network middleware layer offers:

- Cross layer A4C (Authentication, Authorization, Accounting, Auditing, and Charging).
- Service registration and discovery for Grid services and local services (close to the end user)
- Presence and context management.

The software implemented and described here is provided by Akogrimo Work Package 4.2, Mobile Network Middleware Architecture, Design and Implementation. The software serves as part of the integrated prototype to be produced in Phase 1 (within PM19) of the project [D5.1.2].

The document has particular focus on the following aspects:

- Requirements
  Functional (and to some extent also non-functional) requirements for the various components have been defined. The purpose is to show the scope of implementation, and to form a basis for testing
- Interface description
  Interfaces are identified, labelled and described to a level of details adequate for developers from other WPs
- Design and implementation
  The objective is to show how the required functionality is designed and implemented. State diagrams and Message sequence charts are widely used
- Testing
  A set of test cases are proposed that verify fulfilment of functional requirements

Table of Contents

List of Figures

List of Tables

Abbreviations

A table with used abbreviations is strongly recommended

Term	Full text
3PCC	3rd Party Call Control
A4C	Authentication, Authorization, Accounting, Auditing and Charging
AA	Authentication and Authorization
AAAC	Authentication, Authorization, Accounting, and Charging
ACE	Adaptive Communication Environment
ADSL	Asynchronous Digital Subscriber Line
Akogrimo	Access To Knowledge through the Grid in a Mobile World
API	Application Programming interface
AR	Access Router
AVP	Attribute-Value-Pair
BVO	Base Virtual Organization
CCPP	Composite Capability/Preference Profiles
CM	Context Manager
CPU	Central Processing Unit
CRMPA	Centro di Ricerca in Matematica Pura ed Applicata
DOW	Description of Work
DSCP	Differentiated Services Code Point
EAP	Extensible Authentication Protocol
EMS	Emergency Medical Service(s)
GrSDS	Grid Service Discovery System
GSDS	General Service Discovery System
IANA	Internet Assigned Numbers Authority
ID	Identifier
IETF	Internet Engineering Task Force
IP	Internet Protocol
ISDN	Integrated Services Digital Network
JNI	Java Native Interface
LSDS	Local Service Discovery System or Life Signs Detection System
MSC	Message Sequence Chart
MT	Mobile Terminal
NAI	Network Access Identifier
NAS	Network Access Server
NGG	Next Generation Grid
NVUP	Network View of the User Profile
OGSA	Open Grid Services Architecture
OS	Operating System
OWL	Ontology Web Language
P2P	Peer to Peer
PA	Presence Agent
PAA	PANA Authentication Agent
PaC	PANA client
PANA	Protocol for carrying Authentication for Network Access
PC	Personal Computer
PIDF	Presence Information Data Format
PS	Presence Server
PUA	Presence UA
QoS	Quality of Service
RDF	Resource Description Framework

Term	Full text
RFC	Request For Comment
RFID	Radio Frequency Identification
RPC	Remote Procedure Call
RTCP	Real Time Control Protocol
RTP	Real Time Protocol
SAML	Security Assertion Markup Language
SD	Service Discovery
SDP	Session Description Protocol (SIP) or Service Discovery Protocol (Bluetooth)
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SLP	Service Location Protocol
SOAP	Simple Object Access Protocol
SP	Service Provider
SQL	Structured Query Language
SSL	Secure Socket Layer
SSO	Single Sign-On
SW	Software
TID	Telefonica I+D
TLS	Transport Layer Security
TN	Telenor
UA	User Agent
UAC/UAS	UA Client / UA Server
UAProf	User Agent Profile
UDDI	Universal Description, Discovery and Integration
UML	Unified Modelling Language
UMTS	Universal Mobile Telecommunications System
UNIZH	University of Zurich
UPM	Universidad Politecnica de Madrid
UpnP	Universal Plug and Play (Microsoft)
URI	Uniform Resource Identifier
USTUTT	University of Stuttgart
UTF	Unicode Transformation Format
VO	Virtual Organization
WAP	Wireless Access Protocol
WLAN	Wireless Local Area Network
WP	Work Package
WS	Web Services
WSDL	Web Services Description Language (XML format for describing network services)
WSMF	Web Service Modeling Framework
WSMO	Web Service Modeling Ontology
WSRF	Web Services Resource Framework
XML	Extensible Markup Language

For descriptions, please refer to www.wikopwdia.org, www.mpirical.com/companion/mpirical_companion.html or www.howstuffworks.com/

Definitions

Authentication

The authentication mechanism defines a process for verifying a user's identity.

Authorization

Authorization is the process of decision on an entity's allowance to perform a particular action or not. The Authorization decision depends on service specific attributes (e.g. service class for QoS service, device requirements) and user-specific attributes (e.g., name, affiliation to a certain group,
age, etc.).

Accounting

Accounting performs two main tasks. The first is collecting data on resource consumption from the services via a metering component while the second task is retrieving stored accounting data whenever this is requested by a legitimate entity.

Auditing

Auditing defines the process of storage and retrieval, when needed, of information on events taking place in the system, history of the service usage, SLA (Service Level Agreement) compliance, and customer charging and tariff schemes applied.

Charging

Charging is the task of calculating the price for a given service consumption based on accounting information. Charging maps technical values in monetary units and then applies a previously established contractual agreement between service provider and service consumer upon a tariff.

Context

Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. In this document context will be user-centric.

Context-aware system

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevance depends on the user's task

Presentity

Provider of SIP presence/context information.

Single Sign-On

Single sign-on (SSO) is the process whereby a single action of user authentication enables to access different administrative domains with no need of re-authentication.

SAML assertion

A SAML assertion is a piece of data produced by a SAML Authority regarding either an act of authentication performed on a user, attribute information about the user, or authorization data applying to the user with respect to a specified resource.

1. Introduction

Scope

The purpose of this document is to describe functionality developed within Akogrimo WP 4.2, Mobile Network Middleware Architecture, Design and Implementation. This includes a listing of components, interface description, design and implementation and suggested test cases. The described functionality will serve as a part of D5.1.2, Integrated prototype. This document is a continuation of the work presented in D4.2.1, Overall Network Middleware Requirements Report [D4.2.1].

Intended audience

This document is primarily intended for

- Partners from WP 4.2 and other WP's involved in design and implementation
- Partners involved in developing Akogrimo Service (e.g. realization of eHealth, eLearning, crisis management, etc)
- Partners involved in system integration

1.1. Akogrimo Network Middleware layer

Akogrimo

The 6th Framework program IP *Akogrimo – Access to Knowledge through the Grid in a mobile World* – aims on the *technology level* at the integration of mobile communication into the Open Grid Services Architecture (OGSA). On the *application level* Akogrimo aims at validating the thesis that the vision of Grid-based computing and the future development of Grid technologies as well as the development of Grid infrastructures can and will substantially draw from the integration of a valid mobility perspective.

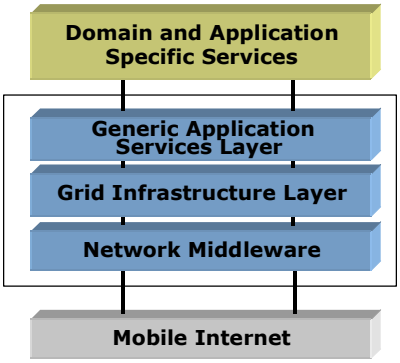


Figure 1, overview of Akogrimo layers.

Figure 1 provides an overview of the Akogrimo architecture. The focus of the Akogrimo project is on the following layers:

- Mobile Internet: This layer focuses on network-related issues such as QoS and resource assignment, mobility, handover, call control and network management.

- The Network Middleware layer: This layer provides a set of basic infrastructure functions to the above layers, including cross-layer A4C, presence and context management, and semantic service discovery. Implementation of functionality in this layer is also the topic of this document.
- The Grid Infrastructure Layer: This layer provides the Grid services/Web services infrastructure (as defined by WSRF/OGSA), execution management (allocating jobs to available resources), data management (replication, fragmentation, unification of heterogeneous storage), and Service Level Agreement (SLA) monitoring and enforcement.
- The Generic Application Services Layer: This layer provides Virtual Organization (VO) management and workflow management (orchestration).

A more detailed view of the Akogrimo architecture is given in Figure 2. The figure indicates responsibilities, functional components, and deployment.

Mobile Network Middleware

The purpose of the Mobile Network Middleware layer is to provide a user-centred and programmer-friendly service platform for Akogrimo offering new opportunities towards the Grid layers. In this document we describe design and implementation for the Mobile Network Middleware layer. This description focuses on the following parts:

- SIP for presence handling
  Provisioning and maintenance of SIP Presence (SIP SIMPLE) for end users
- A4C
  Addressing issues related to Authentication, Authorization, Accounting, Auditing and Charging
- Context Manager
  Collection and distribution of end user context
- Service Discovery (LSDS – Local Service SD and GrSDS – Grid Service DS)
  Registration of and search functionality for local (close to the user) and Grid services

The actual implementation is placed both in the Mobile Network Middleware Architecture (marked with clear blue in Figure 2) and on the Mobile Terminal.

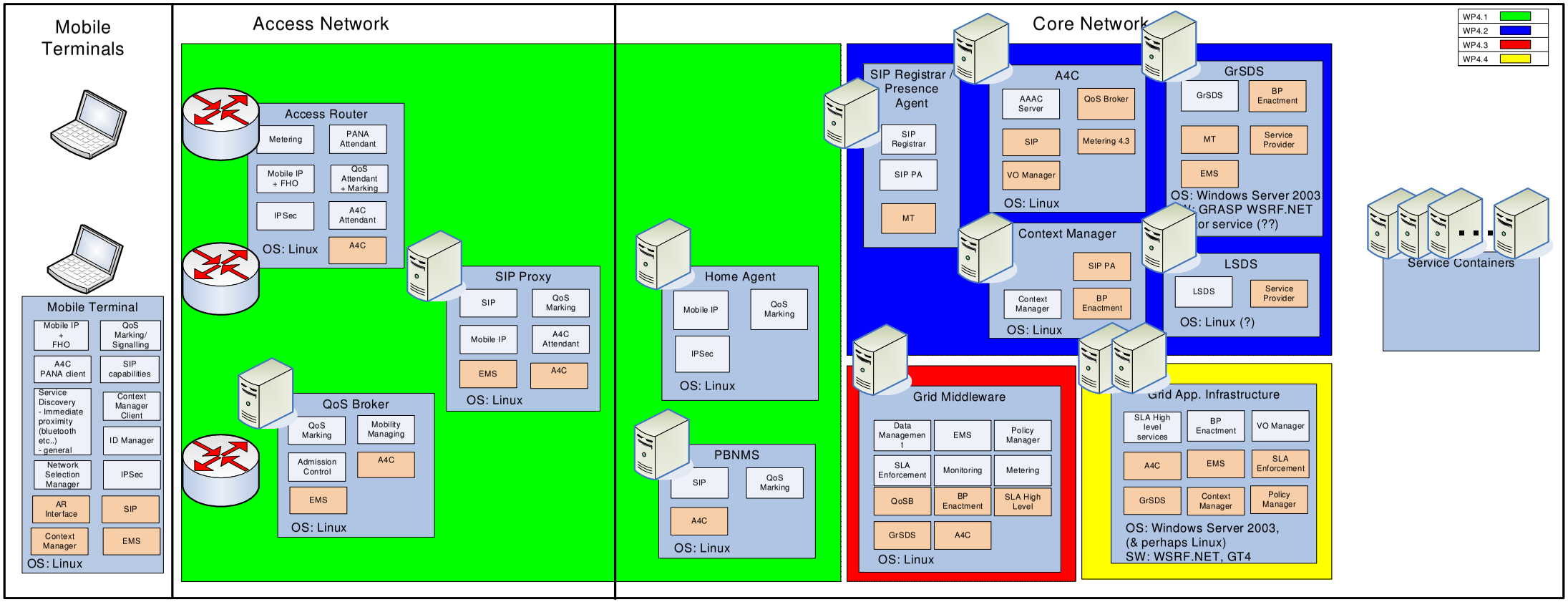


Figure 2 Akogrimo architecture, indicating responsibilities, functional components, and deployment

2. Interfaces

This section identifies interfaces offered by WP 4.2 components:

- SIP Presence
- A4C
- Context Manager
- Service Discovery (Local and Grid SD)

Interface protocols and labels are shown in Figure 3.

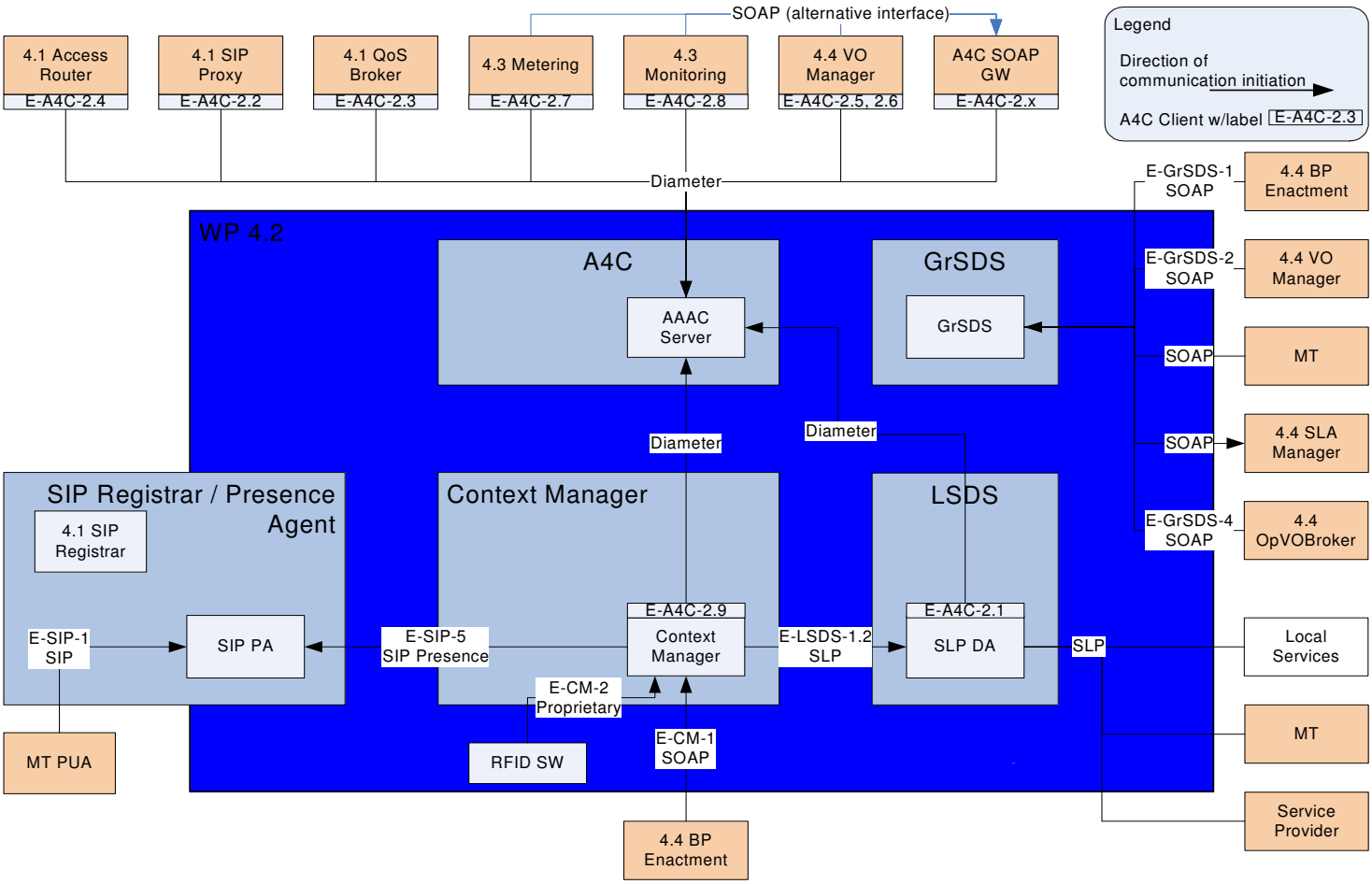


Figure 3 WP4.2 internal and external interfaces

Table 1 SIP Presence Agent – Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
PA WP4.2 [E-SIP-5]	SIP watcher in Context Manager WP4.2	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Presence Event State*	SIP
PA WP4.2 [E-SIP-1]	PUA in MT WP4.2	SIP presence document upload (PUBLISH) *-Input: UserID, Presence Event State* *-Return: OK/Error*	SIP
PA WP4.2 [E-SIP-5]	PUA in MT WP4.2	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Error: 403 "Forbidden" (1st phase), Presence Event State (2nd phase)*	SIP

Table 2 SIP PUA – Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
SIP PA (SIP Server – WP4.2) [E-SIP-1]	SIP PUA (MT – WP4.2)	Publication of user presence and context information (PUBLISH). *-Input: UserID, Presence Data* *-Return: Publication Status*	SIP
SIP PA (SIP Server – WP4.)2 [E-SIP-5]	PUA in MT (MT – WP4.2)	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Error: 403 "Forbidden" (1st phase), Presence Event State (2nd phase)*	SIP
SIP PUA (MT – WP4.2) [E-SIP-7]	CM Client (MT – WP4.2)	Provide the PUA with context information. *-Input: Link to User Agent Profile (String)* *-Return: Operation Result*	Java Interface

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
SIP PUA (MT – WP4.2) [E-SIP-8]	Applications (MT – WP?)	Provide the PUA with user presence information. *-Input: Attributes (undefined for the moment)* *-Return: Operation Result*	Java Interface

Table 3 A4C Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
A4C Server (WP4.2) [I-A4C-1.1] – [I-A4C-1.8]	A4C Client (WP4.2)	Provide A4C tasks	Diameter
A4C Server (WP4.2) [I-A4C-1.1] – [I-A4C-1.8]	A4C Server (WP4.2)	Communicate with other A4C Servers or servers of other domains to perform A4C tasks.	Diameter
A4C Client (WP4.2) [E-A4C-2.4]	Access Router (WP4.1)	Accounting: *-Input: UserID, sessionID, serviceID, accounting data* *-Return: OK/Error*	C++ API
A4C Client (WP4.2) [I-A4C-2.11]	Access Router PAA (WP4.2)	Authentication of users: *-Input: UserID, credentials, serviceID* *-Return: IDToken or Deny*	C++ API
A4C Client (WP4.2) [E-A4C-2.3]	QoS Broker (WP4.1)	QoS authorization and retrieve QoS profile: *-Input: UserID, serviceID* *-Return: QoS profile, (parent sessionID)* (QoS profile is defined in the scope of WP4.1)	C++ API
A4C Client (WP4.2) [E-A4C-2.2]	SIP Server (WP4.1)	AAA in case of SIP registration and invite. Authentication, Authorization: *-Input: UserID, IDToken, serviceID* *-Return: OK/Error, new IDToken*	C++ API
A4C Client (WP4.2) [E-A4C-2.9]	Context Manager (WP4.2)	Retrieve generic user profile: *-Input: UserID* *-Return: generic user profile*	Java API
A4C Client (WP4.2) [E-A4C-2.5], [E-A4C-2.6]	VO Manager (WP4.4)	Authentication based on IDToken: *-Input: UserID, IDToken, serviceID* *-Return: OK/Error, new IDToken* Retrieve generic user profile: *-Input: UserID* *-Return: generic user profile*	Java API
A4C Client	Metering	Accounting:	Java API

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
(WP4.2) [E-A4C-2.7]	(WP4.3)	*-Input: UserID, sessionID, serviceID, accounting data* *-Return: OK/Error*	
A4C Client (WP4.2) [E-A4C-2.8]	Monitoring (WP4.3)	Auditing of service events *-Input: UserID, sessionID, serviceID, eventID, eventData* *-Return: OK/Error*	Java API
A4C Client (WP4.2) [E-A4C-2.x]	A4C SOAP Gateway	Provide a SOAP based interface for A4C tasks.	Java API
A4C SOAP Gateway	Akogrimo Component (WP4.3, WP4.4)	Access A4C services.	SOAP
A4C Client (WP4.2) [E-A4C-2.1]	LSDS (WP4.2)	Authorize User or SP	C++ API

Table 4 Context Manager (CM) – interfaces

Server: Component (Layer) [Label]	Client: Component (Layer)	Purpose	Protocol
A4C Client (WP4.2) [E-A4C-2.9]	CM	- Obtain static user data (SIP address, RFID, …) - Authorise	Java API
CM (WP 4.2) [E-CM-1]	BP Enactment (WP4.4)	- Queries for specified context data - Specify domain specific context extensions - Subscribe to notifications about changes in context info	OWL over SOAP
SIP PA (WP 4.2)	CM (WP4.2)	- Obtain user presence	SIP SIMPLE
LSDS (WP4.2) [E-LSDS-1]	CM (WP4.2)	- Search of Local Services (capabilities, location)	SLP
CM (WP4.2) [E-CM-2]	RFID SW (WP4.2)	- Send position readings for RFID tags	Proprietary

Figure 4 Grid Service Discovery Server (GrSDS) – Interfaces

Interface	Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
E-GrSDS-1	GrSDS	BPE (WP4.4)	Search and Fetch a Grid Service.	SOAP
E-GrSDS-2	GrSDS	BVO Manager (WP4.4)	Publish/Deregister Service	WSDL/SOAP
	SLA Manager (WP4.4)	GrSDS	GrSDS interacts with the SLA-Access to retireve the SLA from published SLA-Template and associate them to the interface of service published in GrSDS	SOAP
E-GrSDS-4	GrSDS	OpVOBroker (WP4.4)	Look for services to carry out a certain WF	SOAP

Table 5 Local Service Discovery Server (SDS) – Interfaces

Interface label	Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
E-LSDS-1.2	DA	CM	Search of Local Services for user Context	SLP
[E-A4C-2.1]	A4C Client	DA	Authorize User or SP	C++ API / Diameter

3. SIP Presence handling

One of the chosen mechanisms to provide context information in Akogrimo is based on SIP. The SIP presence/context framework, as part of the Akogrimo context management infrastructures, will provide information regarding user terminal capabilities, user availability and preferences regarding how to contact him/her, etc... This information is gathered by the Context Manager and offered to the corresponding services which demand it.

3.1. Requirements

RFC 3856 [Ros04] defines how SIP is able to provide presence/context information and defines the needed components. This IETF Request for Comments represents a concrete instantiation (for presence-related events) of the general event notification framework defined in RFC 3265 [Roa02].

From the several alternatives we described in [D4.2.1] (i.e. regarding where to locate the different components, the mechanism to upload presence documents), we have decided the following configuration for Akogrimo:

- A Presence User Agent (PUA) included in the MT SIP component. This element will use the powerful PUBLISH method described in RFC 3903 [Nie04] as the publication mechanism. The MT will not implement the presence watcher interface (at least in phase one), so SIP presence/context information will be only available for services, which will have to request for it to the Context Manager.
- A Presence Agent (PA) located in the SIP Server, which handles subscriptions and publications, and generates the corresponding notifications when the status of a presentity changes. The co-location with the rest of the network infrastructures, especially with the Registrar Server, enables the usage of software APIs instead of having additional physical interfaces with other SIP components. Apart from that, it takes advantage on the SIP handling capabilities framework that the chosen implementation of the SIP Server natively offers.
- The last decision imposes that the Context Manager will implement a SIP watcher interface (a SIP User Agent with SUBSCRIBE and NOTIFY support) to get SIP-based presence/context information from the Presence Agent.

Next subsections describe the functional requirements of both the PA and the PUA. The SIP watcher component embedded on the Context Manager will be described in Section 5.3.3.

3.1.1. SIP Presence Agent

SIP Presence Agent has been build as a dynamic library embedded on SIP Server, which will be implemented starting from the open source SIP Express Router and will run in a Linux/Ubuntu machine. Implementation details are described in section 3.4.1.

The following table lists the functionalities that will be implemented in each phase of the project. The main reason that explains the criteria we have adopted is to have an operational component in Phase 1 covering the most relevant functionalities, and leave for the second phase the most advances features.

Table 6 SIP PA – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**Presence/Context document handling**	**X**	**X**
F 1.1	PUBLISH method support	X	
F 1.2	PIDF format support	X	
F 1.3	RPID format support	X	
F 1.4	Default event state (when no info. from the user)	X	
F 1.5	Event State Compositor (aggregation from different sources)		X
F 1.6	State deltas support (users provide only the information that changed, but the PA maintains the complete state)		X
F 1.7	Permanent storage (database support)		(X)
F 2	**Subscriptions handling**	**X**	**X**
F 2.1	SUBSCRIBE method support	X	
F 2.2	NOTIFY method support	X	
F 2.3	Basic Presence Authorisation policy (only subscription from Context Manager must be accepted in phase 1)	X	
F 2.4	Full Presence Authorisation policies		X
F3	**Generic**	**X**	**X**
F 3.1	Fully RFC compliance (all possible response codes for all possible “abnormal” situations)		X

(X) Optional requirement

Next table shows the most relevant non functional requirements of the Presence Agent.

Table 7 SIP PA – Non-functional requirements

Req. nr	Description	Phase 1	Phase 2

Req. nr	Description	Phase 1	Phase 2
N 1	Running on Linux (Ubuntu) on a PC platform	X	
N 2	Performance improvement (efficiency, speed)		X

3.1.2. SIP Presence User Agent

The SIP PUA will run in the Mobile Terminal. As it is Java based, it will run in Java Virtual Machine (JVM).

In the next table, F1 refers to the ability of the SIP PUA to get presence data in a standardized format (PIDF/RPID, [Sug04] and [Schu05]) coming from the Context Manager Client and the Applications, merge this information into a single document and send it to the SIP PA which is running in the SIP Server.

F2 refers to phase 2 functionality that will allow a user to subscribe to other users' presence data via the SIP PA.

Table 8 SIP PUA – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**Presence/Context document handling**	**X**	**X**
F 1.1	PUBLISH method support	X	
F 1.2	PIDF format support	X	
F 1.3	RPID format support		X
F 1.4	Event State Composition (aggregation from different sources)	X	
F 1.5	Publication expiration handling		X
F 2	**Subscriptions handling**	**X**	**X**
F 2.1	SUBSCRIBE method support (sender)		X
F 2.2	NOTIFY method support (receiver)		X

Table 9 SIP PUA – Non-functional requirements

Req. nr	Description	Phase 1	Phase 2
N 1	Running on Linux (Ubuntu) on a PC platform	X	
N 2	Running on PDA platform.		X (?)

3.2. Interfaces

3.2.1. SIP Presence Agent

The Presence Agent, as part of the SIP server, interacts with the rest of the SIP Server components through the corresponding APIs. In fact, it will offer a couple of functions that the Broker Engine will invoke when it receives a PUBLISH or SUBSCRIBE request (see section 3.4.1.1). The Broker Engine exposes to the PA some functions from the SIP proxy for sending outgoing responses and NOTIFY messages. Another API for persistent publication/subscription storage is offered to the PA by the Presence/Context Database.

The Presence Agent implements two external SIP-standard interfaces, a PUA interface with the MT and a watcher interface with the Context Manager. In the second phase, the interface with the MT will be extended, adding also mentioned presence watcher interface.

Table 10 SIP PA – Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
PA WP4.2 [I-SIP-1.7.1]	Broker Engine WP4.2	Handling of incoming PUBLISH and SUBSCRIBE requests *-Input: incoming message* *-Return: OK/Error*	C API
Presence/Context Database WP4.2 [I-SIP-1.6.2]	PA WP4.2	Handling (storage, retrieval…) of (persistent) publications and subscriptions.	C API
SIP proxy WP 4.2 [I-SIP-1.1.1]	Broker Engine	Handling replies and notifications (SIP NOTIFY) to the MT. This interface is exposed directly by the Broker Engine to the PA.	C API
PA WP4.2 [E-SIP-5]	SIP watcher in Context Manager	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods)	SIP

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
	WP4.2	*-Input: UserID (AoR, default SIP URI)* *-Return: Presence Event State*	
PA WP4.2 [E-SIP-1]	PUA in MT WP4.2	SIP presence document upload (PUBLISH) *-Input: UserID, Presence Event State* *-Return: OK/Error*	SIP
PA WP4.2 [E-SIP-5]	PUA in MT WP4.2	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Error: 403 "Forbidden" (1st phase), Presence Event State (2nd phase)*	SIP

Next MSCs provides a clearer explanation on how this SIP interfaces works. Note that only relevant actors (MT, SIP Server and Context Manager) and interactions from the point of view of these SIP interfaces have been depicted. This means, for example, that we assume that the Context Manager have been requested from the grid layers to get the context information of certain user; and we assume also that the Context Manager has obtained the default AoR (default SIP URI of the user) from the A4C. These necessary interactions are part of the general process of getting context information and are described in detail in the Context Manager section of this document, so have not been included in the following figures.

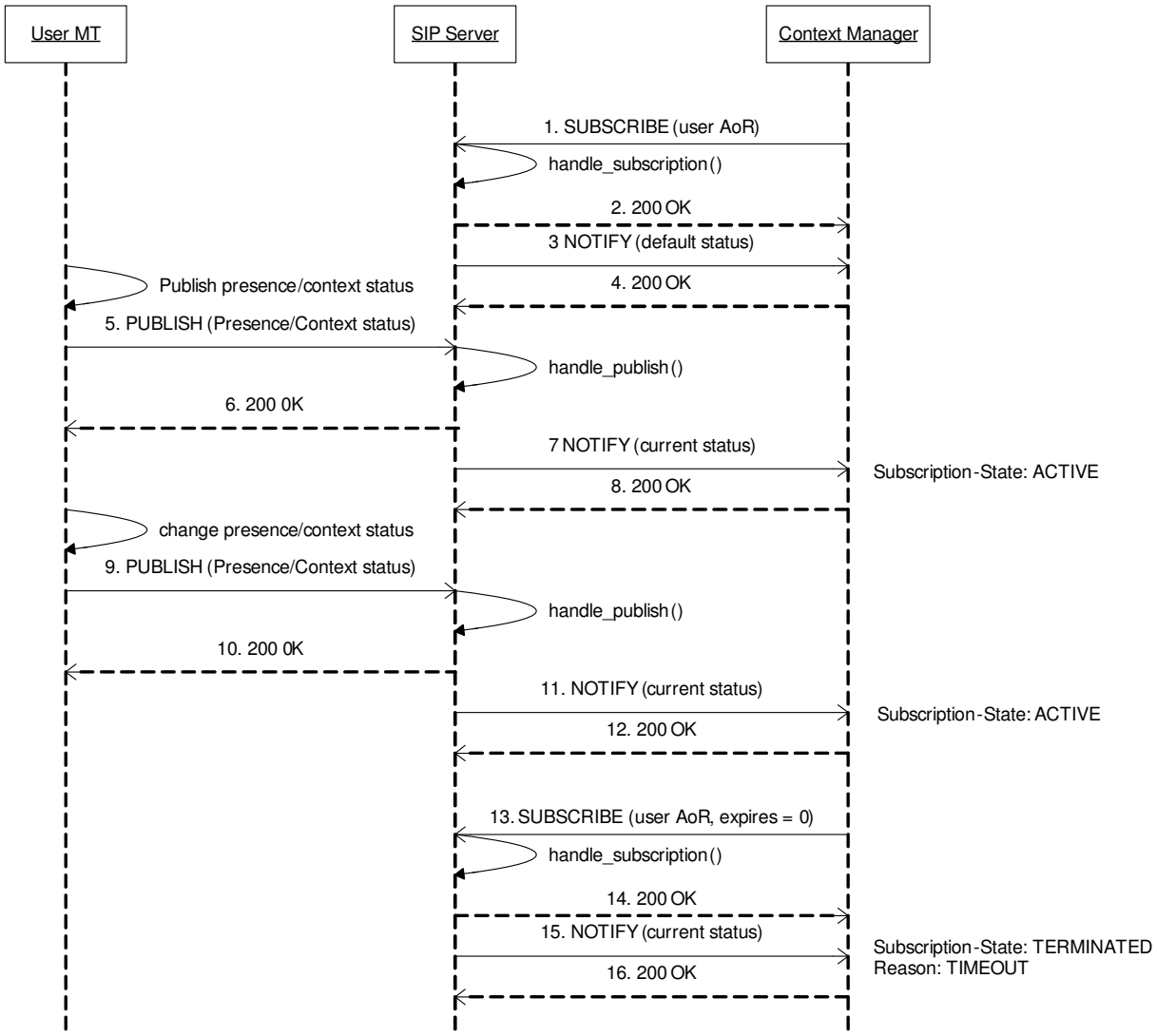


Figure 5 Presence Agent external interfaces: publications and subscriptions

Messages from 1 to 4 describe a typical successful subscription. The Context Manager (CM) sends a SUBSCRIBE message to the Presence Agent (PA) indicating we wants to know the presence/context status of certain user (1, 2). The PA does not have information about the user, so it sends a NOTIFY (3, 4) containing the default user state and creates the subscription. When the user publishes his/her presence status (5, 6), the PA detects there is an active subscription and informs the CM accordingly (7, 8). Messages from 9 to 12 show how the CM is informed about a change on the user presence/context status. Finally, the CM performs an unsubscription (from 13 to 16).

Other situations are described in the next figure

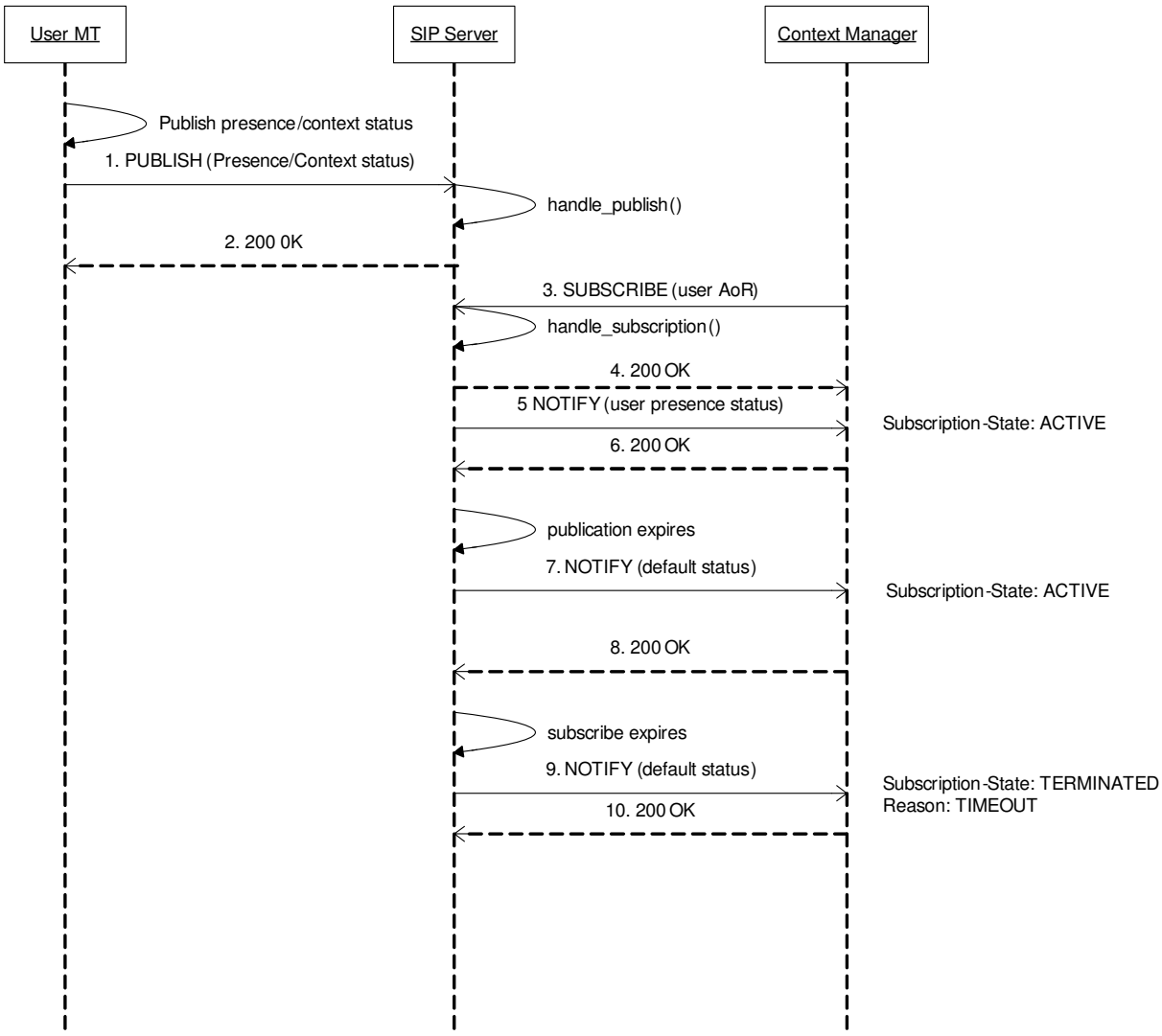


Figure 6 Presence Agent external interfaces: publications and subscriptions (2)

In this case, the user publishes his/her presence/context information (1, 2) prior to the subscription from the CM (3, 4), which of course is notified accordingly (5, 6). If the publication is not refreshed and expires, the CM receives a NOTIFY message indicating that the presentity document previously stored is no longer valid, so the default status is sent in the message body (7). Finally, the CM is also informed if the subscription expires (9, 10). In this case, the NOTIFY message includes a "Subscription-State" header field set to "TERMINATED" and a "reason" parameter set to "TIMEOUT".

3.2.2. SIP Presence User Agent

The SIP PUA, residing in the user's Mobile Terminal, provides simple logic to allow applications and the Context Manager Client to publish desired presence/context information about the user in the Akogrimo's SIP infrastructure.

Thus, the SIP PUA has three interfaces which allow this communication. It provides two internal Java interfaces, one to the CM Client and one to the applications. Although it was first decided that CM Client and the applications would provide complete PIDF XML documents, the interfaces have been redesigned to keep PIDF XML construction mechanisms centralized. This will ease the work on CM Client and the applications, since

there is no need to create any XML document now, but just to provide the attributes to be published. For now, in order to keep things simple, a method for each attribute will be provided. This approach does not lead to a scalable interface, since the addition of attributes would require the addition of new methods to the interface, so a redesign to provide a generic attribute publication interface will be done in phase 2.

Finally, the SIP PUA provides a third interface to the SIP PA in the SIP Server. This interface is a common SIP interface based on the PUBLISH method defined in RFC 3903 that will be accomplished using the services provided by the SIP UA infrastructure in the MT.

Table 11 SIP PUA – Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
SIP UA (MT – WP4.1) [I-SIP-3.1]	SIP PUA (MT – WP4.2)	Sending and reception of SIP messages.	Java Interface
SIP PA (MT – WP4.2) [I-SIP-3.2]	Presence/Context Description Module (MT – WP4.2)	Conversion from application and CM Client provided attributes to PIDF/RPID XML and vice versa.	Java Interface
SIP PUA (MT – WP4.2) [I-SIP-3.3]	CM Client IF Module (MT – WP4.2)	SIP PUA interface adaptation for CM Client (if needed).	Java Interface
SIP PUA (MT – WP4.2) [I-SIP-3.4]	App IF Module (MT – WP4.2)	SIP PUA interface adaptation for applications (if needed).	Java Interface
SIP PA (SIP Server – WP4.2) [E-SIP-1]	SIP PUA (MT – WP4.2)	Publication of user presence and context information (PUBLISH). *-Input: UserID, Presence Data* *-Return: Publication Status*	SIP
SIP PA (SIP Server – WP4.)2 [E-SIP-5]	PUA in MT (MT – WP4.2)	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Error: 403 "Forbidden" (1st phase), Presence Event State (2nd phase)*	SIP
SIP PUA (MT – WP4.2) [E-SIP-7]	CM Client (MT – WP4.2)	Provide the PUA with context information. *-Input: Link to User Agent Profile (String)*	Java Interface

		-Return: Operation Result	
SIP PUA (MT – WP4.2) [E-SIP-8]	Applications (MT – WP?)	Provide the PUA with user presence information. *-Input: Attributes (undefined for the moment)* *-Return: Operation Result*	Java Interface

The interactions between the SIP PUA, the SIP UA, the CM Client and future applications is described in the next MSC.

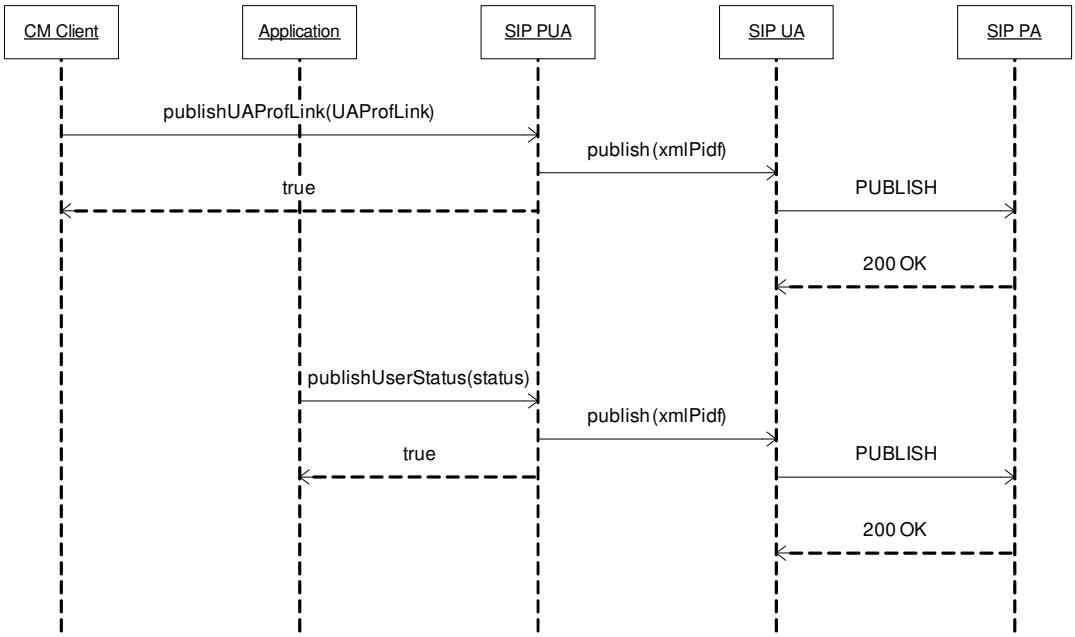


Figure 7 Use of the different SIP PUA interfaces

As can be seen in the diagram, the SIP UA does not inform about the success of the publication operation. Adding this functionality may be considered if needed.

At the moment, user status attributes remain undefined, since they depend on the application. However, several attributes are expected, so a generic interface is desirable.

3.3. Design

3.3.1. SIP Presence Agent

As mentioned earlier, Presence Agent is part of the SIP Server. The following figure shows the functional description of the whole component, with special focus on the subset of components and interfaces directly involved in presence/context handling (the rest of them have been blurred in the figure). Note that for simplicity, the WS-App IF and the Routing IF subcomponents have been considered as part of the proxy submodule; in fact, the SIP server only will act as a proxy in both cases.

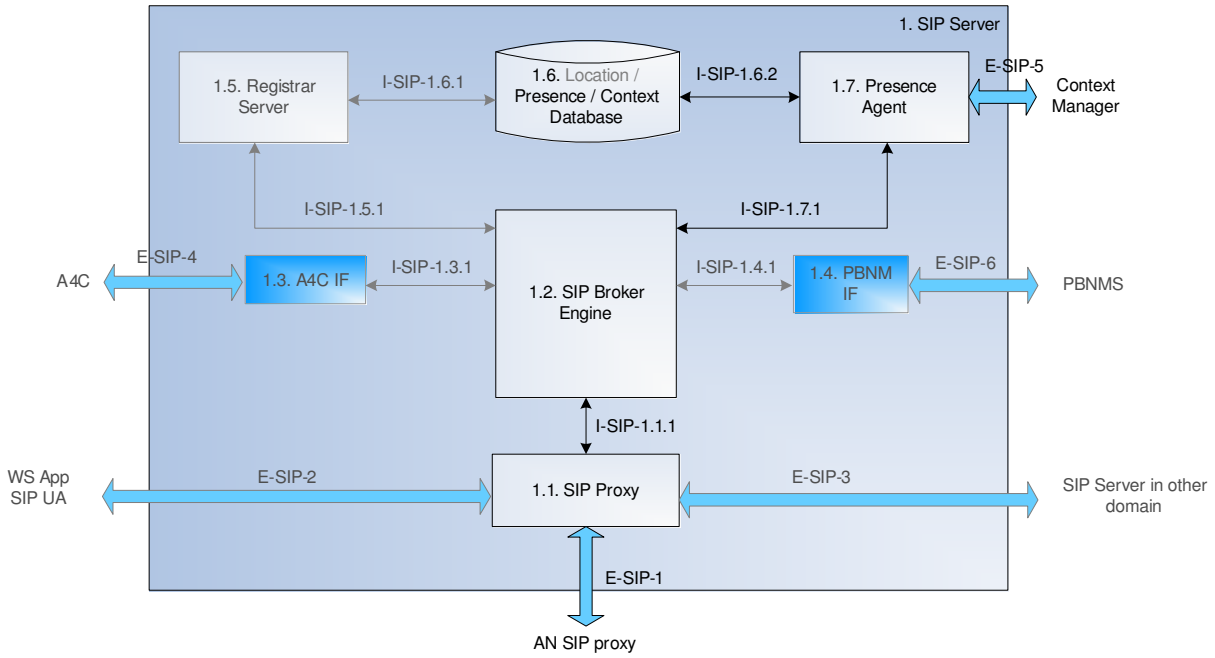


Figure 8 . Akogrimo SIP Server functional description

The SIP Presence Agent provides SIP-related presence/context information delivered by the users using the SIP method PUBLISH to the Context Manager through a presence watcher interface (SIP SUBSCRIBE and NOTIFY). The presence/context status is stored in a database, which can be associated to the Location Database if we consider that the information it stores is part of the user context. In fact, the registration information could be useful to provide a "default user context" when there is no more information available. For example, if the presence information of a non-registered user is requested, the PA could send and empty or neutral body to the Context Manager to indicate this unavailability.

Location Database and Presence Agent are both exclusive WP4.2 components; however, other SIP Server components, like the Broker Engine or the SIP embedded SIP proxy, play some roles in the WP42 interactions

The SIP Broker Engine is the core part of the SIP Server. It implements the interconnection logic between the different SIP entities as well as the interfaces with other network entities. Acting as the "brain" of the SIP Server, it takes decisions based on the nature and the content of all incoming requests, previously pre-processed by the SIP proxy. This means that the SIP proxy acts a listener of the available SIP ports of the SIP Server (typically the 5060 port) and first it decides if the SIP Server should process the corresponding incoming message. If yes, it passes it to the Broker Engine, which depending on the receive message, routes it to the corresponding subcomponent (i.e. REGISTER messages to Registrar Server, PUBLISH messages to Presence Agent, incoming INVITE to the A4C IF submodule for first authentication…). From the perspective of the SIP Presence handling, both the Broker Engine and the embedded SIP proxy should include some logic to deliver incoming SUBSCRIBE and PUBLISH requests to the PA module.

3.3.2. SIP Presence User Agent

The SIP Presence User Agent is part of the SIP infrastructure in the MT. The next figure shows the functional entities that provide needed logic and interfaces.

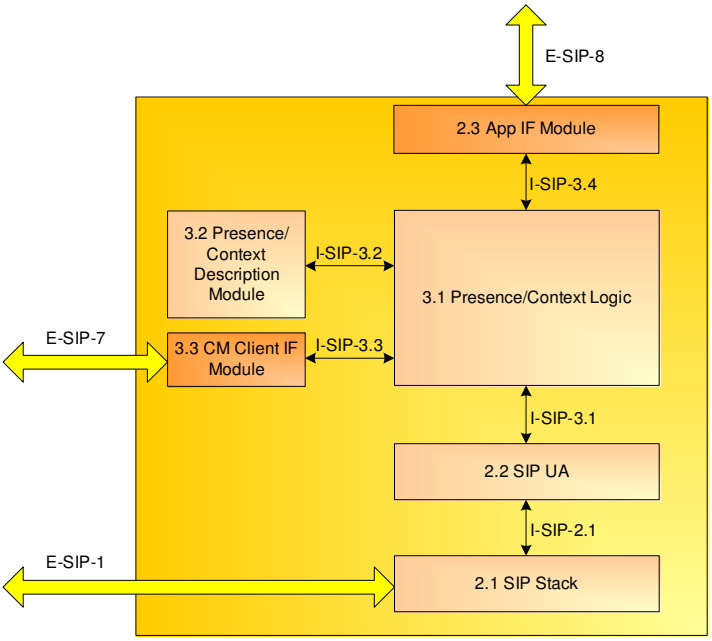


Figure 9 SIP Presence User Agent design in the MT

As can be seen in the figure, the SIP PUA is composed by four main components:

- **Presence/Context Logic:** This is the central component. It gathers presence and context data from the CM Client and the applications. From this data, a PIDF XML document will be created and sent via the SIP UA.
- **Presence/Context Description Module:** This module is targeted at the creation of presence documents in various formats. It encapsulates XML creation functionality and future XML parsing capabilities.
- **Application Interface Module:** This module will be used to provide an appropriate interface to the application in case the Java interface is not enough or any transformation is needed. As applications are to be decided, so it is this interface.
- **Context Interface Module:** If any adaptation of the Java interface provided by the Presence/Context Logic to the CM Client is to be done, it will be done in this component. For the moment, the Java interface is enough, so this component will not be materialized in implementation.

3.4. Implementation

3.4.1. Presence Agent Implementation

The whole SIP Server will be build based on SIP Express Router (SER) implementation [SER], which is a powerful, easily configurable, C-written and free SIP server implementation. It can act as a proxy, registrar or redirect server. Most relevant characteristics of SER are:

- Speed: SER is able to process thousands of calls per second even on low-cost platforms.
- Extensibility: using SER it is possible to link new C code to redefine, or extend, current logic. New code, which can be developed independently on SER core, is linked to it in run time. In this way, the best way to proceed is to add new functionality as SER modules (*.so libraries) keeping the SER core really fast and compact.
- Flexibility: administrators may write textual scripts, which determine routing decisions, SIP method processing...By modifying the main script (named ser.cfg) administrators can control and modify lot of parameters and introduce new logic (loading new modules, and invoking exported functions when necessary).
- Portability: written in ANSI C, and deeply tested on PC/Linux, Sun/Solaris platforms.
- Interoperability: SER is based in open SIP standards (SER core implements basically RFC 3261).
- Small size: core footprint is 300K, add-on modules take up to 630K.

SER already includes a Presence Agent module, but it is an experimental version and seems not to work properly, so a new one was developed from scratch. Next sections describe the main features of the library we have developed. As mentioned earlier, SER is written in C, so the design and implementation can not be object oriented; however, we will describe most significant C structs we have defined.

SER has been built to be configured using a configuration file, ser.cfg. It is a C-Shell script which defines how incoming SIP requests should be processed. During the server startup, this file is translated into an internal binary representation (basically because the direct interpretation of this file would make SER very slow). Considering the functional description provided in section 3.3.1, the main configuration file and the main part of the SER core implements the so called "SIP Proxy" and the "SIP Broker Engine" modules. We can modify that file in order to deliver incoming SIP request to the specific module that implements a more concrete functionality, like the Registrar or the A4C interface.

The Akogrimo PA has been built as a SER module (*.so dynamic library), that we will describe in next section. It implements the Presence Agent itself.

3.4.1.1. Akogrimo Presence Agent SER module

The library that implements the Akogrimo Presence Agent functionality is named esc.so (ESC: event state compositor, which is a generalisation of the Presence Agent for any kind

of event as defined in RFC 3903 [Nie04]). The basic functionality of this module is to maintain a list of valid presence/context documents from users and the corresponding subscriptions from the Context Manager, which must be notified if the presence/context status of an observed user changes. So the library must provide functions to process incoming PUBLISH and SUBSCRIBE SIP methods, in parallel with the maintenance of some data structs to store both subscriptions and publications.

This library exports two main methods, named handle_publish() and handle_subscription(), responsible for processing incoming PUBLISH or SUBSCRIBE methods, respectively. This implies that the following lines should be added to the ser.cfg configuration file:

```
if (method = PUBLISH)
{
      # Next if statement is needed if module involves transaction creation
      if (!t_newtran())
      {
            log(1, "newtran error\n");
            sl_reply_error();
      }
      handle_publish();
      break;
}

if (method = SUBSCRIBE)
{
# Next if statement is needed if module involves transaction creation
if (!t_newtran())
{
      log(1, "newtran error\n");
      sl_reply_error();
}
      handle_subscription();
      break;
}
```

Next table summarises exported functions.

Table 12 esc.so exported functions

Function	Description	Parameters
handle_publish	This function handles incoming PUBLISH request	None when invoked from ser.cfg file (1)
handle_subscription	This function handles incoming SUBSCRIBE request	None when invoked from ser.cfg file (1)

(1) The SER framework passes implicitly the incoming SIP message structure to the function.

The library also offers several configurable parameters to control some default or maximum values such the maximum permissible values for the expired fields of incoming PUBLISH or SUBSCRIBE requests. This implies that, after module initialisation, the following line – one line per parameter – should be added to the ser.cfg config file:

```
mod_param("esc", "parameter_name", value);
```

Next tables summarises the proposal of available library parameters:

Table 13 esc.so configurable parameters

Name	Description
trace_level	For debugging purposes a traces mechanism (independent from the SER traces framework) has been included. 5 levels of traces (from lower to higher level detail) have been included. This allows to debug the library without "interferences" from the rest of the SER code, if we configure the SER debug level to its minimum level of detail. Default is 0 (minimum).
check_expires_timer_interval	The library includes a timer which periodically checks the validity of both publications and subscriptions. Timeout is configurable using this parameter (default is 60 seconds).
max_subscription_expired_value	Maximum subscription validity. If the "expires" field of an incoming subscription is greater than this value, it is automatically adjusted to this figure. It is also the default value if no "expires" field is present.
max_publish_expired_value	Maximum publication validity. If the "expires" field of an incoming presence publication is greater than this value, it is automatically adjusted to this figure. It is also the default value if no "expires" field is present

To maintain presence documents and subscriptions, two C structs have been defined, the subscriber node (which contains all relevant information associated to a subscription) and the EPA node (EPA: event publication agent, a generalisation of the concept presentity, valid for any kind of events). The module maintains in shared memory a double linked list of EPA nodes, containing the presence/context document of the users who have published their presence/context status. Each EPA node, apart from the necessary presentity information (URI, pidf document, expiration time…) has associated a linked list of watchers with active subscriptions to its presence/context information. Note that for the first phase only subscriptions from Context Manager will be accepted, but in order to consider multiple watchers in the future we have designed a data model that supports it.

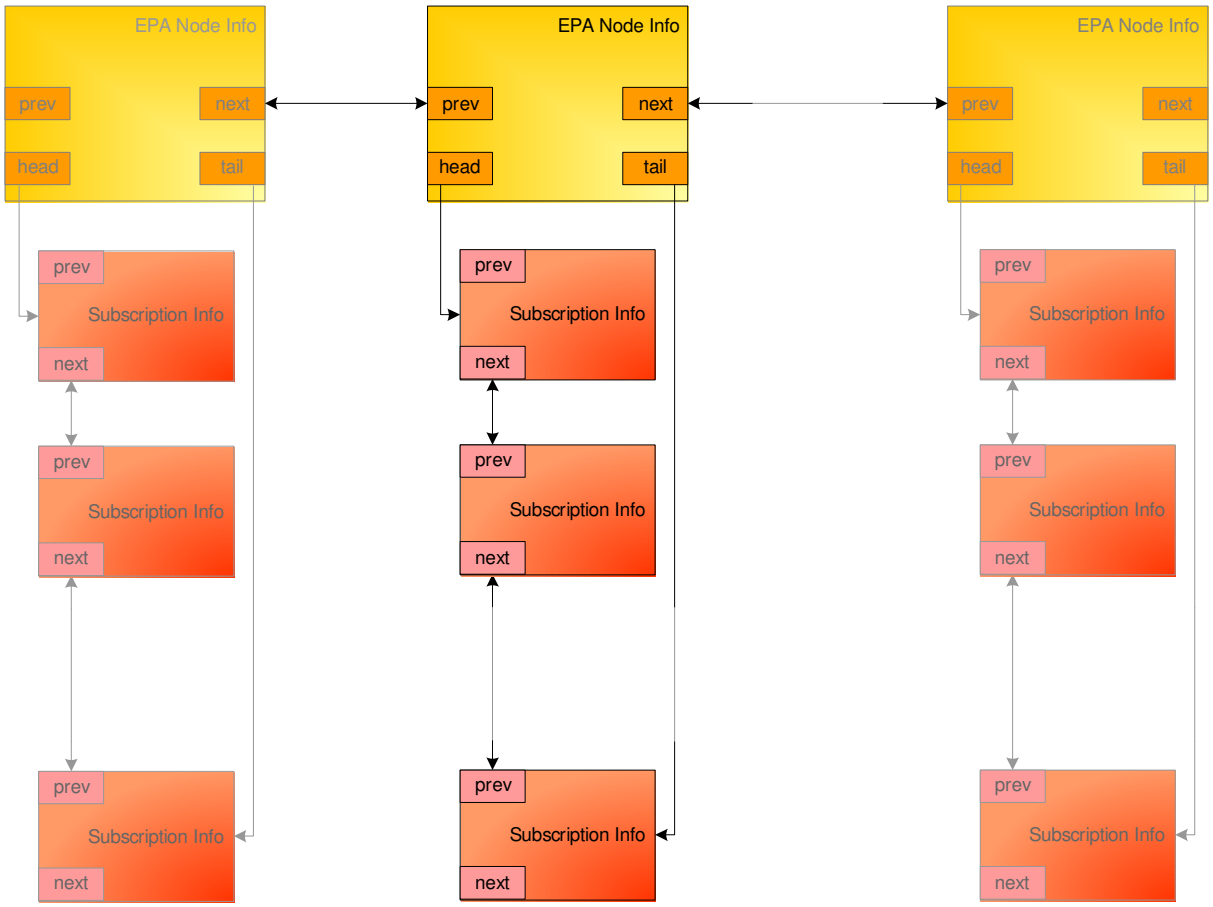


Figure 10 Publications and subscription data model

Periodically, a timer checks the validity of both publications and subscriptions, comparing the current time with the "expires" field of these structures. Expired subscriptions are simply removed (the corresponding NOTIFY messages are sent before, as the RFC 3265 indicates). The same applies for expired publications without active subscriptions; however, if there are still active subscriptions when the timeout reaches, the EPA node is not removed; only its presence document is set to the neutral state.

The timer timeout can be set through the parameter "check_expires_timer_interval".

Apart from the timer, incoming messages PUBLISH and SUBSCRIBE also modify this data, as described in the following sections.

3.4.1.2. Processing PUBLISH requests

The method handle_publish() is intended to provide full PUBLISH message processing support. Following the algorithm described in RFC 3903 [Nie04], the main algorithm is depicted in the following figure.

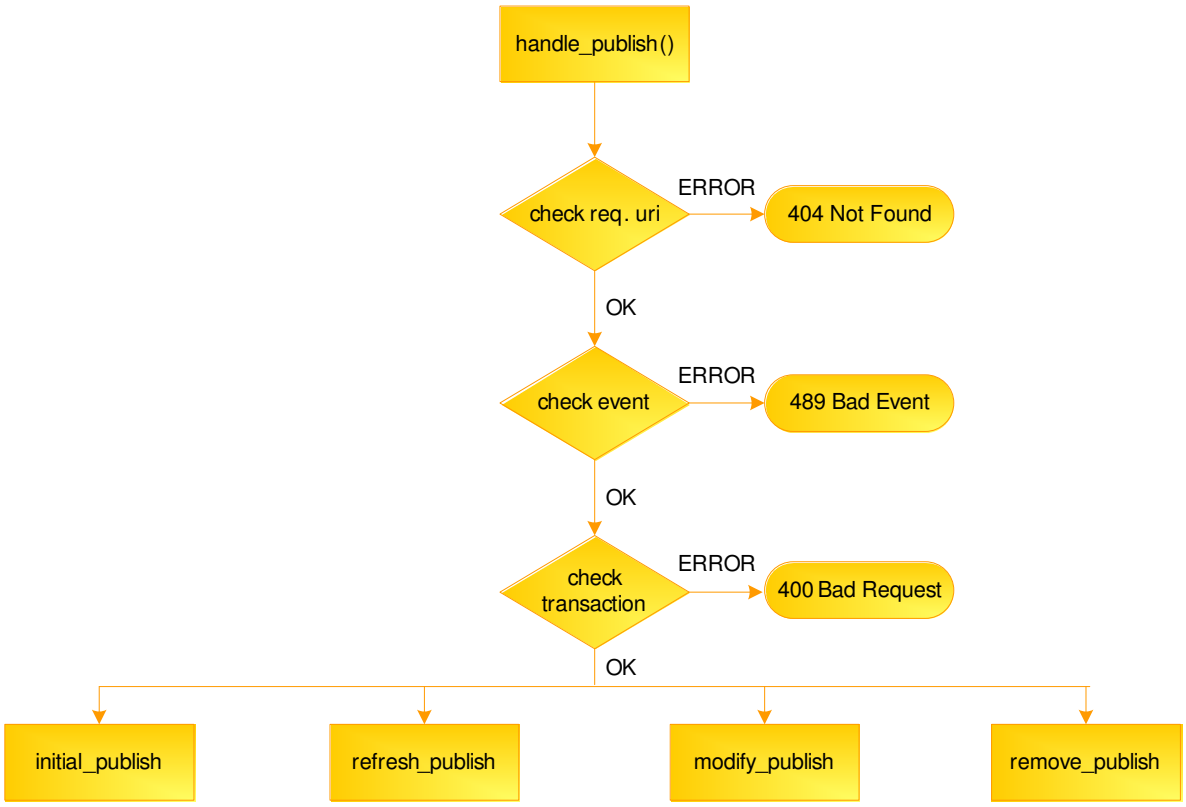


Figure 11 handle_publish () algorithm

First step is to check if request URI concerns to an EPA which the PA is responsible for; if not (for example, a user from another domain) a 404 Not Found response is delivered and the process finished. Then, the "Event" field of the PUBLISH message is checked; only presence events will be accepted; otherwise, a 489 Bad Event before finish. If everything is ok, next step is to check the type of PUBLISH transaction based on the presence/absence and/or the content of the header fields "SIP-If-Match", "Expires" and if the message includes or not a body, as defined in RFC 3903. If the incoming message does not match with any of the situations described in the next table, a 400 OK Bad Request is delivered and the process finishes.

Table 14 PUBLISH transactions

Operation	Body?	SIP-If-Match?	Expires
Initial publish	Yes	No	Optional, >0
Refresh	No	Yes	Mandatory, >0
Modify	Yes	Yes	Optional, >0
Remove	No	Yes	0

Depending of the result of this checking, a different algorithm is invoked. If an internal error occurs before processing is complete (i.e. unavailability to allocate the necessary memory to store some information) the publication process is aborted and a 500 "Internal

Error" response is sent. This is valid for the whole process and has not been included in the figures for simplicity purposes.

3.4.1.2.1. Initial publish

An initial publish takes place when the incoming PUBLISH includes a non empty body and there is no "SIP-If-Match" header field. Optionally, it can include a nonzero "Expires" header field. Basic algorithm is described below:

First step is to check the content of the body field. If not valid pidf/rpid xml field is presented, the process finishes by sending a 415 "Unsupported Media Type" response. If there is a valid body, the ESC generates an entity tag that will be stored in the presence document and sent in the corresponding reply. For Akogrimo, this entity tag is a random sequence in the form abcXYZxyz, where a, b, c, x, y and z are random characters in the interval a…z and X, Y, Z are random numbers in the interval 0…9. This is the tag that PUAs have to include as "SIP-If-Mach" header fields for the other PUBLISH-related transactions.

It could be possible that there is an EPA node created for that presentity. This can happen if the subscription from the Context Manager arrives to the PA before the PUA performs the initial subscription. In this case, the EPA node is modified (the presence document and the expires value), the response to the PUA is sent and the watchers (only the Context Manager for first phase) are notified (delivery of a NOTIFY message containing the presence document); if the EPA node for that presentity does not exist, it is created and added to the linked list, at the same time that the PUA is notified on the process success (200 OK response). No notifications are needed in this case, because the publication arrived before the subscription.

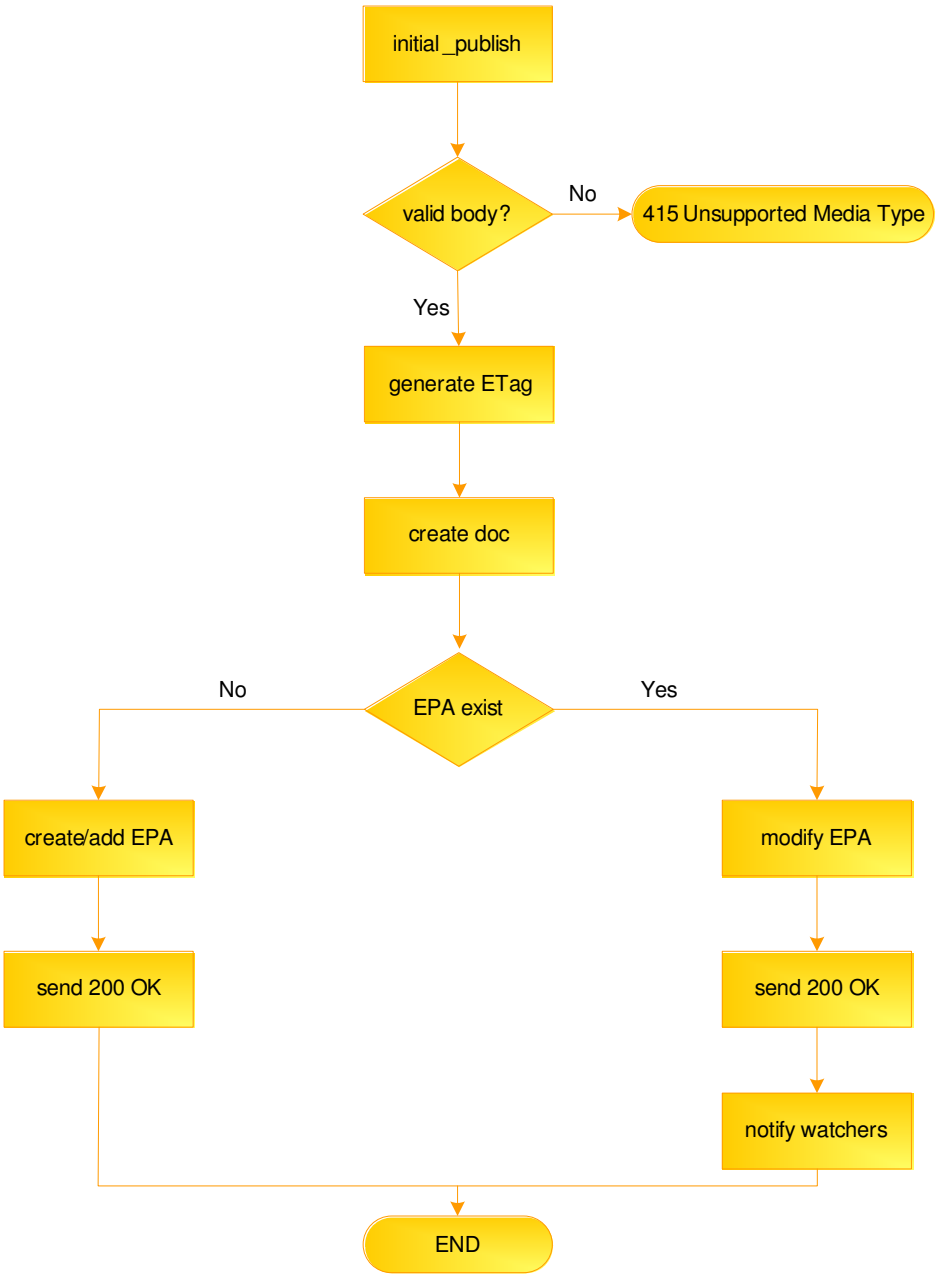


Figure 12 handle_publish () algorithm: initial publish

3.4.1.2.2. Refresh publish

PUAs can refresh the validity of their publications at any time before the presence/context document expiration. To achieve this, they have to send a PUBLISH request including a "SIP-If-Match" header field which contains the SIP-ETag delivered in the response of the initial transaction. It may contain also an "Expires" header field (the default value will be used if not) and it must not contain a body. If all of these conditions are satisfied, the refresh publish algorithm (described in the following figure) is executed.

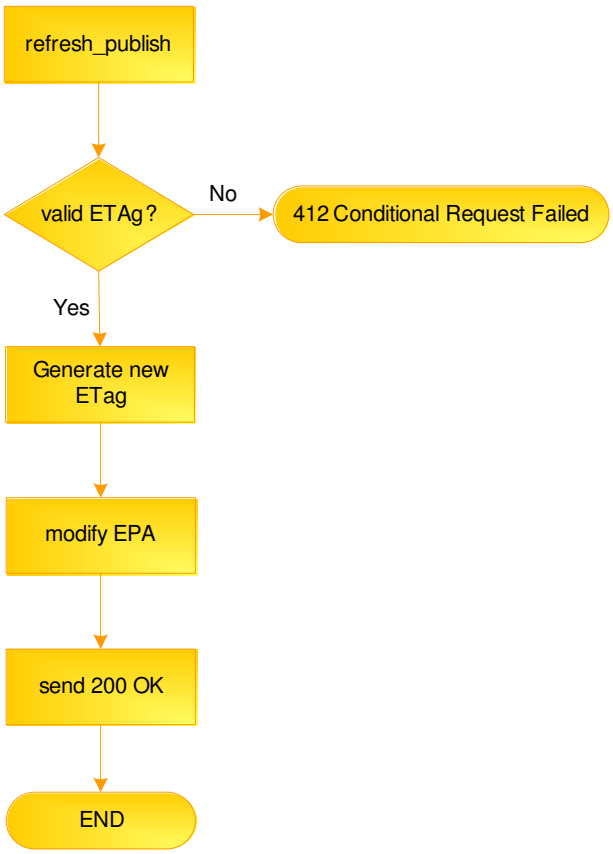


Figure 13 handle_publish () algorithm: refresh publish

This process only modifies the "Expires" field of the EPA node and generates a new entity tag, which replaces the old one and is sent to the PUA in the 200 OK reply, and also stored in the presence/context document. Note that watchers are not notified (this operation do not affect the presence/context status). Note that if the "SIP-If-Match" content of the incoming PUBLISH request do not match with any SIP-Etag stored for the target presentity, a 412 "Conditional Request Failed" response is sent and the process finishes.

3.4.1.2.3. Modify publish

Presentities can modify their presence/context status using a SIP PUBLISH request which contain a valid "SIP-If-Match" header field, a body with a valid pidf-rpid xml document and optionally, one single "Expires" header field. The algorithm to modify the presence/context status of a presentity is depicted in the next figure.

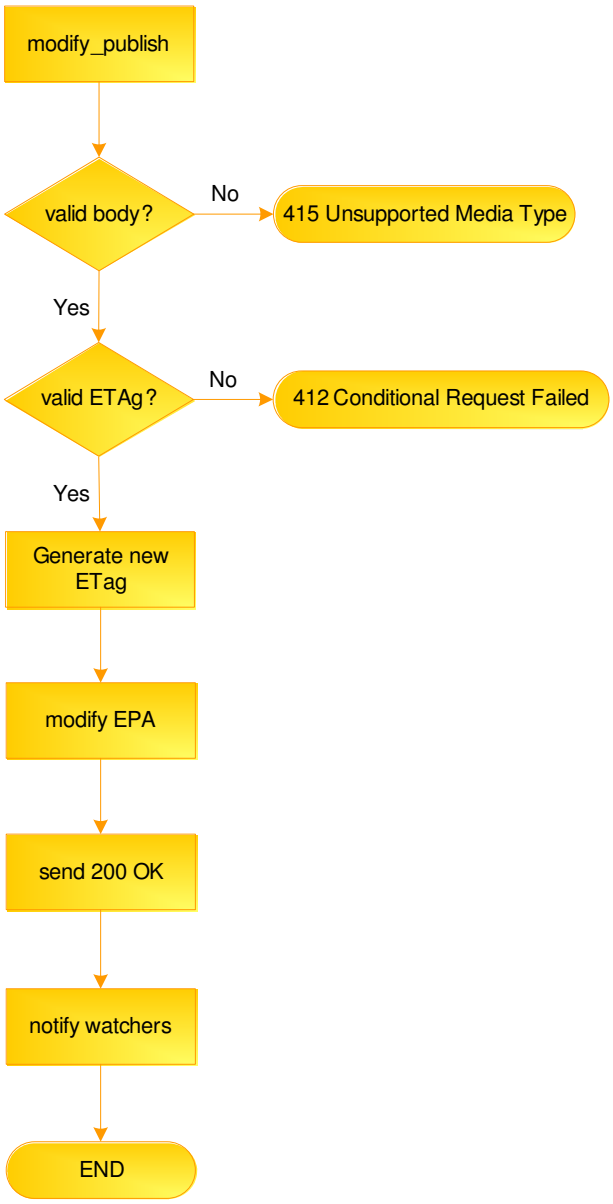


Figure 14 handle_publish () algorithm: modify publish

The process starts checking the integrity of the body content and the validity of the "SIP-If-Match" header field content. If everything is ok, the EPA node is modified (update of the presence/context document, including the new generated ETag and update of the EPA node "Expired" field) and a success response is sent. Subscribers (if any) are notified.

3.4.1.2.4. Remove publish

To request the immediate removal of the event state, an EPA has to create a PUBLISH request with an "Expires" header field set to value "0" and the "SIP-If-Match" header field containing the entity-tag of the publication to be removed. No body must be included. If all of these conditions are satisfied, the process described below takes place.

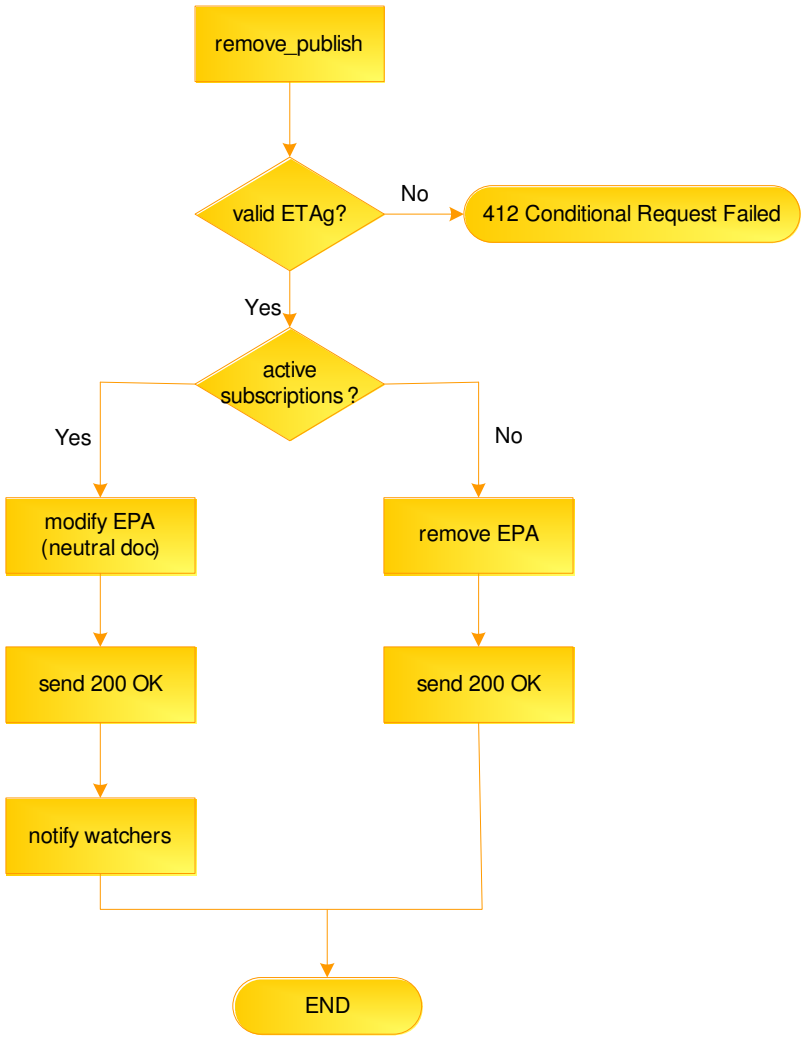


Figure 15 handle_publish () algorithm: remove publish

After checking the validity of the entity-tag value included in the SIP-If-Match header field, the EPA node is removed before sending the response if there are no active subscriptions associated to that presentity; otherwise, EPA structure is modified (presence document and Expires field are set to its neutral values). Then the 200 OK response and the corresponding notifications are sent.

3.4.1.3. Processing SUBSCRIBE requests

The algorithm which describe the behaviour of the Presence Agent when a SUBSCRIBE request arrives is described in RFC 3265 [Roa02] and concretised in RFC 3856 [Ros04]. The following figure depicts the corresponding flow chart.

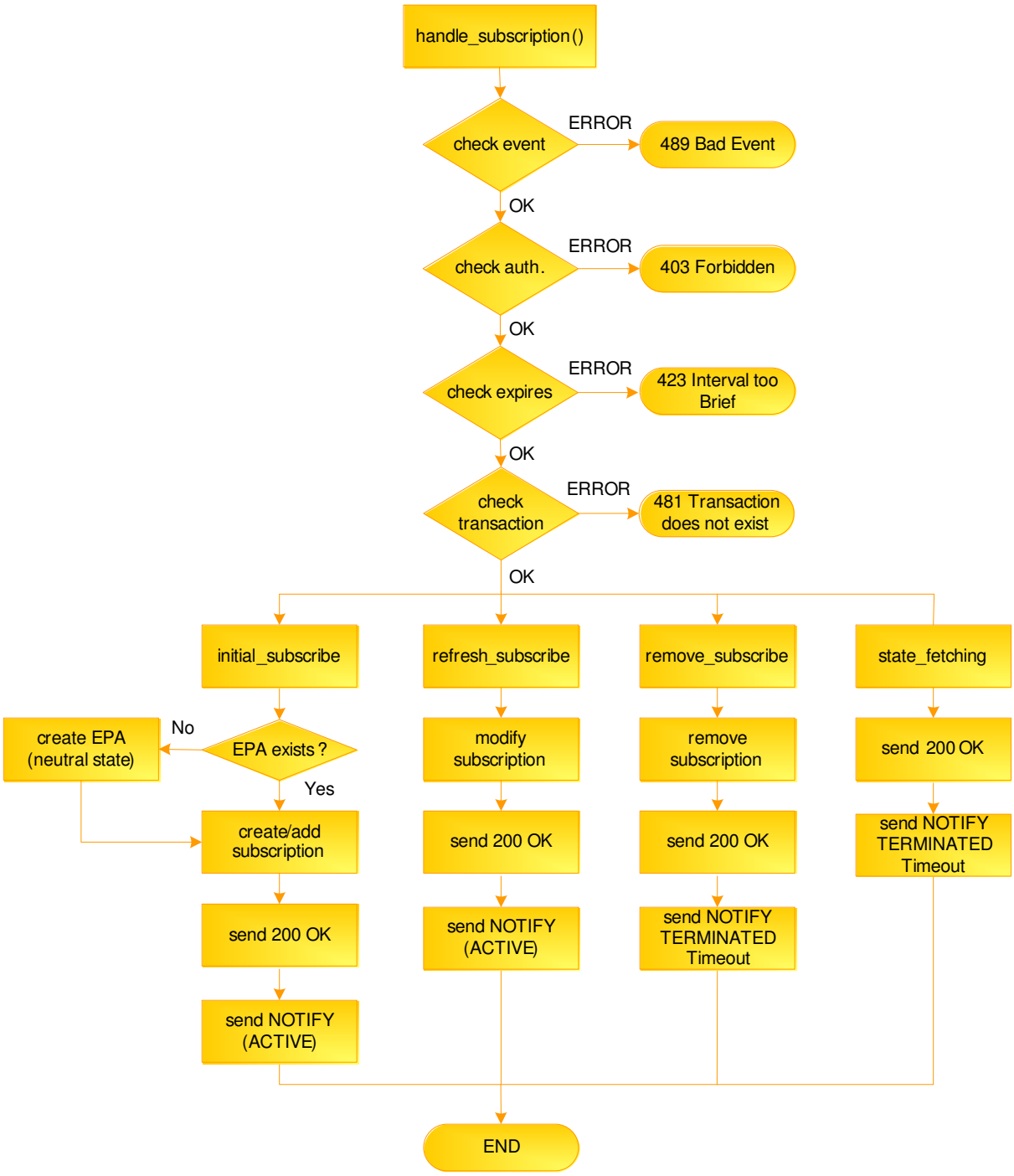


Figure 16 handle_subscriptionh () algorithm

The complete algorithm is described at follows:

1. First step is to check the "Event" field of the incoming SUBSCRIBE request; only presence events will be accepted; otherwise, a 489 Bad Event before finish.
2. Then the authorisation policy is applied. For the Akogrimo first phase, only subscriptions from the Context Manager will be accepted, so if the subscriber is not this component, a 403 "Forbidden" response will be delivered.
3. The process continues by checking the "Expires" header field of the incoming request, which must be 0 or greater than a minimum value, if lower than an hour.

4. Before adjusting this value, transaction type is determined. As indicated in RFC 3265, subscription requests create a dialog, which is fully determined by three parameters: a call identificator, a local tag (tag parameter in the "From" header field) and remote tag (tag parameter in the "To" header field). There are four alternatives for valid SUBSCRIBE transactions, depending on the existence of a previous dialog and the content of the "Expires" header field. Next table shows these alternatives.

Table 15 SUBSCRIBE transactions

Type of subscription	Call ID?	From Tag?	To Tag?	Expires?
Initial	Yes	Yes	No	Optional, >0
Refresh	Yes	Yes	Yes	Optional, >0
Remove	Yes	Yes	Yes	0
Fetch	Yes	Yes	No	0

If the request is trying to modify a subscription within a non existing dialog, a 481 "Transaction does not exist" is sent and the process finishes. In the same way, if "Call-ID" or "From" tag parameters are not present, a 400 "Bad Request" reply is sent.

5. Initial subscription involves the creation of an EPA node in a "default" status if the targeted resource does not exist. This can occur if the subscription arrives before the presentity publishes its status. Then subscription is created and added to the EPA subscription lists. Finally a 200 OK is sent prior to the corresponding NOTIFY with a "Subscription-State" header field set to ACTIVE.

6. A subscription request modifies only the field "Expires" of a subscription node. The rest of the process is the same than initial subscriptions.

7. A subscription remove involves the sending of a 200 OK reply and a NOTIFY response with a "Subscription-State" header field set to TERMINATED and including a "reason" parameter set to TIMEOUT. Previously, the subscription node is removed from the EPA subscription list and deleted.

8. There is a particular situation when a SUBSCRIBE request is received, no previous dialog was established and the "Expires" header field was set to cero. The effect of this is an immediate fetch of the presentity status, without creating a persistent subscription. In this way the subscription node is created but immediately destroyed (not added to the corresponding EPA subscription list) after sending the corresponding replay and notification.

3.4.2. SIP Presence User Agent Implementation

For the first phase, Presence/Context Logic will be kept simple. Since there is no information available about what the applications will be publishing as user state, only the UAProf link provided by the CM Client will be included in the presence document of the user.

Two preconfigured or even hard-coded PIDF XML documents will be used for demonstration purposes. One of this documents will contain presence data saying that the user is online, while the other will say he is offline. A little demonstration application will be implemented to allow the user to switch between the two documents.

Everytime the user switches between one document and the other, a PUBLISH message will be sent to the SIP PA, which will in turn inform the Context Manager. That way, basic presence functionality can be demonstrated.

This is the document that will be used to show that the user is online:

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          entity="sip:server@dit.upm.es">
  <tuple id="sg89gt4">
     <status>
        <basic>open</basic>
     </status>
     <contact priority="1.0">sip:server@138.4.7.148</contact>
     <UAProfLink>http://link.to/uaProfile</UAProfLink>
   </tuple>
</presence>
```

And this is the one that shows the user as being offline:

```
<?xml version="1.0" encoding="UTF-8"?>
<presence xmlns="urn:ietf:params:xml:ns:pidf"
          entity="sip:server@dit.upm.es">
  <tuple id="b5ljt6s">
     <status>
        <basic>closed</basic>
     </status>
     <contact priority="1.0">sip:server@138.4.7.148</contact>
     <UAProfLink>http://link.to/uaProfile</UAProfLink>
   </tuple>
</presence>
```

Only the status element changes its value.

The two of them carry the UAProf link provided by the CM Client. The exact name of the element carrying the UAProf link has to be decided and adapted to the conventions stated in the PIDF specification, RFC 3863 [Sug04].

For the first phase, SIP PUA will also be capable of deleting the user's presence information from the Presence Database, by sending an empty PUBLISH message with an expiration time of 0. That way, Context Manager will be aware of users leaving the system.

For second phase, complete XML handling features are envisaged.

3.5. Testing

3.5.1. Presence Agent Testing

In order to check the Presence Agent functionality, we propose several test cases, basically oriented to verify the functional requirements related to phase I described in section 3.1.1. It should be convenient to enable the trace mode of the Presence Agent library to track debug messages. To perform the test, both a Presence User Agent and a presence watcher are needed.

Test Case	Description	Result
1	Check presence/context publication storage: send an initial PUBLISH message from a PUA	Traces should indicate that the PUBLISH message have been received and the corresponding EPA node created. PUA must receive the corresponding reply (with the corresponding entity-tag).
2	Check presence/context publication storage update: send a PUBLISH refresh message from a PUA (update expires)	Traces should indicate that the PUBLISH message have been received and the corresponding EPA node updated. PUA must receive the corresponding reply.
3	Check presence/context publication storage update: send a PUBLISH update message from a PUA (new PIDF body)	Traces should indicate that the PUBLISH message have been received and the corresponding EPA node updated. PUA must receive the corresponding reply.
4	Check presence/context publication storage delete: send a PUBLISH remove message from a PUA (expires field set to zero)	Traces should indicate that the PUBLISH message have been received and the corresponding EPA node deleted (if no active subscriptions; updated, otherwise). PUA must receive the corresponding reply.
5	Check presence/context publication expiration control: send a PUBLISH message from a PUA with a very short expires value and let it expires.	Traces should indicate that the PUBLISH message have been received and the corresponding action over the EPA node performed; next PA timer iteration (if set to a higher value than the expiration time) should delete the EPA Node.
6	Check presence/context publication RFC compliance: send several invalid PUBLISH messages (no Event field included, try to refresh an expired publication....)	Traces should indicate that the PUBLISH message have been received, the problem detected and the corresponding reply sent to the PUA, which must receive the corresponding reply.
7	Check presence/context subscriptions: send an initial SUBSCRIBE message from a PUA to request for the presence status of an active/inactive presentity (active if the presentity has already published its event	Traces should indicate that the SUBSCRIBE message have been received and the corresponding subscription node created. PUA must receive the corresponding reply and a NOTIFY response including a body with the published presentity status

Test Case	Description	Result
	state).	(empty/default if no publications from that presentity stored).
8	Check presence/context subscriptions: send a SUBSCRIBE refresh message from a PUA	Traces should indicate that the SUBSCRIBE message have been received and the corresponding subscription node updated. PUA must receive the corresponding reply and a NOTIFY response including a body with the published presentity status (empty/default if no publications from that presentity stored).
9	Check presence/context subscriptions: send a SUBSCRIBE remove message from a PUA (expires field set to zero).	Traces should indicate that the SUBSCRIBE message have been received and the corresponding subscription node removed. PUA must receive the corresponding reply and a NOTIFY response including a body with the published presentity status (empty/default if no publications from that presentity stored). If presentity has expired its publications and it has no more active subscriptions, EPA node should be also removed.
10	Check presence/context subscriptions expiration control: send a SUBSCRIBE message from a PUA and let it expires.	Traces should indicate that the SUBSCRIBE message have been received and the corresponding action over the subscription node performed; next PA timer iteration (if set to a higher value than the expiration time) should delete the subscriber node. PUA should receive a NOTIFY message with the adequate parameters.
11	Check presence/context subscriptions RFC compliance: send a invalid SUBSCRIBE message from a PUA.	Traces should indicate that the SUBSCRIBE message have been received, the problem detected and the corresponding reply sent to the PUA, which must receive also the corresponding reply.
12	Check presence/context framework authorisation control: send a SUBSCRIBE request from a non-authorised PUA. PA must be configured to admit subscriptions from a certain SIP URI only.	Traces should indicate that the SUBSCRIBE message have been received, the problem detected and the corresponding reply sent to the PUA, which must receive also the corresponding reply.
13	Check PA memory leaks: abort SER process having active publications and subscriptions	Traces should indicate that all memory has been freed (no memory leaks).

3.5.2. SIP Presence User Agent Testing

Test cases will check SIP PUA functionality on presence publication, update and cancellation:

Test Case	Description	Result
1	Initial presence information publication will be performed when CM Client provides UAProf link.	SIP PUA will create a new PIDF XML document and will send it in a PUBLISH message. Publication will reach the SIP PA and the Context Manager will be notified. The Context Manager will receive the presence information of the user and will be able to retrieve the UAProf link successfully.
2	Publication update to reflect changes in user's presence status.	Updated presence information will be included in PUBLISH messages, which will be sent to the SIP PA to inform about the change in the user's presence status. Updates to the user's presence information will reach the SIP PA, which will notify the Context Manager about the changes. The Context Manager will update the user's presence information accordingly.
3	Publication will be cancelled on application shutdown to show the user is leaving the system.	Correctly formatted PUBLISH message will be sent to the SIP PA. The PUBLISH message will reach the PA and the user's presence information will be deleted from the Presence Database. The Context Manager will be notified.

4. A4C

The A4C system provides the necessary functionality for authenticating users, authorizing access to services, accounting and charging for service usage, as well as auditing within Akogrimo. It provides an interface to other components which allows them to access and store key information about users and their usage of services, this data is stored within a central database that is maintained by the A4C Server. Important components that access A4C include Base VO manager, SIP server, context manager, access router, and QoS broker.

4.1. Requirements

Table 16 provides an overview on the key requirements that the A4C system has to meet.

Table 16 A4C – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**Authentication**	**X**	**X**
F 1.1	Authentication of users based on PANA (username/password) for network access.	X	
F 1.2	Authentication of users based on SAML (IDToken) for grid services and network services (e.g. SIP). Verification of the authentication based on the IDToken.	X	
F 1.3	Single-sign-on support based on SAML (IDToken).	X	
F 1.4	IDToken management, generation and verification.	X	
F 1.5	Additional authentication mechanisms.		(X)
F 1.6	Private Authentication generation		(X)
F 2	**Authorization**	**X**	**X**
F 2.1	Authorization of network services (i.e. type of access, QoS)	X	
F 2.2	Provide QoS Profile to the QoS Broker	X	
F 2.3	Provide the generic user profile		X
F 2.4	Provide anonymous/pseudonymous access to different services		(X)
F 2.5	Authorize User or SP		(X)

Req. nr	Description	Phase 1	Phase 2
F 3	**Accounting**	**X**	**X**
F 3.1	Accounting for network services usage (e.g. volume, QoS)	X	
F 3.2	Support of accounting during handover and roaming		X
F 3.3	Accounting for grid services usage, support of grid accounting parameters	(X)	X
F 3.4	Accounting for compound services, support of service hierarchy		X
F 3.5	Multi-domain support, correlation of accounting records from different domains related to the same service		X
F 4	**Auditing**		**X**
F 4.1	Manage, record and store auditing events (e.g. SLA violation) from grid services (e.g. Monitoring component)		X
F 4.2	Non-repudiation of service usage		(X)
F 4.3	Auditing of A4C messages exchanged between domains		X
F 5	**Charging**	**(X)**	**X**
F 5.1	Charge calculation for service consumption based on accounting records and tariff profiles	(X)	X
F 5.2	Support of service-specific and user-specific tariff profiles	(X)	X
F 5.3	Charging for compound services, support of service hierarchy	(X)	X
F 5.4	Multi-domain support for the charge calculation		X
F 5.5	Charging record exchange between domains		X
F 5.6	Single bill support, create bills for users		X
F 6	**Generic**	**X**	**X**
F 6.1	Manage and store user profiles	X	
F 6.2	Manage and store tariff profiles	X	

Req. nr	Description	Phase 1	Phase 2
F 6.3	Manage and store accounting records	X	
F 6.4	Manage and store charging records	X	
F 6.5	Session support, binding A4C tasks together	(X)	X
F 6.6	Secure communication (IPSec, TLS), protect inter-domain communication		(X)

(X) Optional requirement

Table 17 SAML Authority – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**IDToken Management**	**X**	**X**
F 1.1	IDToken generation	X	
F 1.2	Verification of the IDToken and provision of the SAML authentication assertion	X	
F 1.3	SAML attribute assertion generation	(X)	X
F 1.4	Signature and encryption of IDToken		X
F 1.5	Signature and encryption of SAML Assertions		X
F 1.6	Provision of a token with delegation support		(X)

Table 18 A4C – Non-functional requirements

Req. nr	Description	Phase 1	Phase 2
N 1	Running on Linux (Ubuntu) on a PC platform	X	
N 2	Database support	X	
N 3	Provide A4C Client C++ and Java API	X	
N 4	Performance improvement (efficiency, speed)		X

4.2. Interfaces

Figure 17 provides an overview on the main external and internal interfaces of the A4C component. The A4C Server is a central entity providing A4C functions. Akogrimo network components requiring A4C functions use the A4C Client Library to communicate

with the A4C Server via the Diameter protocol [Cal03]. Grid components can either include the A4C Client Library or the Library can be used to create an A4C SOAP Gateway, which provides a SOAP interface for Grid components. Both alternatives are shown in Figure 17. The SAML Authority assists the A4C Server in the authentication process and provides the IDToken management and verification. Section 4.3 provides additional details on the A4C deployment in a network environment.

The API of the A4C Client Library used by Akogrimo components to integrate the A4C Client is described in Section 4.2.2. The internal interfaces of the A4C infrastructure not visible for external components are described in Section 4.2.3 and Section 4.4.

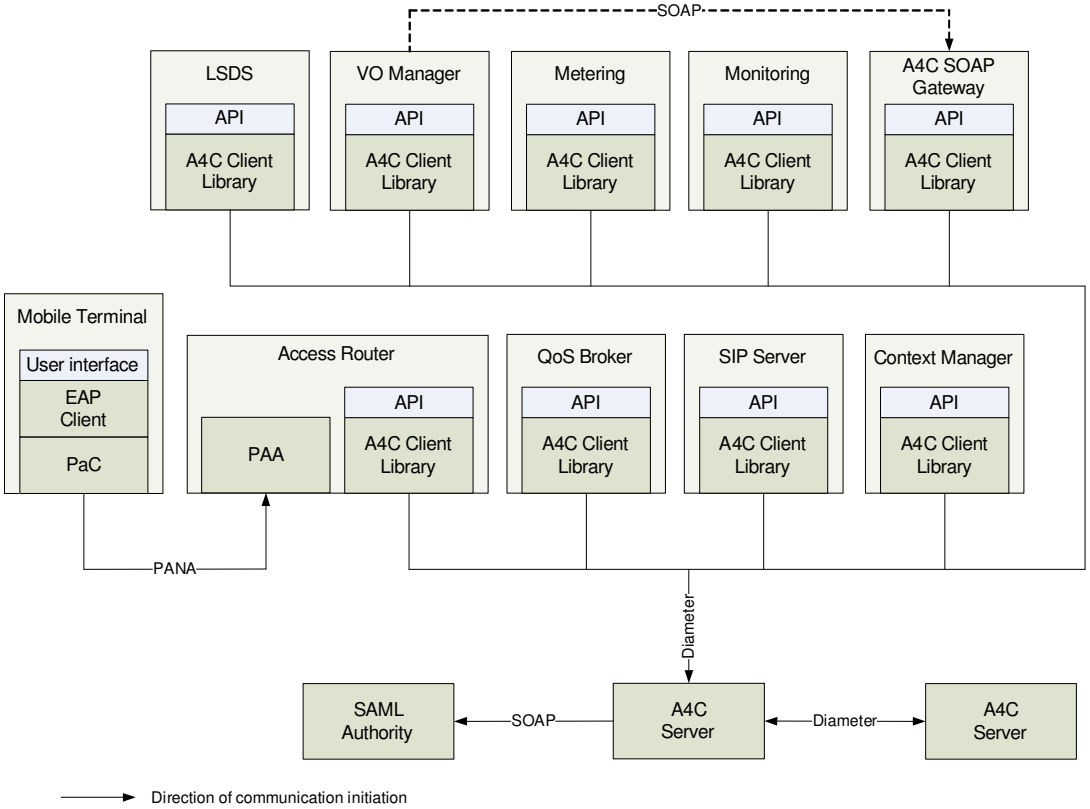


Figure 17: Overview of A4C interfaces

Akogrimo uses the Security Assertion Markup Language (SAML) [SAML] to send security information in the form of authentication and attribute assertions to the Akogrimo components. Akogrimo is provided with a SAML infrastructure through a SAML Authority component. This SAML component generates SAML messages and provides them to the A4C Server. The SAML Authority interfaces directly with the A4C Server in the form of request and response messages.

In order to have a complete SAML infrastructure, all Akogrimo components that require authentication and/or attribute information from the user via the A4C Server, need to have a SAML Client. The SAML Client consists of an API that provides the methods needed to understand SAML assertions. The SAML Client will be a module of the A4C Client.

4.2.1. A4C Server Interface (E-A4C-1)

The A4C Server interface comprises a single management interface used for A4C Server configuration and maintenance.

4.2.1.1. Management Interface (E-A4C-1.1)

It provides an interface for the management and configuration of the A4C Server for the network administrator. In phase 2 it can be extended to support the communication with external components in order to send configuration parameters to the A4C Server, *e.g.* configuration according to a negotiated SLA.

4.2.2. A4C Client API (E-A4C-2)

The A4C Client API comprises several sub-interfaces:

- Authentication and Authorization API
- Accounting API
- Auditing API

4.2.2.1. Authentication and Authorization Interface (E-A4C-2.x, I-A4C-2.x)

The authentication interface will be mainly used by the Access Router, for the initial user authentication, and by the Base VO manager for verification of a user's IDToken. The SIP Proxy can also use this interface if it needs to validate a user's token.

The authorization interface deals mainly with profile management in the A4C, mainly the QoS profile of a user, which will be requested by the QoS Broker, and the Generic User Profile, which will be requested by the Base VO Manager.

The following methods are provided by the Authentication and Authorization API:

- *AuthnContainer userAuthenticationRequest(UserName, Password)*

  The *userAuthenticationRequest()* method is used to trigger the initial user authentication, based on a username and a password. After this authentication the user should receive an IDToken to be used for further service requests and the A4C SessionID for binding further A4C sessions to the initial authentication session.

  The following is the list of parameters and return values of this method:

  - *AuthnContainer:* is a helper class provided by the AA API that is used to store the authentication result received from the A4C Server. The main attributes of this container are:
    - *ErrCode:* is an integer specifying whether the authentication was successful or not.
    - *IDToken:* Contains the token generated by the SAML Authority of the A4C Server for the user during the authentication process. This token needs to be passed to any service that is requested by the

user in order to assure that the service can verify that the user was authenticated before accessing the service.
        - *A4C SessionID:* is the String identifier of the A4C Authentication session. This ID will be provided whenever a service will be started, in order to allow the binding of future service accounting sessions to the initial authentication session.
        - *LifeTime:* is an integer representing when this authentication will expire
    - *UserName*: is a String, representing the full username of the user who wants to authenticate, in the NAI format (username@domain)
    - *Password*: is a String containing the password provided for authentication

- ***AuthnContainer authenticationVerificationRequest(UserName, IDToken, ServiceID)***

The *authenticationVerificationRequest()* method is used to verify a user's authentication, based on an IDToken that he provides. The result of the authentication verification will be stored in the **AuthnContainer** object. This method will be used by the VO Manager during a user's VO Login and by the SIP Proxy for performing a SIP Registration for a user.

- ***QoSProfile qosProfileRequest(UserName)***

The *qosProfileRequest()* is used by the QoS Broker for retrieving the QoS profile of a user. (also called the Network View of the User Profile or NVUP).

    - *QoSProfile:* An XML document containing a QoS Profile. The XML format of this document will be defined in the scope of WP4.1.

- ***GenericProfile userGenericProfileRequest(UserName)***

The ***userGenericProfileRequest()*** method is used for retrieving the Generic User Profile of a user. This method will be primarily used by the VO Manager for retrieving a user profile upon a VO Login, and by the Context Manager for getting the SIP URI and user's RFID from the profile.

    - *GenericProfile:* An XML document containing a generic user profile.

4.2.2.2. Accounting Interface (E-A4C-2.x)

The accounting interface is divided into the following two sub-interfaces:

- *AccountingClient* – manages the communication with the A4C Server, manages different accounting sessions from the same client
- *AccountingRecord* – serves as a container for the accounting data to be sent to the A4C Server

- *AccountingResponse* – is a helper class and serves as a container for the answers received from the A4C Server

The A*ccountingClient* interface offers the following methods:

- ***bool accountingClientStart(ConfigFilename)***

  The *accountingClientStart()* method is used for starting the A4C accounting client.

- ***AccountingResponse accountingSessionStart(UserName, ServiceID, ParentSessionID)***

  The *accountingSessionStart()* method is used for creating a new A4C accounting session for a given user and service instance. Whenever a user starts a service session, an accounting session is created and all accounting records created during the service session will be sent in this accounting session.

  The following is the list of parameters and return values of this method:

  - *ParentSessionID:* is a String containing the A4C session id of the process that started the service. Can be the A4C session ID of the authentication session or the A4C session ID of the accounting session of the parent process.
  - *AccountingResponse:* is a container that stores the answer received from the A4C Server and has the following attributes:
    - *SessionID:* is a String containing the A4C session id of the newly created A4C accounting session. This parameter will be further used to map accounting records to accounting session.
    - *AccountingInterval:* is an integer and specifies the interval to be used between two consecutive accounting records

- ***boolean sendAccountingRecord(SessionID, AccountingRecord)***

  The *sendAccountingRecord()* method is used to send an accounting record to the A4C Server. The accounting record needs first to be created, filled with the required AVPs and then sent. The following is the list of parameters and return values of this method:

  - *SessionID:* The A4C accounting session ID to which the accounting record belongs to
  - *boolean:* True, if the request was successful

- ***AccountingRecord createAccountingRecord(SessionID)***

  The *createAccountingRecord()* method is used to create an accounting record. The accounting record will be automatically filled with some required AVPs like username, serviceId, etc..

  The following is the list of parameters and return values of this method:

- *SessionID*: contains the session ID of the accounting session to which the accounting record belongs
- *AccountingRecord*: is an object that will act as a container for the metrics that need to be included in the accounting record

The *AccountingRecord* sub-interface offers the following methods:

- ***ResultCode addAVP(AvpID, Value)***

  The *addAVP()* method is used for adding an Attribute-Value-Pair to an existing *AccountingRecord* object. It will be used whenever a metering component needs to add a metered parameter to an accounting record.

  The following is the list of parameters and return values of this method:

  - *AVPID*: is the identifier of the parameter to be included in the accounting record, as defined in the Diameter dictionary
  - *Value*: parameter's value
  - *ResultCode*specifies if the operation was successful or not.

- ***ResultCode removeAVP(AvpID)***

  The *removeAVP()* method is used for removing an Attribute-Value-Pair from an existing *AccountingRecord* object.

  The following is the list of parameters and return values of this method:

  - *AVPID*: is the identifier of the parameter to be included in the accounting record, as defined in the Diameter dictionary
  - *Value*: parameter's value
  - *ResultCode*specifies if the operation was successful or not.

4.2.2.3. Auditing Interface (E-A4C-2.x)

The auditing interface will be used by components monitoring service sessions for informing the A4C on detected SLA violations during service provisioning. The auditing API provides the following methods:

- ***ResultCode reportEvent(EventID, EventBody)***

  The *reportEvent()* method is used for reporting an SLA violation event during service provisioning. The following is the list of input parameters and return values of this method:

  - *EventID:* is an integer identifying the event type. The event types for a service need to be specified in the SLA negotiated before the service start
  - *EventBody:* can be a decimal value or a string.

4.2.3. SAML Client API (I-A4C-2.10)

The SAML Client is a module of the A4C Client. The SAML Client consists of the SAML Library and the SAML Client API. The following methods will be provided by the SAML Client:

- *boolean checkAssertionIsValid (SAMLAssertion)*

  When the SAML assertion is received from the A4C Server, the receiver must check the validity of this assertion. This implies e.g. checking the signature of the A4C, checking that the date of issuance of the SAML assertion is previous to the reception of the message etc.

  - True: If the validation checks resulted as expected.
  - False: If an error was found during the process of the validation.

- *time obtainTimeOutFromAssertion (SAMLAssertion)*

  When the SAML assertion is received, the time of validity of the assertion must be checked. . The result of the request is the timeout of the assertion. After the timeout, the assertion is no more valid, and a new authentication request must be issued.

- *arrayList obtainUserProfileFromAttributeAssertion (SAMLAttributeAssertion)*

  This method will extract the attributes from the SAML assertion. It will provide a list of arrays with the name and value of the attributes of the users. It will provide an error instead, if no attributes match.

- *string obtainSubjectFromAuthenticationAssertion (SAMLAuthenticationAssertion)*

  The result of this is a string with the username of the SAML assertion. It can be used for attribute and authentication assertions.

- *string obtainAuthenticationMethodFromAuthenticationAssertion (SAMLAuthenticationAssertion)*

  The result of this is a string with the authentication method that was used by the user to authenticate at the A4C Server.

4.2.4. SAML Authority Client API (I-A4C-5.x)

The A4C Server provides requests to the SAML Authority. The SAML Authority, when receiving the requests, generates the responses based on the A4C Server information request. The communication protocol between them will be SOAP [SOAP]. The A4C Server will make SOAP Requests, acting as a SOAP Client. The SAML Authority acts as a SOAP Server, and will generate the response, and send them back within the body of a SOAP message.

The following requests are expected from the A4C Server:

- *String generateIDToken (username, authenticationMethod)*

  The SAML Authority receives this request from the A4C Server. The result of the method will be the IDToken as a string.

- *String validateIDToken(IDToken, ServiceID, username)*

  The output of this method is the SAML authentication assertion. An example of a SAML assertion can be found in the design section.

- *String generateSAMLAttributeAssertion(List of Attributes, ServiceID, username)*

  The output of this method is a SAML attribute assertion, with a list of the attributes of the user profile that the component is allowed to show.

4.3. Design

The A4C infrastructure of Akogrimo is responsible for the authentication of users, authorization of service access, accounting and charging for service usage, and the auditing of service provisioning. Figure 18 shows the components of the A4C infrastructure and the interfacing components within network architecture (the A4C SOAP Gateway is represented by the Grid components in the figure).

The A4C infrastructure consists of the following components:

- A4C Server: it performs and manages the A4C tasks and communicates with other servers within and across domains.
- A4C Client: it supports the communication and message exchange with the A4C Server and is needed within each network component requiring A4C support.
- SAML Authority: It is responsible for the generation and verification of SAML IDTokens and SAML Assertions.

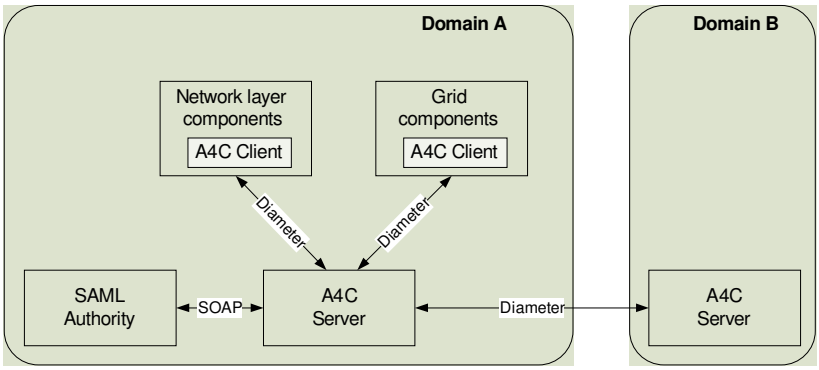


Figure 18: A4C view of the network architecture

The A4C infrastructure and the communication between A4C Client and Server are based on the Diameter protocol [Cal03]. According to Figure 18 each domain has its own A4C Server (Domain B is only represented by one A4C Server in the figure). Within a domain there can be more than one A4C Server, each of which being responsible for a separate group of users. This provides better load distribution, scalability, and redundancy. The communication between different domains is performed via the A4C Servers. The A4C Server is coupled with the SAML Authority, which provides the generation and verification of IDTokens for user authentication. The interface between A4C Server and SAML Authority is based on the SOAP protocol.

In order to enable communication between A4C Servers belonging to different domains, a trust relationship is required to exist between these domains. The trust relationship is established at the A4C level by specifying in the configuration file of the A4C Server the neighbouring domains that it may trust and the A4C applications (authentication, authorization, accounting, etc…) that are allowed to exchange information with the trusted domain.

Every component which needs to communicate with the A4C Server in order to make use of the server's A4C functions needs to integrate its own A4C Client. Several A4C Clients can be connected to one A4C Server, e.g. Access Routers in an access network. The A4C Client provides the means for the Diameter based communication and message handling, but it does not provide the component-specific functionality on the client side. Each component is responsible to implement its desired functionality and to make use of the A4C Client Library according to the required application logic, e.g. authentication. This means that in the case of the Access Router component for example, the A4C Client allows for the transmission of Diameter messages and the communication with the A4C Server, but it does not provide the means to perform access control or to meter network resource usage.

Akogrimo components requiring A4C functions integrate the A4C Client by using the A4C Client Library and communicate with the A4C Client via a C++ or Java interface described in Section 4.2. This enables a direct communication with the A4C Server over the Diameter protocol. The A4C Client Library enables the creation of an A4C SOAP Gateway, which translates between the SOAP and Diameter protocols by providing a SOAP interface. The A4C SOAP Gateway can be used if the Akogrimo component cannot integrate and use the A4C Client Library directly. The A4C SOAP Gateway is deployed on a separate physical component. Several Akogrimo components can use the same A4C SOAP Gateway.

The Diameter base protocol enables the transfer of authentication, authorization and accounting messages. To support the requirements of Akogrimo, the Diameter protocol has to be extended with new messages and AVPs (Attribute-Value-Pair), supporting the transmission of new parameters. Diameter agents, i.e. relay, proxy, redirect and translation agents, provide message forwarding and translation services. The A4C infrastructure of Akogrimo supports the client and server functionality in the first phase.

4.3.1. SAML Authority

Akogrimo uses the Security Assertion Markup Language (SAML) to send security information in the form of authentication and attribute assertions to the Akogrimo components. SAML provides an additional security block concerning high confidential information (like authentication and attribute information of a user) in the Akogrimo architecture. SAML is a secure interoperable language used to share user's information from the A4C Server to other components in order to provide Single Sign-On (SSO) capability to the user and to offer attribute sharing of the user to other components. In order to provide SAML messages, a SAML Engine is needed in Akogrimo. This is the SAML Authority. The SAML Authority is part of the security infrastructure in Akogrimo.

It generates XML messages based on the SAML standard to send authentication and attribute information. The SAML Authority is an internal subcomponent of the A4C Server. It aims at supplying IDTokens and SAML assertions to the A4C Server. The A4C Server contacts the SAML Authority when it requires to generate IDTokens and to verify such tokens presented by different components.

4.3.1.1. IDToken

The SAML Authority is in charge of the generation of the IDToken. The IDToken is a credential that can be reused several times with different entities in Akogrimo for authentication purposes. It is generated by the SAML Authority and it is used by the Mobile Terminal to request access to different components. The IDToken in Akogrimo is designed as an enhanced SAML artefact [SAMLProfile]. The SAML artefact, as defined in the SAML standard, is a random-generated number that acts as a pointer of the SAML assertions of a user at the SAML Server. It can only be used once. Since Akogrimo requires a reusable artefact, it uses an enhanced token based on the Daidalos European Project [DaidalosIDToken]. This token will include the following parameters:

- SAML Artefact: This parameter is a random-generated number. It is the pointer of the SAML assertions in the SAML Authority. This number remains the same at each usage of the token.
- Serial Number: This parameter is a counter. It will be increased by 1 each time the IDToken is used, in order to avoid replay attacks.
- Random Number: This parameter is a random-generated number and it is changed each time the MT uses the token. It is used to avoid security attacks.
- Signature: The signature of the issuer of the token. The IDToken will be signed from the A4C Server, the first time it issues the IDToken and sends it to the MT. It will be signed the MT when it is sent from the MT to request access to a service.

4.3.1.2. Verification of IDToken and Generation of the SAML authentication assertion

When a components receives an access request from the MT, it receives appended the IDToken. In order to know if the user is authenticated, it requests the A4C Server for information of the token. The A4C Server then, contacts the SAML Authority to validate the token and obtain the authentication information in the form of a SAML assertion. The validation of the IDToken consists of verification of the sequence number and signature of the IDToken. When the token is valid, a SAML assertion is issued in order to provide the authentication information of the user. The SAML assertion [SAMLAssertion] will contain the name of the user and the authentication method proceeded. An example of a SAML assertion is shown in Figure 19. The XML elements <Name Identifier> and <AuthenticationMethod> contain the userID and the authentication method which was performed, respectively. The <Conditions> element shows also relevant information like the time the assertion is valid or the recipient of the SAML assertion.

```
<Assertion AssertionID="_a75adf55-01d7-40cc-929f-dbd8372ebdfc"
   IssueInstant="2005-10-17T00:46:02Z"
   Issuer="https://samlauth.a4cserver.uni-stuttgart.de"
   MajorVersion="1" MinorVersion="1"
   xmlns="urn:oasis:names:tc:SAML:1.0:assertion">
<Conditions
   NotBefore="2005-10-17T00:46:02Z"
   NotOnOrAfter="2005-10-17T01:46:02Z">
   <AudienceRestrictionCondition>
      <Audience>http://vomanager.ehealth.com
      </Audience>
   </AudienceRestrictionCondition>
</Conditions>
<AuthenticationStatement
   AuthenticationInstant="2005-04-17T00:46:00Z"
   AuthenticationMethod="urn:oasis:names:tc:SAML:1.0:am:password">
   <Subject>
     <NameIdentifier
     Format="urn:oasis:names:tc:SAML:1.1:nameid-format:emailAddress"
     NameQualifier=" https://samlauth.a4cserver.uni-stuttgart.de">
       ernestosanchez@akogrimo.org
     </NameIdentifier>
     <SubjectConfirmation>….</SubjectConfirmation>
   </Subject>
   <SubjectLocality IPAddress="127.0.0.1"/>
</AuthenticationStatement>
</Assertion>
```

Figure 19: SAML assertion example

4.3.1.3. User Profile Generation

The user profile is a set of attributes that define several characteristics of a user. As the authentication, this information is confidential and must be securely sent to the receiver. The user profile information will be sent in a SAML attribute assertion. When the A4C Server receives a user profile request, it forwards the request to the SAML Authority. The SAML Authority will receive an attribute request together with the list of attributes from the A4C Server. Then, the SAML Authority will generate a SAML attribute assertion with the information of the user and it will send it back to the A4C Server.

4.3.2. Authentication and Authorization

There are different kinds of authentication and authorization sequences supported by the A4C Server. It supports authentication based on username and password for network login, authentication based on the IDToken for Grid services, QoS authorization for network services, and the retrieval of the generic user profile.

Network login

Network login is the first task that needs to be performed before using any Akogrimo service. During this phase the user is authenticated by the A4C Server of his provider and an ID Token is issued for him. The authentication performed during network login is based on username and password. The authentication messages will be encapsulated in EAP (Extensible Authentication Protocol) [Blu98] message and will be transported over PANA (Protocol for Carrying Authentication Information) [For05] between the Mobile Terminal and the Network Authentication Server (NAS), and over Diameter between NAS and the A4C Server. Figure 20 shows the messages exchanged during the network authentication. There are 4 phases that can be distinguished during the network authentication:

1. Handshaking – the components involved in the authentication prepare for the authentication process
2. Credential verification – there can be several messages transporting authentication information between the MT and A4C. At the end of this phase the A4C finally makes the authentication decision
3. Token generation – after a successful authentication an ID Token is generated for the new authenticated user
4. Authentication termination – the new generated ID Token is delivered to the user and the access router is informed that the user was authenticated. From this point the user can request any service using the ID Token provided and the access router can start an accounting session for the user.

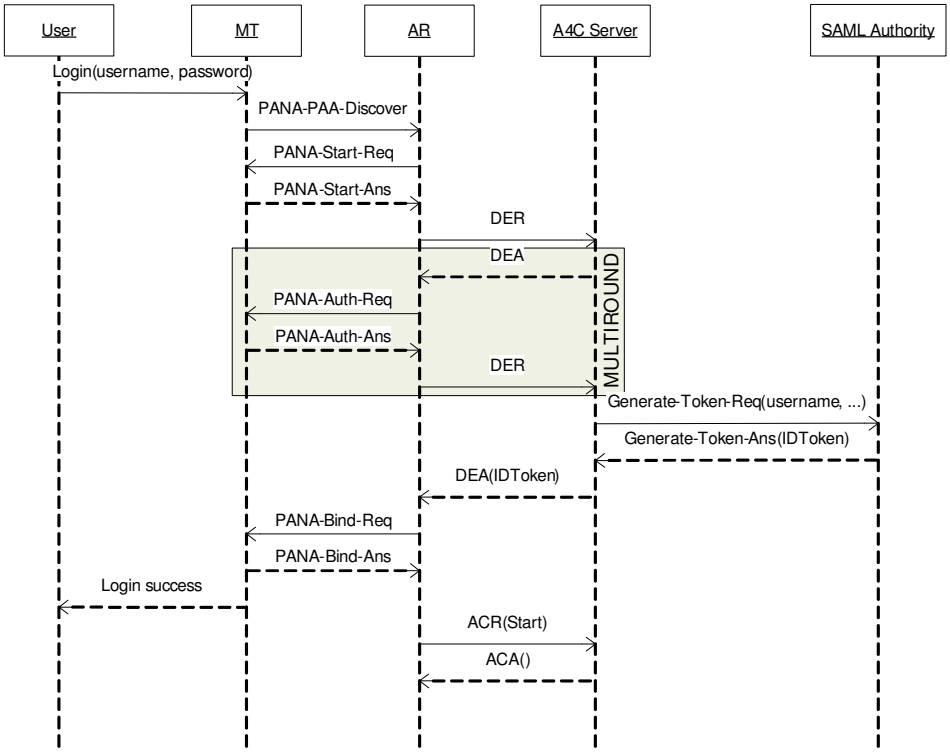


Figure 20: Network login MSC

Network Logout

After the user logs out, the AR stops accounting and sends the last accounting record to the A4C Server in the Accounting-Request (ACR) message. The AR also terminates the running session by sending a Session-Termination-Request (STR) to the A4C Server. The IDToken of the user will be removed in the SAML Authority.

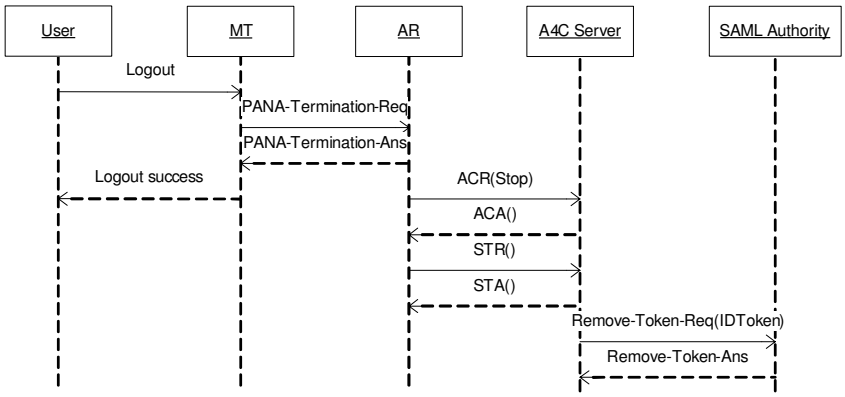


Figure 21: Network logout MSC

IDToken based authentication

The A4C Server supports authentication based on the SAML IDToken. An Akogrimo Component can request authentication by sending an AA-Request (AAR) to the A4C Server. The message includes the username, the SAML IDToken, and the serviceID. The A4C Server requests the verification of the IDToken from the SAML Authority. After verifying the IDToken, the SAML Authority sends back the result to the A4C Server. The A4C Server responds with a AA-Answer (AAA) to the Akogrimo Component. Akogrimo Components requiring the IDToken based authentication are the SIP Registrar, the SIP Server, the Base VO Manager, and WS Applications for 3PCC.

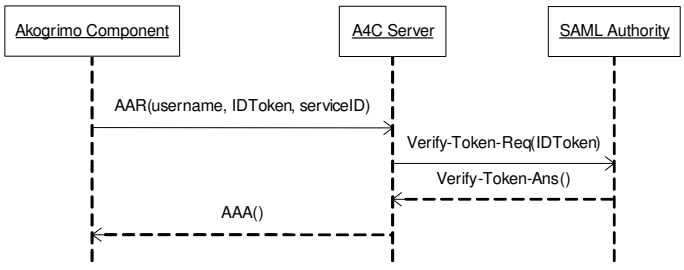


Figure 22: IDToken based authentication MSC

QoS authorization

The A4C Server stores the network QoS profile of every user, which specifies the QoS parameters the user is allowed to request. The QoS Broker in the access network is responsible for the management of the network QoS. The QoS Broker can retrieve the QoS profile from the A4C Server by sending a QoS-Authorization-Request (QAR) as shown in Figure 23. The request contains the username and the serviceID. The serviceID specifies the requested service and it can refer to the access type, e.g. WLAN. The A4C Server

replies with a QoS-Authorization-Answer (QAA) to the QoS Broker. The answer contains the QoS profile, based on which the QoS Broker can grant or deny the requested QoS.

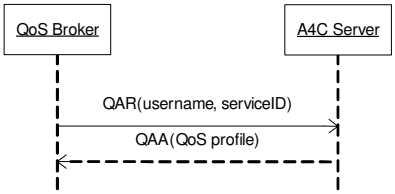


Figure 23: QoS authorization MSC

Retrieve generic user profile

The A4C Server stores the generic user profile of every user, which specifies user details. Akogrimo Components, e.g. Base VO Manager or Context Manager, can retrieve the user profile from the A4C Server by sending a User-Profile-Request (UPR) as shown in Figure 24. The request contains the username as a reference of the user. The A4C Server gets the user profile from the A4C Database and sends it to the SAML Authority in a SAML-Profile-Request. The SAML Authority includes the attributes of the profile into SAML and sends back the profile containing SAML attributes to the A4C Server in the SAML-Profile-Answer message. The A4C Server forwards the profile in SAML in the User-Profile-Answer (UPA) message.

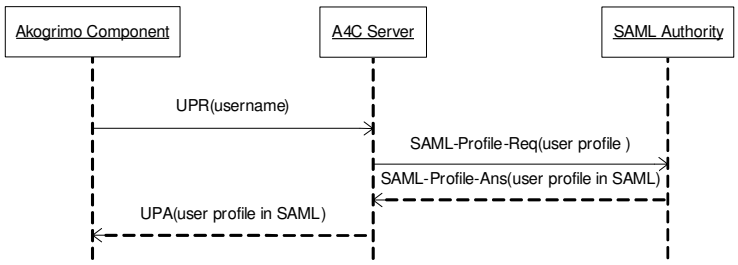


Figure 24: Retrieve user profile MSC

4.3.3. Accounting

The A4C Server controls and manages the accounting process. It collects and stores accounting records, which describe the usage data for a certain session. An accounting session is related to the service usage of the user. Usage data is gathered on different components by metering network traffic and user activities. The Akogrimo components which participate in the accounting process and are responsible for metering service usage are the Access Router (AR) and the Grid Metering Component.

The messages related to accounting are shown in Figure 25. The figure only shows the AR but the same applies for the Grid Metering Component. The Accounting-Request (ACR) Diameter message is used to send accounting records to the A4C Server. The Accounting-Answer (ACA) message contains the response of the A4C Server.

Two different kind of accounting types can be differentiated. If the service has a measurable length, the AR has to send a Start Record at session start in the first Accounting-Request and a Stop Record on session termination in the last Accounting-Request. The Start Record is used to initiate an accounting session. The Stop Record is sent to terminate an accounting session and contains the cumulative accounting data relevant to the session. The A4C Server determines whether to send additional Interim Records. In this case the AR has to send Interim Records periodically in a certain interval specified by the A4C Server. The Interim Record contains cumulative accounting data for an existing accounting session. If the service is a one-time event, meaning that the start and stop of the event are simultaneous, then the AR sends Event Records to the A4C Server. The Event Record contains all accounting data relevant to the service.

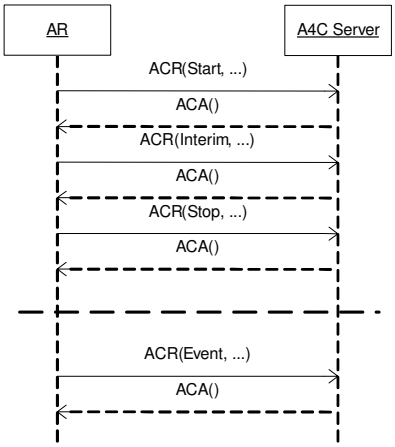


Figure 25: Accounting MSC

4.3.4. Auditing

Auditing can be defined as an examination of Audit Trails to ensure compliance with pre-established procedures, agreements, policies, contracts. In the first phase of the Akogrimo Project, the auditing function offered by the A4C will be used by service monitoring components to report SLA violation events during service provisioning. Later, during charging calculation, these violations will be checked and penalties will be applied according to the negotiated SLA for the service that was monitored. Whenever an SLA violation is detected by a monitoring component a Violation-Report-Request (VRR) message is sent to the A4C Server in order to report the violation. Included in the message is the type of violation that was detected and the service session for which this violation was reported. Possible violation types for a service are stored in the A4C Server. The A4C Server responds with a Violation-Report-Answer (VRA).

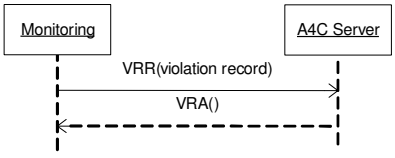


Figure 26: Auditing MSC

4.3.5. Charging

The A4C Server is responsible for the charge calculation of resource and service usage. Based on the accounting records collected and the charging specification stored in the A4C Server it calculates the charge for each service access and creates charging records. A charging record contains the result of the charge calculation and it includes attributes like the username, accessed service, domain where the service was provided, currency used, and the charge for the service.

To support service provisioning in a multi-domain environment where users receive a single bill from their home provider, the exchange of charging records between providers is required. Charging records are used both for the transfer of end user charges and for the exchange of settlement data between providers. Charging records are sent by A4C Servers between different domains as shown in Figure 27. The Charging-Transfer-Request (CTR) Diameter message is used to send a charging record to an A4C Server. The Charging-Transfer-Answer (CTA) message informs the originating A4C Server about the result of the charging record transfer.

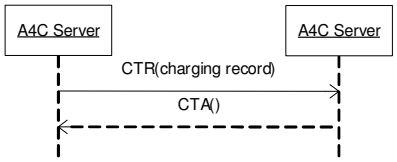


Figure 27: Charging MSC

4.3.6. Database

The A4C Server includes a database to store configuration data required for the A4C task, to manage authentication information, and to store the output of accounting and charging. The database contains user profiles, customer profiles, authentication information, accounting records, charging records, tariff information, and session details.

Within the A4C infrastructure users and customers are separated. Users are the entities which use the services, while customers are the entities which pay the bill. This separation of the usage and commercial aspects allows for a split between corporate users and non corporate users. In case of the personal user, the customer details are the same as the user details but for a corporate user the customer details would have the details of the company which pays for the services.

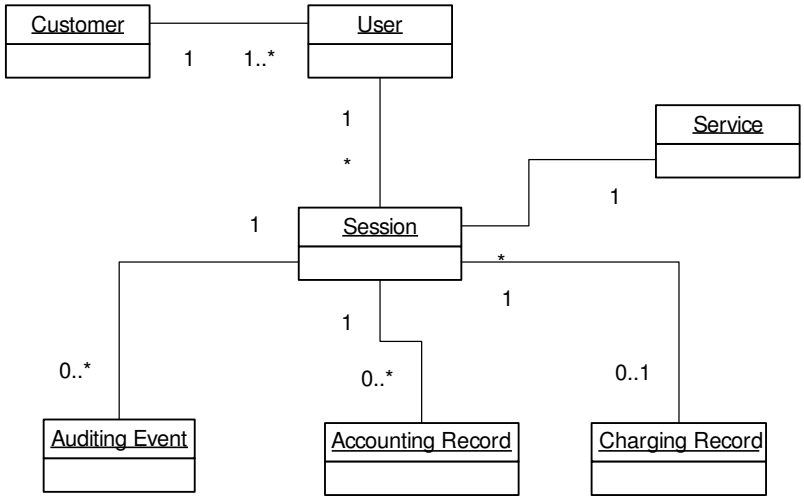


Figure 28: Data Model for A4C Database

Figure 28 shows the main objects that need to be stored in the database and their relation. For a complete list of object attributes and their data types please refer to Section 4.4.9.

The following is a short description of the objects in the data model and their usage in the A4C component:

- **User**
  Any user having a contract with a service provider needs to be present in the A4C database together with some extra information that identifies this user.
- **Customer**
  Any customer having a contract with a service provider needs to be present in the A4C database together with customer details and associated users.
- **Service**
  Every service provided by the service provider needs to be mapped in the A4C database and identified by a set of measurable parameters.
- **Session**
  The central object in the A4C data model is the A4C session, which is the mapping of a service session in the A4C database. Every session needs to be linked to the service that created this session and to the user for which the service session was started. The A4C sessions are represented in a hierarchical manner, fully showing the hierarchical relation between different service sessions.
- **Accounting_Record**
  An Accounting Record stores a number of parameters that define the service usage during a service session. Every accounting record needs to be linked to an A4C session, so that mapping between a service session and service usage parameters metered during that session will be possible during the charging process.
- **Auditing_Event**
  Any SLA violation detected by monitoring components will be stored in the A4C Server database for integration in the charge calculation of a service consumption

- Charging_Records
  Charging records store charging details related to a specific service session.

4.4. Implementation

The implementation description of the A4C infrastructure provides the internal architecture of the A4C Server, A4C Client, SAML Authority, and SAML Authority Client. Additionally, the implementation details of A4C tasks, structure of the A4C database, required software components and libraries are described.

4.4.1. A4C Server

The internal architecture of the A4C Server depicts the fine design of the A4C implementation prototype. The internal A4C Server architecture is shown in Figure 29. It consists of the following components:

- Diameter Library: It is responsible for the handling of the Diameter protocol. It acts as a message dispatcher and sends and receives Diameter messages. It is based on the OpenDiameter library [OPENDIAM].
- AA Component: It is responsible for authentication and authorization. It includes several subcomponents, i.e. NET Authn Handler, SIP Authn Handler, VO Authn Handler, User Profile Handler, SD AA Handler, and QoS Profile Handler, which provide various AA functions. It also includes the SAML Authority Client which enables a SOAP based communication with the SAML Authority.
- Accounting Component: It controls and manages the accounting process.
- Auditing Component: It is responsible for the handling of auditing event messages.
- Charging Component: It is responsible for charging and it supports post-paid charging based on a service-dependent and flexible tariff specification.
- A4C Database: It stores user profiles, authentication information, accounting and charging related configuration data, and accounting and charging records. It is a MySQL database [MySQL].
- Database Interface: It provides the communication with the A4C Database.
- Management Interface: It provides an interface for the management and configuration of the A4C Server for the network administrator.
- SAML Authority: It generates and manages SAML IDTokens required for the authentication process. It is a separate component but highly related to the A4C Server.

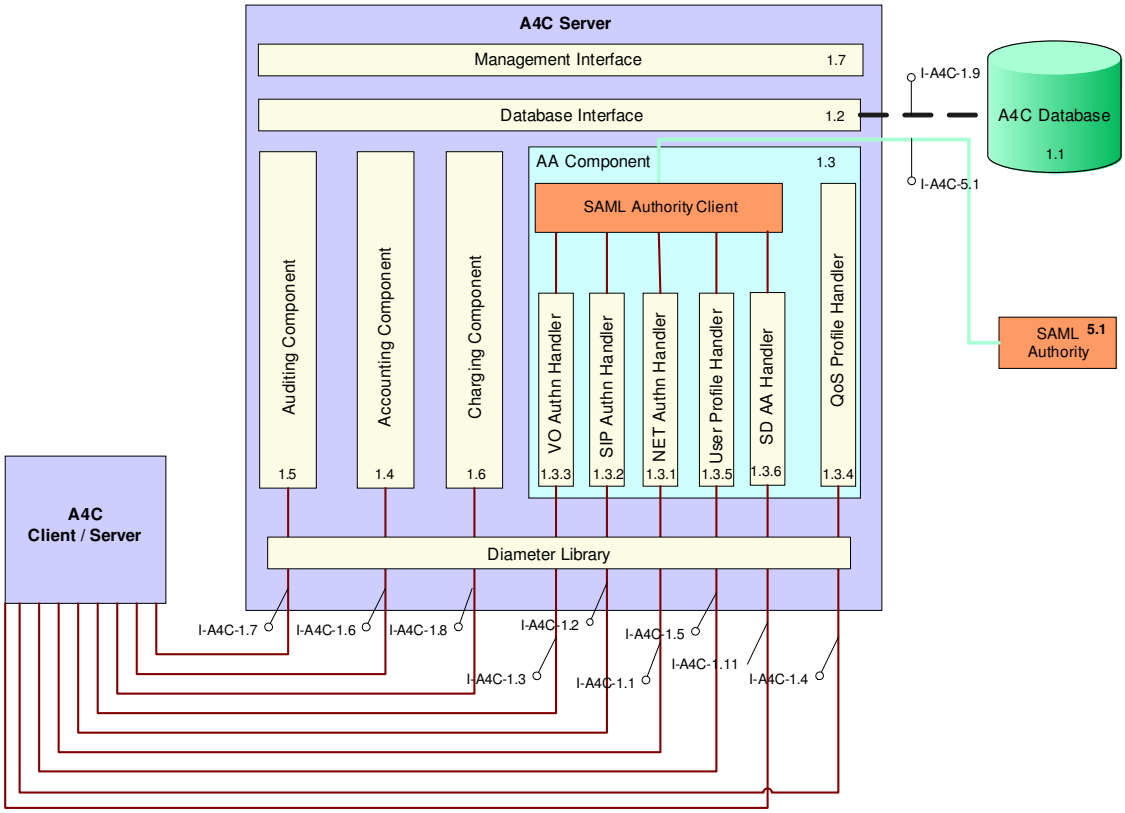


Figure 29: A4C Server Internal Architecture

4.4.2. A4C Client

The internal architecture of the A4C Client depicts the fine design of the A4C implementation prototype. The internal architecture of the A4C Client is shown in Figure 30. The A4C Client consists of the following components:

- Diameter Library: It is responsible for the handling of the Diameter protocol. It acts as a message dispatcher and sends and receives Diameter messages. It is based on the OpenDiameter library [OPENDIAM].
- AA Component: It is responsible for authentication and authorization. It includes several subcomponents, i.e. NET Authn Handler, SIP Authn Handler, VO Authn Handler, User Profile Handler, SD AA Handler, and QoS Profile Handler, which provide various AA functions. It communicates with the AA Component in the A4C Server.
- Accounting Component: It supports the accounting process and enables to send accounting records. It communicates with the Accounting Component in the A4C Server.
- Auditing Component: It supports the auditing process and enables to send auditing events. It communicates with the Auditing Component in the A4C Server.
- SAML Client: It provides the SAML assertion handling, e.g. the verification of its validity.
- A4C Client API: It provides the interface of the A4C Client library both in C++ and Java using JNI [JNI].

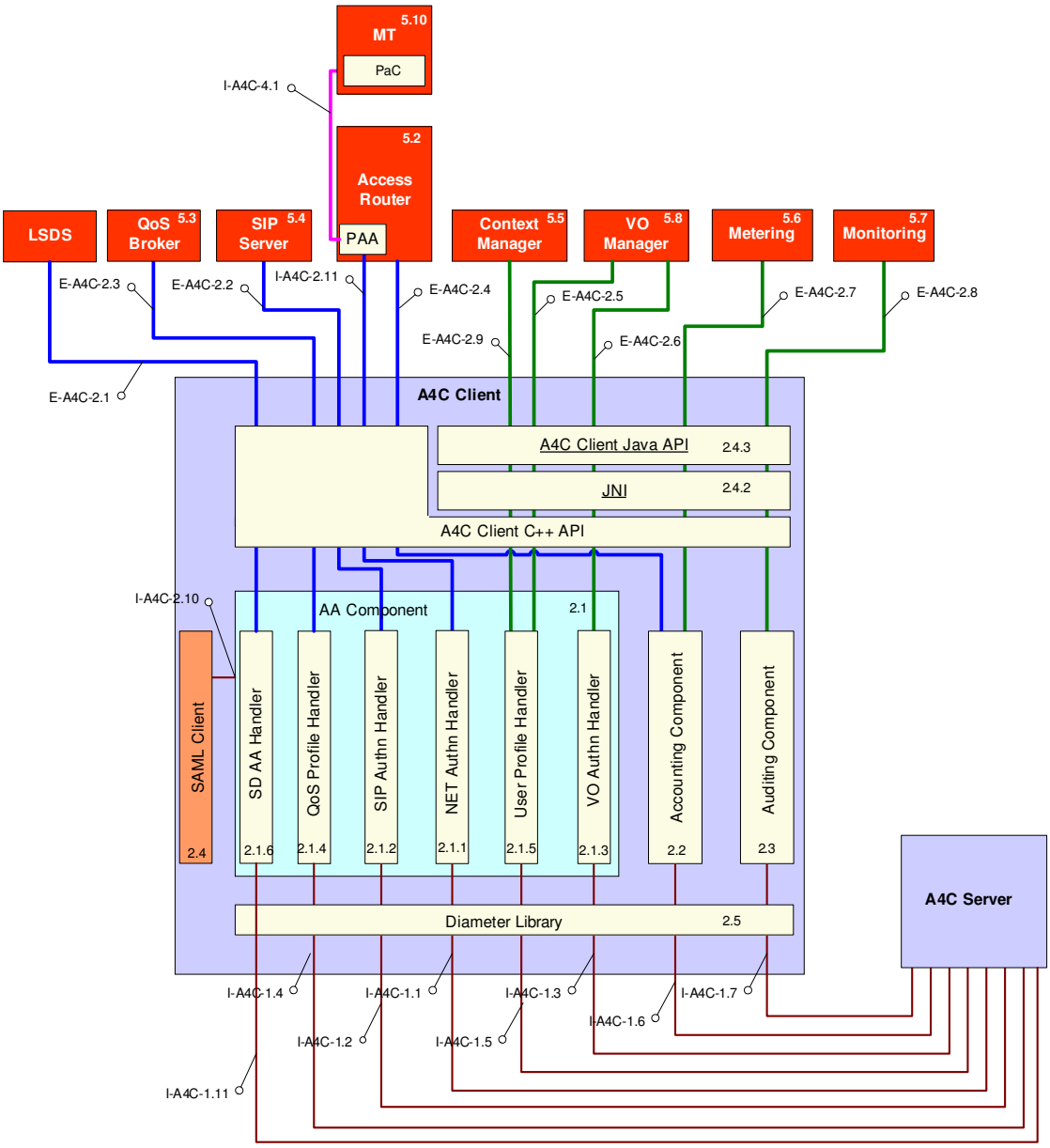


Figure 30: A4C Client Internal Architecture

4.4.3. SAML Authority

The SAML Authority is responsible for the creation of SAML-based authentication, authorization and attribute assertions and provides the functionality of creating, storing and managing SAML assertions and ID-Tokens. Figure 31 represents the internal architecture of the SAML Authority and the communication with the A4C Server.

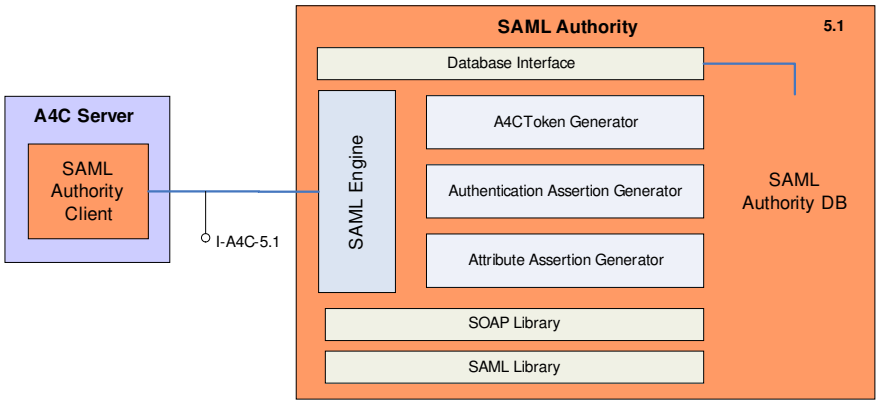


Figure 31: SAML Authority Internal Architecture

The components depicted in the SAML Authority are the following:

- SAML Engine: This is the main element in the SAML Authority. It is an application server that receives requests in the form of SOAP messages from the SAML Authority Client at the A4C Server. It understands the requests, resolves the result internally with the other components, and generates the SOAP Response with the result included at the body.
- IDToken Generator: This element creates the IDToken. It is initiated from the SAML Engine, when it receives an issue request for a user token.
- Authentication Assertion Generator: This element generates the authentication assertion when it is invoked from the SAML Engine.
- Attribute Assertion Generator: This element generates the attribute assertion when it is invoked from the SAML Engine.
- Database Interface: This is an interface used by the SAML Engine and also by the generators to access the database of the SAML Authority.
- SAML Authority DB: This element stores the IDToken and assertions of the users when requested by the SAML Engine or the generator entities.
- SOAP Library: This library is used for the understanding of SOAP Requests and the generation of the SOAP response. It is used by the SAML Engine.
- SAML Library: This library is used by the SAML Engine and the generator entities, in order to create SAML messages.

4.4.4. SAML Authority Client

The SAML Authority Client, a component inside the A4C Server, provides the communication with the SAML Authority and the A4C Server. Since the SAML Authority is a SOAP Server, the SAML Authority Client needs to provide requests in the SOAP protocol. This is the basic functionality of this client. The internal architecture of the SAML Authority Client at the A4C Server is depicted in Figure 32.

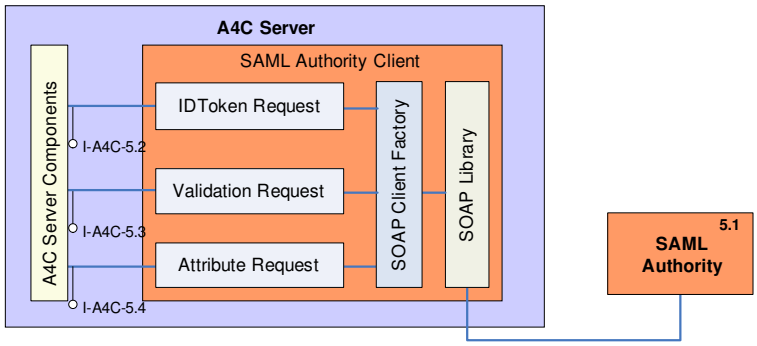


Figure 32: SAML Authority Client Internal Architecture

The SAML Authority Client at the A4C includes the following components:

- SOAP Client Factory: This element is a factory of SOAP Requests. It is the main element in the SAML Authority Client.
- IDToken Request: This element supports the creation of a SOAP Request to ask for an IDToken.
- Validation Request: This element supports the creation of a SOAP Request to ask for validation and authentication assertion issuance.
- Attribute Request: This element supports the creation of a SOAP Request to ask for attribute assertion issuance.
- SOAP Library: This library provides the support for building SOAP Messages.

4.4.5. Authentication and Authorization

The authentication and authorization function of the A4C Server consists of several independent tasks. It provides means for network authentication, for the IDToken based authentication, for QoS authorization, and for user profile transfer.

Network authentication

The network authentication of A4C infrastructure is implemented based on the libeap and libpana libraries of the Opendiameter implementation. The Diameter-EAP-Request (DER), Diameter-EAP-Answer (DEA), and PANA messages are extended to carry the IDToken in the response of the A4C Server (cf. Figure 20). The network authentication function is implemented as a separate Diameter application.

IDToken based authentication

The IDToken based authentication is implemented using the AA-Request (AAR) and AA-Answer (AAA) Diameter messages. The messages are extended to carry the identifier of the requested service and the IDToken.

QoS authorization

For the QoS authorization the QoS-Authorization-Request (QAR) and QoS-Authorization-Answer (QAA) Diameter messages are newly defined. The answer carries the QoS profile. The QoS authorization is implemented as a separate Diameter application.

Retrieve generic user profile

To retrieve the generic user profile the User-Profile-Request (UPR) and User-Answer-Profile (UPA) Diameter messages are newly defined. This function is implemented as a separate Diameter application. The profile is transmitted as a SAML assertion.

4.4.6. Accounting

For the accounting of network services and grid applications the Accounting-Request (ACR) and Accounting-Answer (ACA) Diameter messages are used. The messages are extended with additional AVPs to carry accounting parameters for grid services. The specified accounting AVP are summarized in Section 4.4.6.1.

To support grid and compound services an extended session model is implemented to enable the definition of a session hierarchy. The accounting messages are extended with the parent-session-ID parameter to specify the parent session within the hierarchy.

4.4.6.1. Accounting Parameters

Accounting parameters describing the service usage are specified by AVPs in the Diameter protocol. For network services the AVPs defined for the Diameter network access application [Cal05] are used. Table 19 summarizes the most important parameters.

Table 19 Accounting parameters for network services

AVP	Data type	Description
Accounting-Input-Octets	Unsigned64	Contains the number of octets received from the user.
Accounting-Output-Octets	Unsigned64	Contains the number of packets received from the user.
Accounting-Input-Packets	Unsigned64	Contains the number of packets received from the user.
Accounting-Output-Packets	Unsigned64	Contains the number of packets sent to the user.
Acct-Session-Time	Unsigned32	Defines the length of the current session in seconds.
Event-Timestamp	Time	Indicates the time the reported event occurred.
NAS-Port-Type	Enumerated	Defines the type of the network access provided to the user. Possible values include ISDN, ADSL, Ethernet, Wireless 802.11, and UMTS. The complete list can be found in [IANA05].

There are also additional AVPs specified according to the requirements of Akogrimo. These AVPs enable to account for grid services and extends the attributes for network service accounting. Table 20 summarizes the additional AVPs defined.

Table 20 Akogrimo accounting parameters for network and grid services

AVP	Data type	Description
CPU-Time	Unsigned32	Defines the duration of CPU usage in seconds.
CPU-Type	Enumerated	Indicates the CPU type used. The following values are defined for this AVP: 0 – x86 32 bit 1 – x86 64 bit 2 – PPC 32 bit 3 – PPC 64 bit 4 – SPARC 32 bit 5 – SPARC 64 bit
CPU-Cycles	Unsigned64	Indicates the number of CPU cycles used by the process. The measurement unit is 1000 cycles, thus each number n in the AVP stands for n*1000 cycles.

AVP	Data type	Description
CPU-Count	Unsigned32	Specifies the number of CPUs used.
Node-Count	Unsigned32	Specifies the number of nodes used.
Memory-Size	Unsigned64	Specifies the size of the main memory of the node in bytes.
Memory-Usage-Average	Unsigned64	Specifies the average size of memory used in bytes.
Memory-Usage-Maximum	Unsigned64	Specifies the maximum size of memory used in bytes.
Disk-Usage-Average	Unsigned64	Specifies the average size of disk storage used in bytes.
Disk-Usage-Maximum	Unsigned64	Specifies the average size of disk storage used in bytes.
Host-Name	UTF8String	Specifies the hostname of the node.
Job-Name	UTF8String	Specifies the name of the job executed.
Process-ID	Unsigned32	Specifies the process identifier.
Process-Status	Enumerated	Specifies the status of a job. The following values are defined for this AVP: 0 – aborted 1 – completed 2 – failed 3 – held 4 – queued 5 – started 6 – suspended
QoS-Bandwidth	Unsigned64	Specifies the transmission speed provided in bit/s.
QoS-Delay	Unsigned32	Specifies the delay guarantee provided in ms.
QoS-Jitter	Unsigned32	Specifies the jitter guarantee provided in ms.
QoS-Priority	Unsigned32	Specifies the priority class of the traffic.
QoS-DSCP	Unsigned32	Specifies the DiffServ Code Point used.
QoS-Score	Unsigned32	Specifies a relative value for the QoS provided.
Accounting-Dropped-Octets	Unsigned64	Contains the number of octets dropped by the access network.
Accounting-Dropped-Packets	Unsigned64	Contains the number of packets dropped by the access network.
Request-Count	Unsigned32	Number of requests
Successful-Request-Count	Unsigned32	Number of successful requests
Failed-Request-Count	Unsigned32	Number of failed requests

4.4.7. Auditing

To transfer auditing events the Violation-Report-Request (VRR) and Violation-Report-Answer (VRA) Diameter messages are newly defined. The auditing function is implemented as a separate Diameter application. For the description of auditing events new AVPs are defined.

The auditing function will be implemented in phase 2.

4.4.8. Charging

To transfer charging records the Charging-Transfer-Request (CRT) and Charging-Transfer-Answer (CTA) Diameter messages are newly defined. Additional AVP are defined for the charging purpose. Charging is implemented as a separate Diameter application.

The charging component of the A4C Server supports post-paid charging, and calculates charges per service based on the accounting data collected by the accounting component

and stored in the database according to the charging specification defined in Section 4.4.8.1.

4.4.8.1. Charging Specification Format

The charging specification defines the rules and prices to be applied for charging and it enables to specify user, service, and domain-specific tariffs. This means that the charging specification defines how a certain user in a given domain will be charged for a particular service. Different services have different charging specifications according to the charging requirements of the service. Different users can have different charging specifications, but users can be also mapped to the same specification, enabling to define groups of users and to apply the same charging for them. Different domains will have most probably different pricing specifications but the same specification might be also applied. The mapping for the charging specification is stored in the userTariffs table in the A4C database specified in Section 4.4.9.

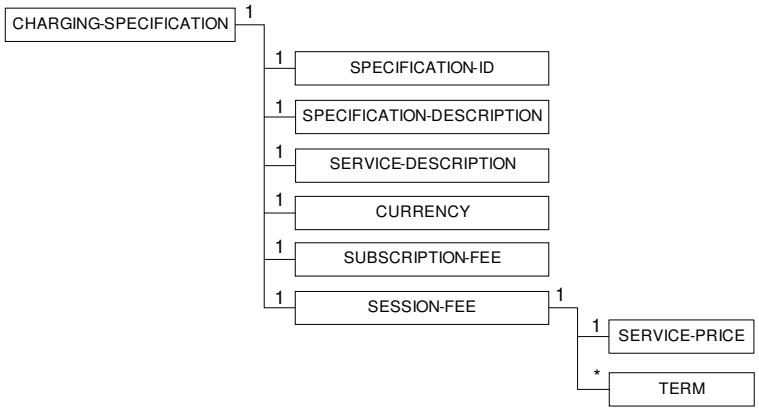


Figure 33: Tariff specification

The charging specification is defined in a file in XML format. The basic structure of the XML file is shown in Figure 33 and it consists of the following elements:

- SPECIFICATION-ID tag: It defines the unique identifier of the charging specification.
- SPECIFICATION-DESCRIPTION tag: It allows for a description of the charging specification.
- SERVICE-DESCRIPTION tag: It allows for a description of the service that the charging is applied for.
- CURRENCY tag: It specifies the currency to be used for billing.
- SUBSCRIPTION-FEE tag: It specifies a fixed subscription fee for the particular service.
- SESSION-FEE tag: It specifies the charging rules and prices to be applied for a service session. It consist of one service price and one or more terms.
    - SERVICE-PRICE tag: It specifies a fix price to be applied for every usage of the particular service.
    - TERM tag: It specifies one element of the charge calculation.

The subscription fee defines a monthly fee for the service. During the charging process for every service session a charge (C_S) is calculated as the sum of the service price (P_S) and the sum of all terms (T_i):

$$C_S = P_S + \sum_i T_i$$

A term specifies an element of the charge calculation, depending on some accounting attributes and a price. The accounting attributes are specified as Diameter AVPs. The AVPs defined in Section 4.4.6.1 can be used to specify the accounting parameter to be taken for the charge calculation. The price can be constant, i.e. a fixed price for every unit consumed, or variable depending on some conditions, e.g. the amount of consumption or the time of the consumption (time-of-day). A term is in general the product of one or more AVPs (AVP_k) and a price (P):

$$T_i = \prod_k AVP_k \cdot P$$

The structure of a term applying a constant price is shown in Figure 34. It consists of the following elements:

- AVP tag: It refers to the accounting attribute (AVP) relevant for charging. This tag can be included several times.
  - AVP-ID tag: It defines the identifier of the AVP.
  - AVP-NAME tag: It defines the name of the AVP.
- CONSTANT-PRICE tag: It defines a constant price per unit applied for charging in this term.

As an example, if *Acct-Session-Time AVP* is defined in the AVP tag, the price is calculated according to the time the service was used. The CONSTANT-PRICE tag specifies the price for a second in this case.

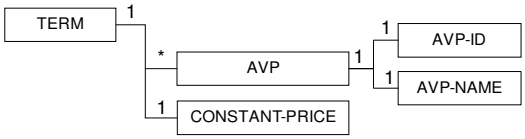


Figure 34: Constant price specification

If the price to be applied is variable, it depends on the value of a particular AVP. Two cases can be differentiated. In the first case, the price is on the basis of a certain value of the AVP. In the second case, the price is on the basis of the value of the AVP being in numerical range.

The charging specification for variable prices is shown in Figure 35. It consists of the following elements:

- AVP tag: It has the same meaning as in the constant price case. It can be included several times.
- VARIABLE-PRICE tag: It defines the condition and the charging rule in case of a variable price.

- BASIS-AVP tag: It defines the AVP that is used to determine the price to be applied.
- CASE tag: It groups the condition and the price together.
  - AVP-VALUE tag: It defines a certain value of the AVP. If the AVP has this value, the price defined in the CONSTANT-PRICE tag will be applied. If the tag is empty, it represents the default value.
  - CONSTANT-PRICE tag: It defines the price to be applied.
  - DESCRIPTION tag: It contains an optional description for the condition.

As an example, if the BASIS-AVP is specified as the *CPU-Type AVP*, different prices can be defined for each CPU type. The AVP-VALUE could be e.g. x86 32 bit or x86 64 bit.

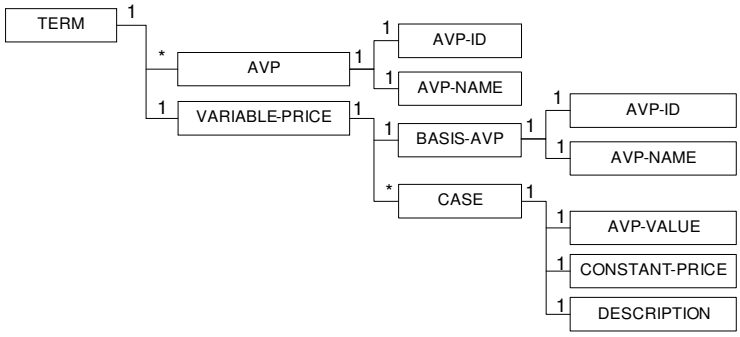


Figure 35: Variable price specification

The charging specification for variable prices defined by ranges is shown in Figure 36. It consists of the following elements:

- AVP tag: It has the same meaning as in the constant price case. It can be included several times.
- VARIABLE-PRICE tag: It defines the condition and the charging rule in case of a variable price.
  - BASIS-AVP tag: It defines the AVP that is used to determine the price to be applied.
  - RANGE tag: It groups the condition and the price together in case of a range.
    - AVP-VALUE-MIN tag: It defines the minimum value of the AVP. If the value of the AVP is between AVP-VALUE-MIN and AVP-VALUE-MAX, the price defined in the CONSTANT-PRICE tag will be applied. It the tag is empty, the range has no lower limit.
    - AVP-VALUE-MAX tag: It defines the maximum value of the AVP. If the value of the AVP is between AVP-VALUE-MIN and AVP-VALUE-MAX, the price defined in the CONSTANT-PRICE tag will be applied. It the tag is empty, the range has no upper limit.
    - CONSTANT-PRICE tag: It defines the price to be applied.
    - DESCRIPTION tag: It contains an optional description for the condition.

As an example, the BASIS-AVP might be *Accounting-Input-Octets AVP* and the ranges defines different prices for the amount of data transferred, e.g. below 100MB, between 100MB and 1GB, and above 1GB.

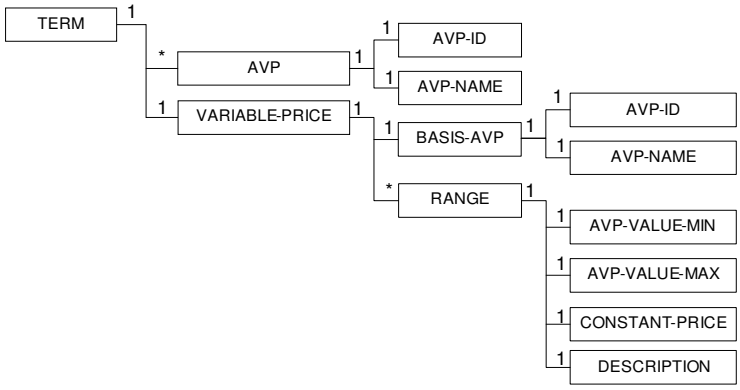


Figure 36: Variable price specification with ranges

The charging component enables a generic charging specification to be linked to the user's subscription, which is also stored in a XML file similar to the service related fees. The subscription file specifies a subscription fee, which is applied for every bill, i.e. every month, and it can also contain some special discounts for the final bill. The mapping is also stored in the userTariffs table with serviceId=0.

As for every service a separate charging specification can be applied, also the charging for compound services, consisting of several sub-services, can be specified. In this case each sub-service has an own charging specification, which is applied for the sub-service, and finally all charges for the sub-services are added up, resulting in the final charge for the service. The charging specification can be defined also per domain, allowing charging for compound services executed in different domains.

4.4.9. Database Structure

The database structure with the tables specified is summarized in Figure 37. In the following the database tables and their fields are explained in detail.

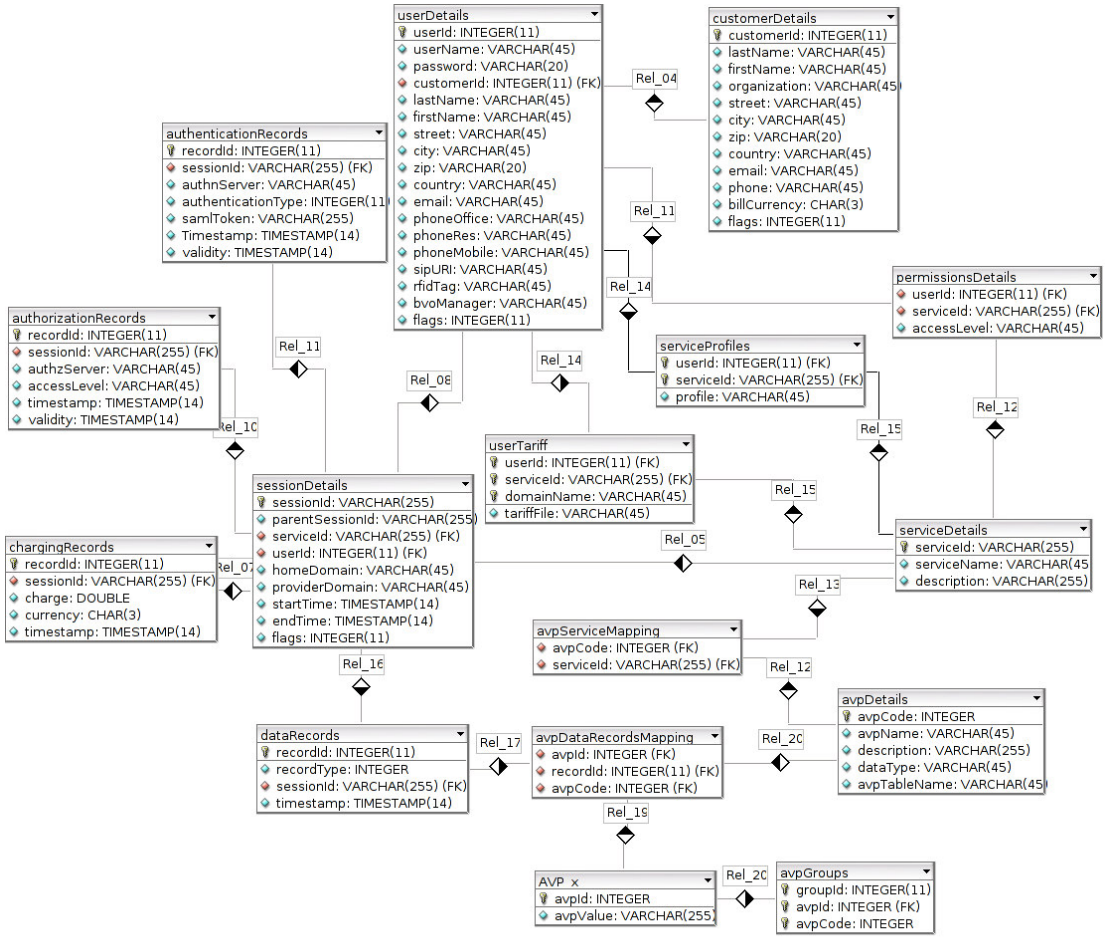


Figure 37: A4C database structure

User and customer details

The data fields of the userDetails and customerDetails tables are shown in Table 21 and Table 22 respectively. They include generic attributes like username, password, and personal details.

Table 21 Data fields of the userDetails table

Data Field	Data Type	Description
userId	Integer	Database internal identifier of the user
username	Varchar(45)	Username in user@domain (NAI) format
password	Varchar(20)	Password of the user
customerId	Integer	The customer identifier of the user
lastName	Varchar(45)	Last name
firstName	Varchar(45)	First name
street	Varchar(45)	Street
city	Varchar(45)	City
zip	Varchar(20)	Zip code
country	Varchar(45)	Country
email	Varchar(45)	Email address
phoneOffice	Varchar(45)	Phone number office
phoneRes	Varchar(45)	Phone number residence
phoneMobile	Varchar(45)	Phone number mobile
sipURI	Varchar(45)	Default SIP address of the user
rfidTag	Varchar(45)	RFID tag associated to the user

Data Field	Data Type	Description
bvoManager	Varchar(45)	Reference to the BVO Manager
flags	Integer	Internal flags for management purposes

Table 22 Data fields of the customerDetails table

Data Field	Data Type	Description
customerId	Integer	Database internal identifier of the customer
lastName	Varchar(45)	Last name
firstName	Varchar(45)	First name
organization	Varchar(45)	Organization
street	Varchar(45)	Street
city	Varchar(45)	City
zip	Varchar(20)	Zip code
country	Varchar(45)	Country
email	Varchar(45)	Email address
phone	Varchar(45)	Phone Number
billCurrency	Char(3)	Currency used for billing
flags	Integer	Internal flags for management purposes

Service details and profiles

The serviceDetails table shown in Table 23 stores available services, specified by a service identifier. Additionally, each service has a name and an optional description.

Available AVPs used in Diameter messages are listed in the avpDetails table (Table 24). Each AVP is defined with its Diameter AVP code, AVP name, and the data type of the AVP. Each type of AVP is stored in a separate table (cf. Table 33), which is specified by the avpTableName field.

The relation between services and AVPs are defined in the avpServiceMapping table (Table 25), which specifies the AVPs used for a particular service. The serviceProfiles table (Table 26) lists for every user the enabled services with the service profile. The profile defines the network QoS parameters enabled for the user.

Table 23 Data fields of the serviceDetails table

Data Field	Data Type	Description
serviceId	Varchar(255)	Identifier of the service
serviceName	Varchar(45)	Name of the service
description	Varchar(255)	Description of the service

Table 24 Data fields of the avpDetails table

Data Field	Data Type	Description
avpCode	Integer	Diameter AVP code
avpName	Varchar(45)	Name of the AVP
description	Varchar(255)	Description of the AVP
dataType	Varchar(45)	Data type of the AVP
avpTableName	Varchar(45)	Name of the database table storing this type of AVP

Table 25 Data fields of the avpServiceMapping table

Data Field	Data Type	Description
avpCode	Integer	Reference to the AVP

Data Field	Data Type	Description
serviceId	Varchar(255)	Reference to the service

Table 26 Data fields of the serviceProfiles table

Data Field	Data Type	Description
userId	Integer	Reference to the user
serviceId	Varchar(255)	Reference to the service
profile	Varchar(45)	File name of the profile

Session details

The sessionDetails table (Table 27) is used to store all A4C session related information. An A4C session is created for authentication, authorization, accounting, auditing and charging tasks. The session is identified with a globally unique session identifier, specified by the Diameter protocol. The session might have a parent session if it is part of a compound service with a session hierarchy. The session is associated to a user, a service and to a provider domain. It has a start and an end time. The flags field is used for internal tasks, like specifying the session type.

Table 27 Data fields of the sessionDetails table

Data Field	Data Type	Description
sessionId	Varchar(255)	Session identifier
parentSessionId	Varchar(255)	Session identifier of the parent session
serviceId	Varchar(255)	Reference to the service
userId	Integer	Reference to the user
homeDomain	Varchar(45)	Home domain of the user
providerDomain	Varchar(45)	Domain of the service provider
startTime	Timestamp	Start time of the session
endTime	Timestamp	End time of the session
flags	Integer	Internal flags, e.g. type of the session (authentication, accounting etc.)

Authentication records

The authenticationRecords table (Table 28) stores authentication events associated to a session. It specifies the server authenticated the user and the type of authentication. It also has a timestamp and a validity field.

Table 28 Data fields of the authenticationRecords table

Data Field	Data Type	Description
recordId	Integer	Identifier of the record
sessionId	Varchar(255)	Reference to the session
authnServer	Varchar(45)	Diameter identity of the server authenticating the user
authenticationType	Integer	Type of the authentication, e.g. password based, certificate based
samlToken	Varchar(255)	The SAML IDToken
timestamp	Timestamp	Timestamp of the record
validity	Timestamp	Validity of the authentication

Authorization records

The authorizationRecords table (Table 29) stores authorization events associated to a session. It specifies the server authorized the user and the access permissions granted for the user. It also has a timestamp and a validity field. The permissionDetails table (Table 30) specifies possible access rights per user and service.

Table 29 Data fields of the authorizationRecords table

Data Field	Data Type	Description
recordId	Integer	Identifier of the record
sessionId	Varchar(255)	Reference to the session
authzServer	Varchar(45)	Diameter identity of the server authorizing the user
accessLevel	Varchar(45)	Applied access permission
timestamp	Timestamp	Timestamp of the record
validity	Timestamp	Validity of the authorization

Table 30 Data fields of the permissionDetails table

Data Field	Data Type	Description
userId	Integer	Reference to the user
serviceId	Varchar(255)	Reference to the service
accessLevel	Varchar(45)	Access permission

Accounting and auditing records

Accounting and auditing records are mapped to several tables in the database. The dataRecords table (Table 31) stores the records with the record identifier, the type of the record, the reference to the related session, and the timestamp. The content of the records, i.e. the AVPs, are stored in the AVP_x tables. Every AVP has a separate AVP_x table, where x is the Diameter code of the AVP. The mapping between records and AVPs is realized by the avpDataRecordsMapping table (Table 32), which associates the recordId with avpId and avpCode. The avpGroups table (Table 34) is used to map grouped AVPs.

Table 31 Data fields of the dataRecords table

Data Field	Data Type	Description
recordId	Integer	Identifier of the record
recordType	Integer	Type of the record, 0 = accounting, 1 = auditing
sessionId	Varchar(255)	Reference to the session
Timestamp	Timestamp	Timestamp of the record

Table 32 Data fields of the avpDataRecordsMapping table

Data Field	Data Type	Description
recordId	Integer	Reference to the record
avpId	Integer	AVP index in AVP_x table
avpCode	Integer	Diameter AVP code

Table 33 Data fields of the AVP_x table

Data Field	Data Type	Description
avpId	Integer	Database internal index
avpValue	According to the AVP type	Value in the AVP

Table 34 Data fields of the avpGroups table

Data Field	Data Type	Description
groupId	Integer	Identifier of the group
avpId	Integer	Database internal index
avpCode	Integer	Reference to the AVP

Tariffs and Charging records

The userTariffs table (Table 35) specifies the tariff file associated to a certain user, service and domain. The tariff file contains the charging specification as specified in Section 4.4.8.1. The chargingRecords table (Table 36) stores data related to charging records.

Table 35 Data fields of the userTariffs table

Data Field	Data Type	Description
userId	Integer	Reference to the user
serviceId	Varchar(255)	Reference to the service
domainName	Varchar(45)	Name of the domain providing the service
tariffFile	Varchar(45)	File name of the tariff specification

Table 36 Data fields of the chargingRecords table

Data Field	Data Type	Description
recordId	Integer	Identifier of the record
sessionId	Varchar(255)	Reference of the session
Charge	Float	Charge for the session
Currency	Char(3)	Currency used for the record
timestamp	Timestamp	Timestamp of the record

4.4.10. Database Interface

The database interface provides access and control over the information stored in the database. The interface is used as an internal interface in the A4C Server. The interface specifies basic functions that can be combined to provide more sophisticated and complete functionality. The database interface provides access to an administrator for administrative purposes, like adding new users, and it also enables internal components of the A4C Server to get or store information in the database. The following main functions of the interface are identified and specified:

- Connect to the database.
- Add a new user, deactivate a user, and manage user details.
- Add a new customer, deactivate a customer, and manage customer details.
- Add, remove, and manage tariff profiles.
- Add a new session and manage session details.
- Add a new authentication and authorization record and manage details.
- Add a new accounting record and manage accounting parameters.
- Add a new charging record and manage accounting parameters.
- Add a new auditing record and manage auditing parameters.

- Retrieve the list of users and user details.
- Retrieve customer details.
- Retrieve sessions of a particular user.
- Retrieve details of a particular session.
- Retrieve the charging specification for a particular user for a service in a particular domain.
- Retrieve accounting records for a particular session.
- Retrieve charging records.
- Retrieve auditing records.
- Manage accounting parameters and details for a service.
- Manage AVP details.

4.4.11. Software Components and Libraries

Table 37 summarizes the software components and libraries required for the A4C and the SAML Authority.

Table 37 Software components and libraries

Software components and libraries	Version	Purpose	Remark
Opendiameter	v1.0.7-f	Diameter protocol handler	Provides the following libraries: libdiamparser, libdiameter, libeap, libeaparchie, libpana, libdiametereap, libdiameternasreq, libdiametermip4, libodutl.
Xerces C++	2.6.0	XML parser	Required by Opendiameter and the charging component.
ACE library	>5.4.1	Adaptive Communication Environment	Required by Opendiameter. Compiled with IPv6 and SSL support
Boost library	1.0.3	Parser	Required by Opendiameter.
OpenSSL		SSL functionality	Required by Opendiameter.
OpenSAML	1.1	Build SAML messages	OpenSAML is an open source library that is fully consistent with the SAML 1.1 specifications. It lets an application use SAML messages or SAML application profiles to carry security Information.
Axis	1.2	Build SOAP messages	Axis is an opens source library fully consistent with the SOAP protocol specification of W3C.
XMLsec	1.1	XML Security support	It supports implementation of major XML security standards, such as XML Signature and XML Encryption. They provide security functionality for XML data.

Software components and libraries	Version	Purpose	Remark
MySQL	4.0	Database of the A4C Server and SAML Authority	
mysql-connector-java	3.0.15	Database connector	This library is a connector for java programmers to mysql database. It is used to write and read from the database.
MySQL++	2.0.5	MySQL C++ API	The library is used by the A4C Server to access the database.

4.5. Testing

Several test cases have been developed to test the functionality provided by the A4C components, as described in the following.

4.5.1. Network Authentication

The network authentication test case is used to test the initial user authentication function of the Akogrimo platform. The initial network authentication test is performed using a sample PANA authentication, the Client (PaC) receives a username and a password and then asks the A4C Server to perform authentication based on the received credentials.

4.5.1.1. Successful Authentication

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database with the password *AkoPass*
- There is IP-based communication between the mobile terminal and the A4C Server

Input data:

- Username: *AkogrimoUser@akogrimo.org*; password: *AkoPass*

Expected results:

- A result code saying the user was authenticated
- A new file on the mobile terminal (*/etc/IDToken*) containing the generated IDToken for the user

4.5.1.2. Authentication Failure

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database with the password *AkoPass*
- There is IP-based communication between the mobile terminal and the A4C Server

Input data:

- Username: *AkogrimoUser@akogrimo.org*; invalid password

Expected results:

- A result code saying the user was not authenticated

4.5.2. IDToken-based Authentication

The IDToken verification test case is used to test the capabilities of confirming an authentication based on an IDToken. A sample A4C Client uses the IDToken stored in */etc/IDToken* and asks the A4C Server to check the validity of the token.

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database
- User *AkogrimoUser* was authenticated using the network authentication mechanism
- The IDToken generated during network authentication is stored in */etc/IDToken*

Input data:

- Username: *AkogrimoUser@akogrimo.org*; IDToken from */etc/IDToken*

Expected results:

- A result code saying the user was authenticated

4.5.3. QoS Authorization

The QoS authorization test case is used to test the capabilities of the A4C infrastructure to deliver the QoS profile of a user to the QoS Broker.

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database and has a QoS Profile stored in the database.

Input data:

- Username: *AkogrimoUser@akogrimo.org*

Expected results:

- An XML document containing the user's QoS Profile

4.5.4. Generic Profile Request

The generic profile request test case is used to test the capabilities of the A4C infrastructure to deliver the generic user profile of a given user.

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database and has his generic user profile stored in the database

Input data:

- Username: *AkogrimoUser@akogrimo.org*

Expected results:

- An XML document containing the user's generic profile

4.5.5. Accounting

The accounting test-case is used to test the capabilities of the A4C infrastructure to deliver accounting messages. The test is performed by using a sample accounting client that initiates an accounting session with an A4C Server and then sends a custom accounting record to the A4C Server.

Initial conditions:

- User *AkogrimoUser* is registered in the A4C database

Input data:

- Username: *AkogrimoUser@akogrimo.org*, ServiceID: *NetworkTransfer*
- Input-Octets: *100000,* Output-Octets: *50000*

Expected results:

- A result code saying the transfer of the accounting data was successful
- The accounting record is stored in the database

4.5.6. Auditing

The auditing test case is used to test the auditing functionality of A4C, i.e. the capability to transport and store auditing records. A sample application that simulates an SLA violation called "End-to-End-Delay-Violation" uses an A4C Client to send an auditing record about this violation.

Initial conditions:

- The EventID *End-to-End-Delay-Violation* is present is declared in the Diameter dictionary

Input data:

- Event ID: *End-to-End-Delay-Violation*
- Event body: *800 ms*
- Accounting session: a string identifying an accounting session to which the auditing record belongs

Expected results:

- A result code saying whether the auditing record was successfully received.
- The auditing record is stored in the database.

4.5.7. Charging

The charging test cases are used to test the charging functionality of A4C.

4.5.7.1. Charge Calculation

The charge calculation test case is used to test the charge calculation for a service based on accounting records.

Initial conditions:

- A service has been offered by a service provider to a service user. Accounting records have been sent and stored by the A4C Server.

Input data:

- Accounting records: *25 x 10 kB data download, 14 x 10 kB data upload*
- Tariff: *1 ₮ per 10 kB data download, 2 ₮ per 10 kB data upload*

Expected results:

- The charge resulting from applying the tariff to the accounting records.

4.5.7.2. Charging Record Transfer

The charging record transfer test case is used to test the charging record exchange between A4C Servers.

Initial conditions:

- Charging records are created and stored in the foreign A4C Server.

Input data:

- Username: *AkogrimoUser@akogrimo.org*
- Charging record ID

Expected results:

- The charging record is successfully transferred to the home A4C Server of the user.
- The charging record is stored in the database.

5. Context Manager

5.1. Requirements

This section contains a brief listing and description of requirements related to the Context Manager. The objective is to emphasize what should (and what should not) be implemented, and thereby focusing the scope of the development process. Moreover, requirements will be useful during testing activities.

Requirements are grouped as functional (what functions should the component provide) and non-functional (how should functions be provided). For each requirement, a phase indication is used to specify if a requirement should be realised in phase 1 (within PM 16) or in phase 2 (PM n, beyond PM 19).

Table 38 Context Manager – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**Gather context (end user information)**	**X**	**X**
F 1.1	Presence	X	
F 1.2	Position	X	
F 1.2.1	Position – RFID based	X	
F 1.2.2	Position – based on e.g. GPS or WLAN		(X)
F 1.3	Terminal capabilities for users terminal	X	
F 1.4	Discover services in user's proximity	X	
F 1.4.1	Terminal-based discovery (discovery performed by user's terminal)		(X)
F 1.4.2	Centralised, directory-based discovery (match users position with services for that position)	X	
F 2	**Distribute context (to context consumers)**	**X**	
F 2.1	Offer subscription for user context	X	
F 2.1.1	Possibility to limit context scope (specify specific parts of context to subscribe to, e.g. only location)	(X)	X
F 2.2	Query (single response) for user context	(X)	X
F 2.2.1	Possibility to limit context scope (specify specific parts of	(X)	X

Req. nr	Description	Phase 1	Phase 2
	context to subscribe to, e.g. only location)		
F 2.3	Ontology-based queries		X
F 2.4	WS-based interface for context consumers	X	
F 2.4.1	Mechanism for returning context: Regular WS with callback	X	
F 2.4.2	Mechanism for returning context: WS Base Notification or WS Pubscribe	(X)	X

(X) Optional requirement

Table 39 Context Manager – Non-functional requirements

Req. nr	Description	Phase 1	Phase 2
N 1	Availability		X
N 2	Performance		X
N 3	Scalability		X

5.2. Interfaces

Context Manager has external interfaces as listed in Table 4. This section specifies interfaces for which Context Manager is the server. Interfaces where CM acts as a client are described in other sections.

5.2.1. Context Manager – A4C

Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
A4C Client (WP4.2)	CM	- Obtain static user data (SIP address, RFID, …) - Authorise context consumers (not implemented in Phase 1)	Java API

Protocol details are described in Section 4.2. Basic flows are included in Figure 38.

5.2.2. Context Manager – Context consumer

The Context Manager provides a Web Service interface to context consumers. Results are returned as over a so called call-back interface, i.e. the Context Consumer offers a Web Service interface that context manager may call to return results. Basic flows are shown in Figure 38, methods are shown in Table 40 and Table 41. The WSDL (Web Service Definition Language) file that defines the interfaces is rendered in Appendix B.1.

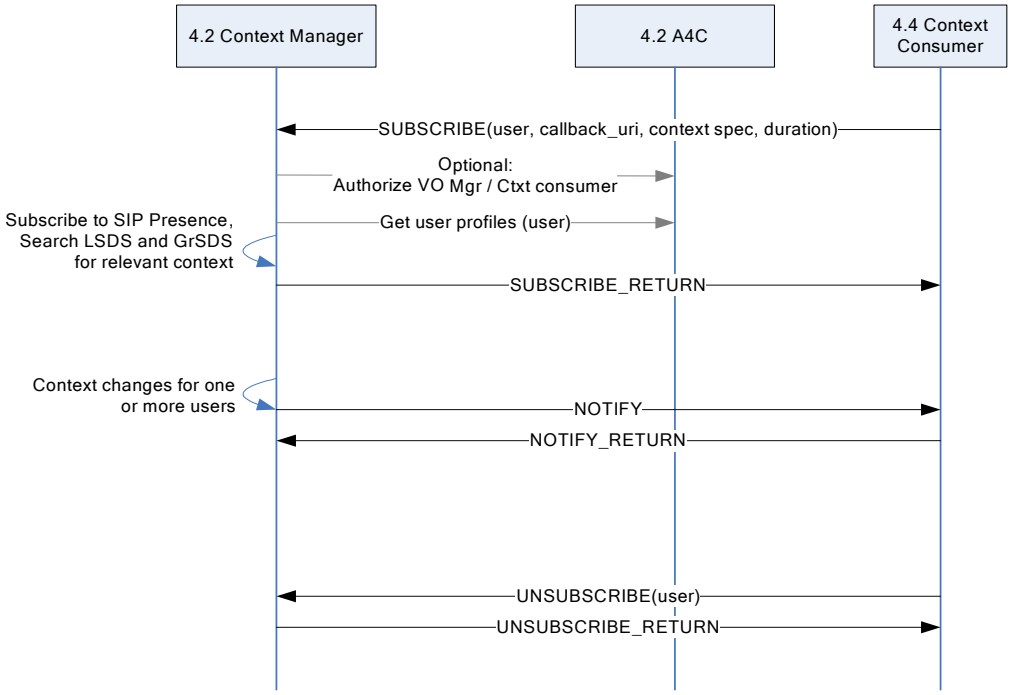


Figure 38 Context Manager – Context consumer interface

Table 40 Methods offered by the Context Manager

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation
CM	Context Consumer (WP4.4)	SOAP	SUBSCRIBE	User	AkogrimoID	Mandatory User to be subscribed to
				Callback_URI	String	Mandatory URI for returning NOTIFY

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation
				context_spec0 context_spec1 context_spec2 context_spec3	Integer Integer Integer Integer	Optional all (default) Presence Location Devices (SLP) Request context if context_specX=1
				Start	Date	dd.mm.yyyy hhmm
				End	Date	dd.mm.yyyy hhmm
				SUBSCRIBE_ REPLY_ACK	Return parameter	Response to successful SUBSCRIBE
				SUBSCRIBE_ REPLY_ ERROR	Return parameter	Response to unsuccessful SUBSCRIBE
CM	Context Consumer (WP4.4)	SOAP	UNSUBSCRIBE	User	AkogrimoID	Removes an existing subscription. Ignored if the subscription does not exist User = -1 removes all subscriptions for this consumer
				UNSUBSCRIBE_ REPLY_ACK	Return parameter	Response to successful UNSUBSCRIBE
				UNSUBSCRIBE_ REPLY_ ERROR	Return parameter	Response to unsuccessful UNSUBSCRIBE

Table 41 Methods offered by the Context Consumer (callback interface)

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation
Context Consumer (WP4.4)	CM	SOAP	NOTIFY	User	AkogrimoID	Mandatory User for which context is provided
				Context	XML schema	Contains requested context
				NOTIFY_ REPLY_ ACK	Return parameter	Response to successful NOTIFY
				NOTIFY_ REPLY_ ERROR	Return parameter	Response to unsuccessful NOTIFY

5.2.3. Context Manager – SIP Presence Agent

Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
SIP PA (WP 4.2)	CM (WP4.2)	- Obtain user presence	SIP SIMPLE

Protocol details are described in Section 3.2.1. Basic flows are shown in Figure 39.

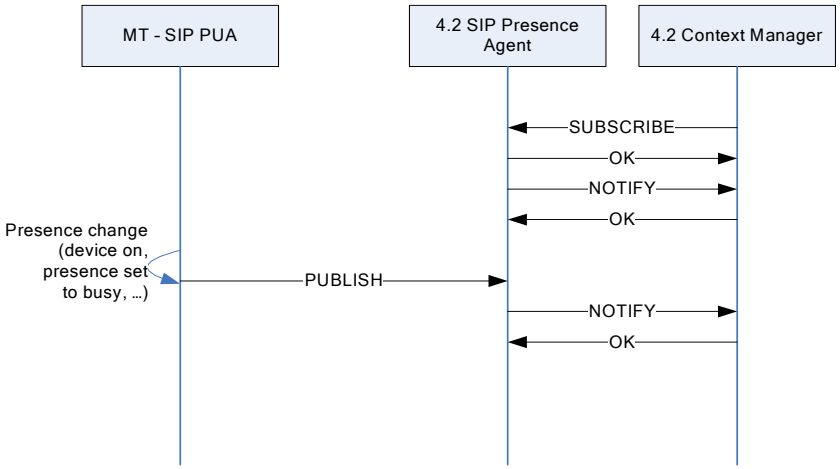


Figure 39 Context Manager – SIP Presence Agent interface

5.2.4. Context Manager – LSDS (SLP Directory Agent)

Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
LSDS (WP4.2)	CM (WP4.2)	- Search of Local Services (capabilities, location)	SLP

Protocol details are described in Section 7.1. Also refer to Section 5.3.1 for design and basic flows.

5.2.5. Context Manager – RFID SW

The method offered by the Context Manager is shown in Table 42. Please refer to Section 5.3.2 for basic flows and application logic.

Table 42 Methods offered by the Context Manager

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation
CM	RFID Reader	Telenor Proprietary	TagReading	TagID	String	Mandatory RFID serial of the RFID tag
				Sensor Location	Location object	Mandatory Location object corresponding to position DB table (see Section 5.3.7.1)

5.3. Design

In this section we describe the various parts of the Context Manager:

- Applications/processes:
  - Context Engine
  - Context consumer Gateway (GW)
  - SIP Presence GW
  - SLP GW

- RFID GW and RFID Software
- Context Manager client at the mobile terminal

- Databases
  - Context DB

Figure 40 and Figure 41 give an overview of these parts as well as their interfaces.

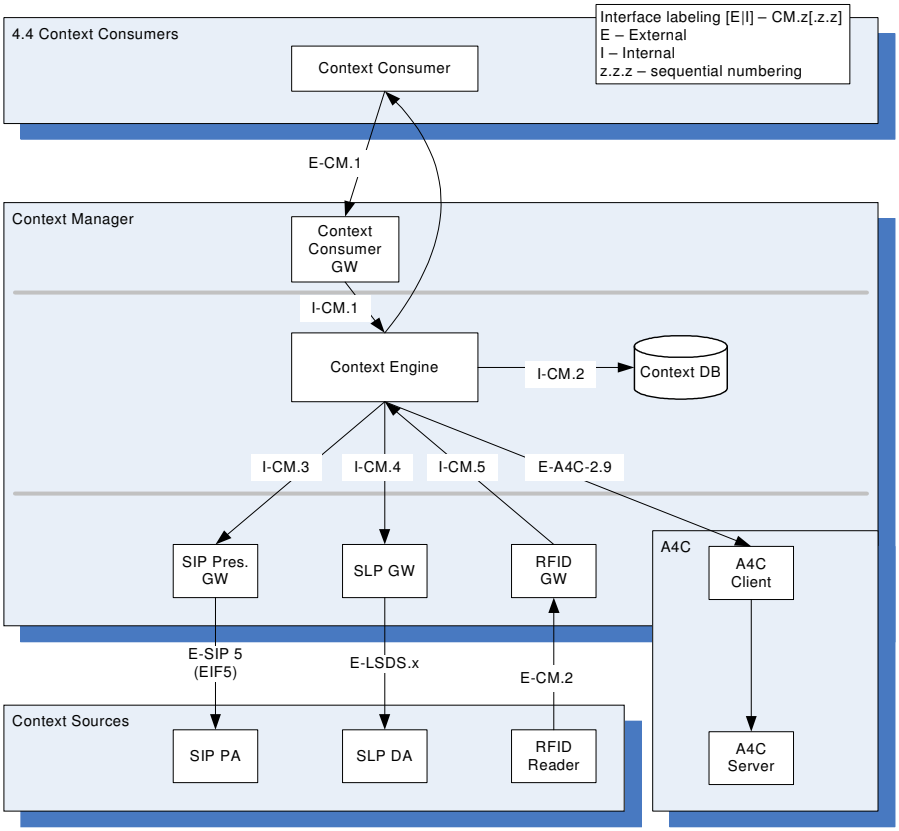


Figure 40 Context Manager overview

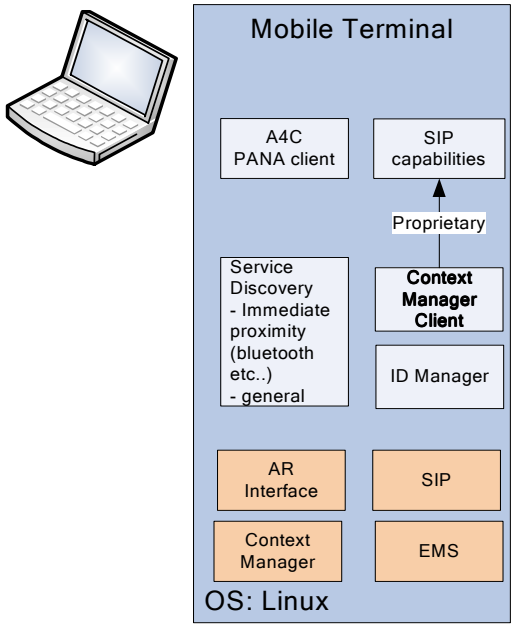


Figure 41 Context manager client at Mobile Terminal

5.3.1. Service Locator Protocol (SLP) Gateway

Local Service Discovery in Akogrimo is based on SLP [Gut99a] which is a service discovery protocol designed for IP networks. SLP specifies three types of agents: User Agent (UA), Service Agent (SA) and Directory Agent (DA). UAs search for services on behalf of an application or user, SAs advertise service information while DAs store information received from SAs and respond to requests from UAs. A service offer consists of a URL and values of descriptive attributes.

In the Akogrimo infrastructure, it is required that one or more DA exists. The Context Manager acts as a UA by sending service requests to this DA(s), which respond with service reply messages containing one or several URLs. SLP offers attribute pattern matching expressions in the form of LDAPv3 search filters, allowing the Context Manager to search for services at a certain physical location. Furthermore, the Context Manager queries the DA for service attributes by sending a service attribute request.

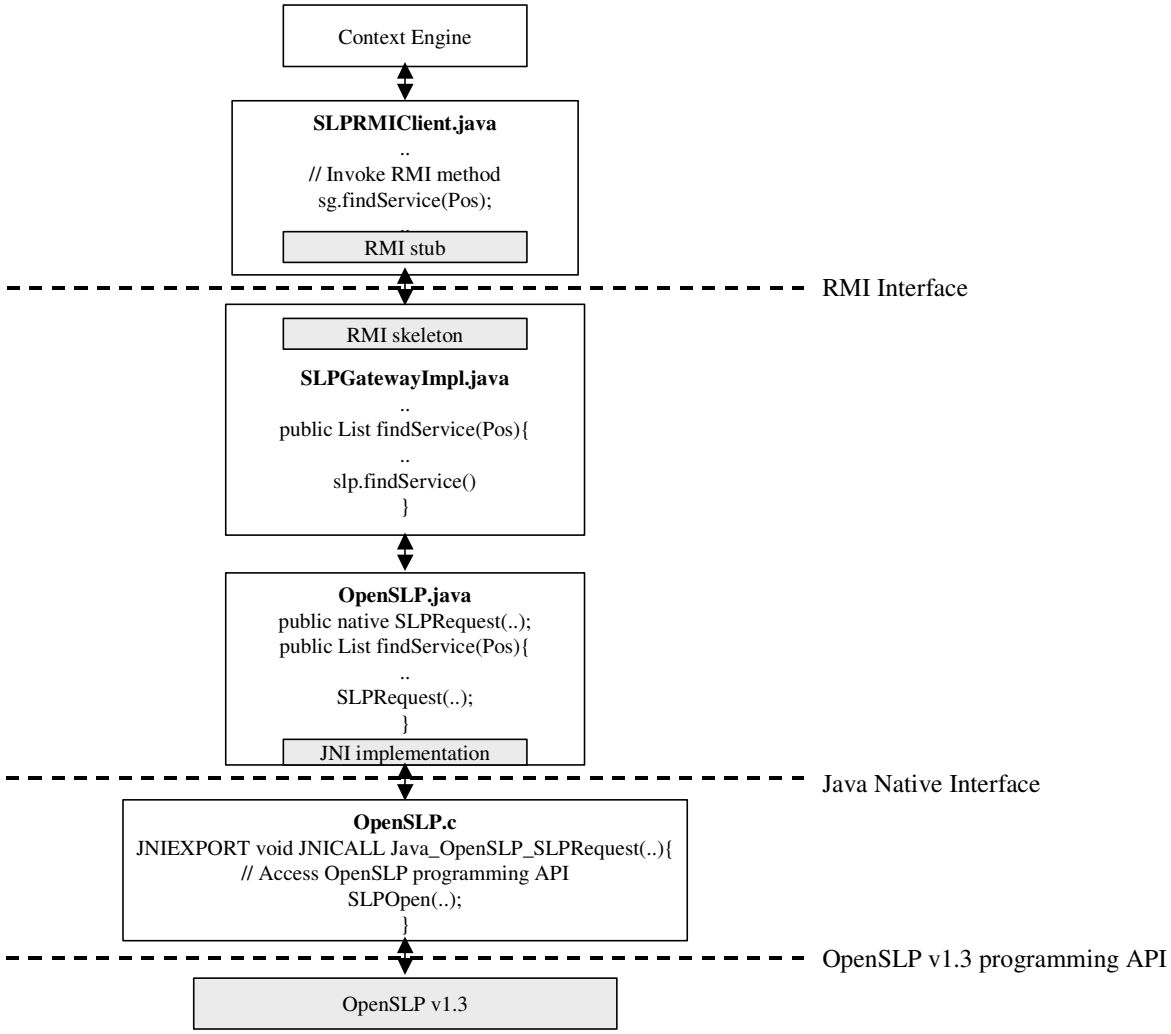


Figure 42 SLP Gateway design

The design of the SLPGateway is illustrated in Figure 42. The Context Engine interacts with the SLPGateway using Remote Method Invocation (RMI). That is, a SLPRMIClient invokes the RMI method offered by the SLPGateway.

The most promising SLP implementation, OpenSLP 1.3 [openslp], is only available in the C programming language. As the Context Manager is implemented in Java, the SLPGateway uses Java Native Interface (JNI) to interact with a native C program. The native C program uses the OpenSLP 1.3 programming API to perform the actual interactions with the SLP DA.

Figure 43 illustrates the interactions required to request service information and corresponding attributes. First, the Context Engine starts a SLPRMIClient (thread). The SLPRMIClient invokes (RMI) findService method offered by the SLPGateway. Using Java Native Interface (JNI) the SLPGateway retrieves service information (urls) from the DA for all services matching the search parameters. Next, the SLPGateway requests the attributes for each service from the SLP DA. Finally, service information, urls and attributes, are returned to the SLPRMIClient.

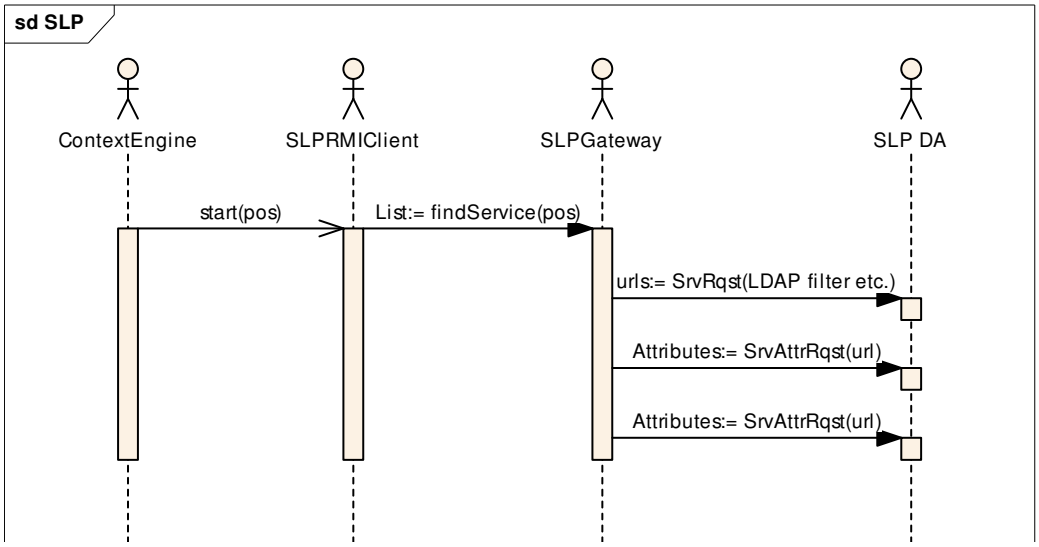


Figure 43 Context Manager – SLP Directory Agent interface

5.3.2. RFID

One way to get location in Akogrimo is by RFID readings. Different RFID sensors can be placed around at specific locations and connected to a RFID Reader. The reader knows the location of each of the different attached sensors. When a RFID tag is read at a sensor the RFID Reader sends a message to the RFID Gateway of the Context Manager.

The RFID Reader is connected to the gateway by a RMI interface. A card reading contains the RFID tag id, which is 16 bytes, sent as a 32 character long string representing the hexadecimal value of the bytes. Information about the location is also sent as string. The location contains longitude, latitude, altitude, horizontal accuracy and vertical accuracy.

An overview of the RFID gateway and RFID Reader is given in Figure 44. The different components interacts using RMI interfaces. The role of the gateway is to act as a proxy for the messages from RFID Readers and to maintain the connections from RFID Readers.

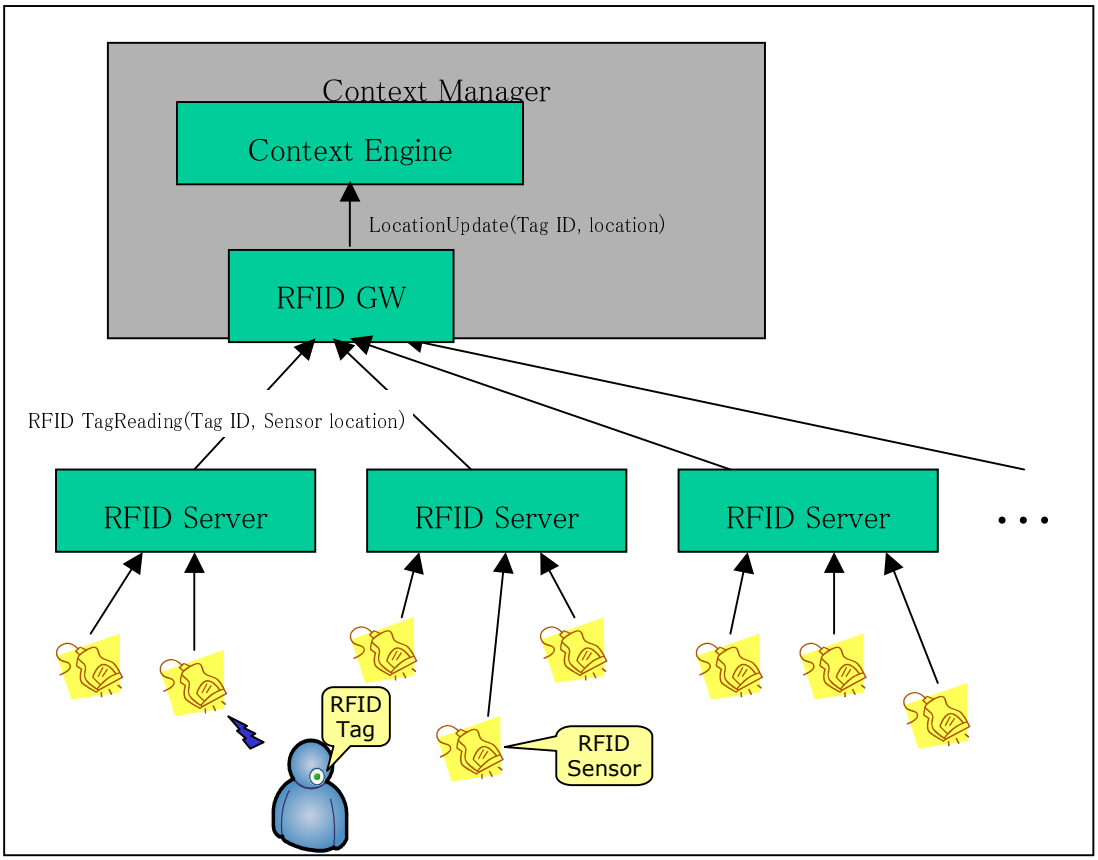


Figure 44 Basic elements for provisioning of RFID position

Figure 45 shows how the different components could interact when a tag is read at a RFID sensor. The Reader is already connected to the gateway using RMI. When the tag is read it transmits the serial of the RFID tag to the sensor. The Reader parses this to a string and adds the location for that reader in a message to the Gateway. The gateway then forwards this message to the Context Engine.

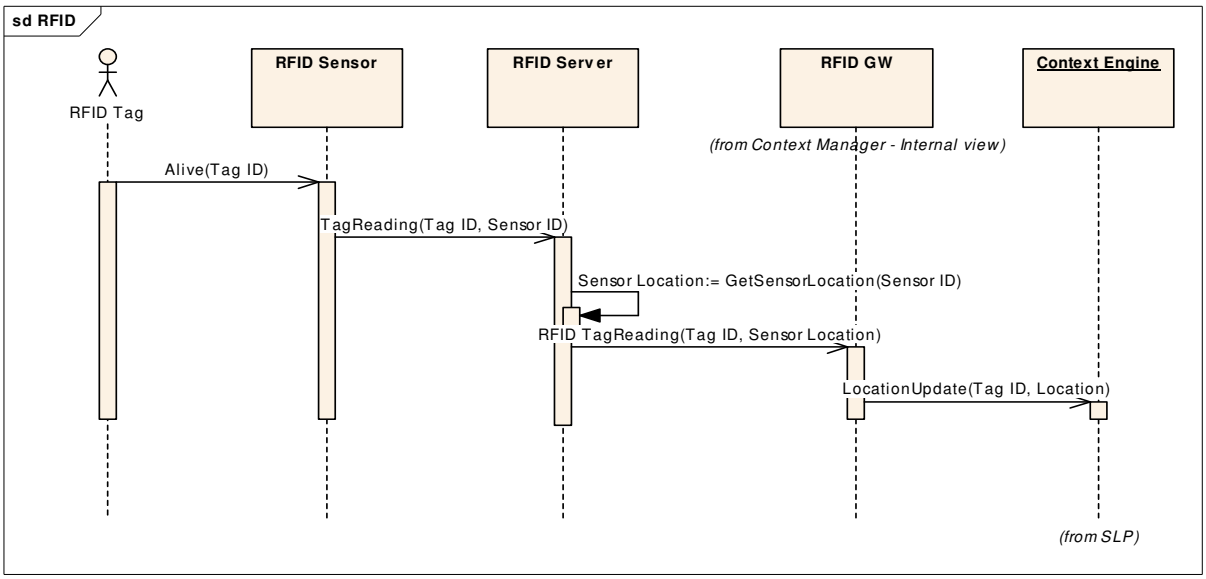


Figure 45 MSC for provisioning of RFID position

5.3.3. SIP Presence GW

The SIP Presence GW is responsible for the communication between Context Manager and SIP Presence Agent. This is realised through the following sub-components:

- **SIP Communicator adapter**, which works as a proxy between Context Engine and SIP Communicator. It receives requests from context engine, and invokes the corresponding java classes in SIP Communicator.
- **SIP Communicator** [SIPComm] acts as a SIP Presence Watcher. It is a Java-based implementation of a SIP User Agent. For further information on the functionality of a SIP Presence Watcher, see the specification of SIP SIMPLE [Ros04].

Basic flows involving the SIP Presence GW is shown in Figure 47 and Figure 48.

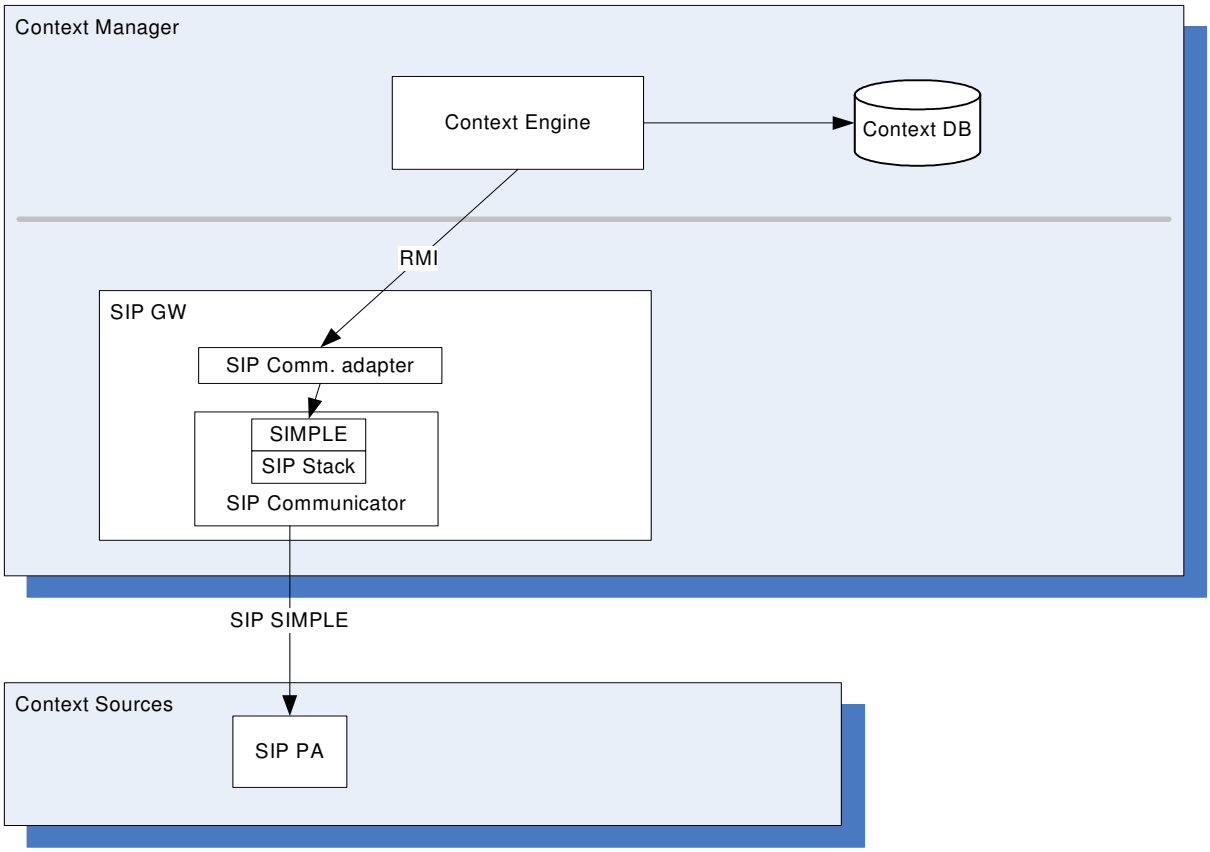


Figure 46 Structure of SIP Presence GW

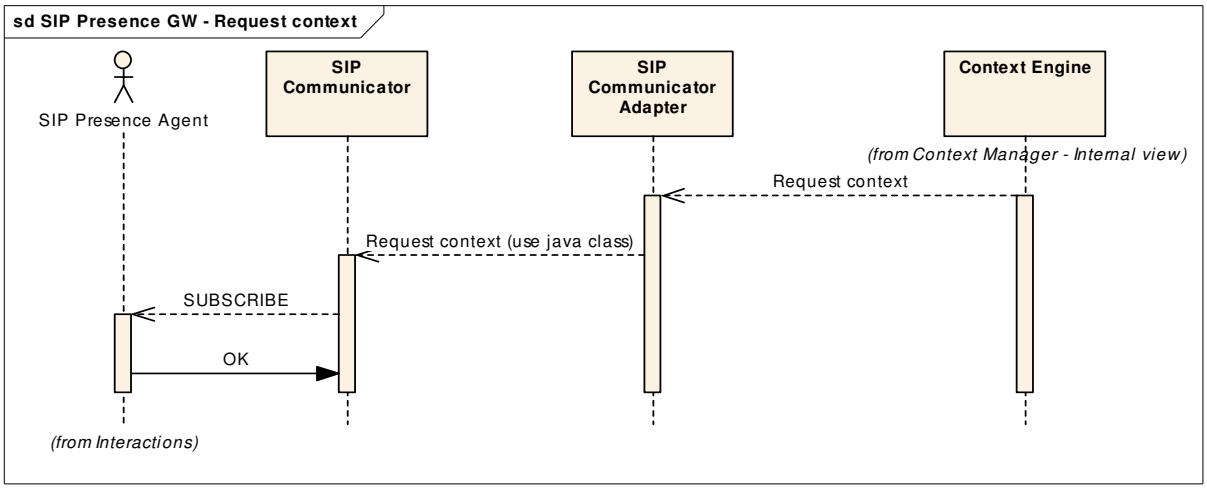


Figure 47 Basic flow – requesting context through SIP Presence GW

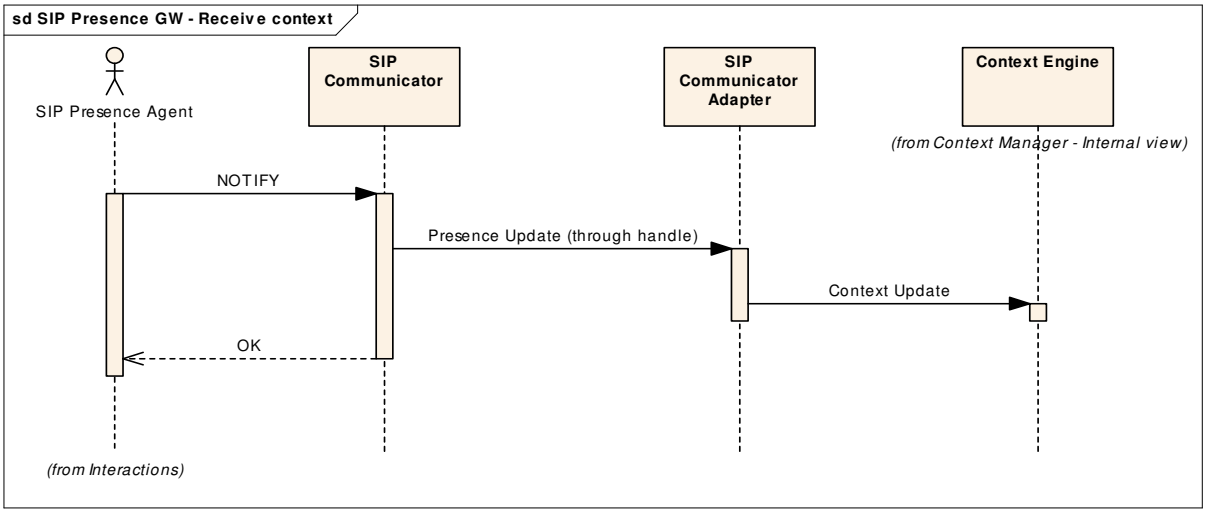


Figure 48 Basic flow – context received from SIP PA

5.3.4. Context Engine

Context engine is the main component in the Context Manager. It's main functional parts are shown in Figure 49. Major tasks are

- Keep overview over subscriptions received through the context consumer gateway
- When needed, request context through context gateways
- Receive context from context gateways, store it in the context database
- When context change occurs, send context updates to appropriate context consumers

The functionality describe here represents a simple approach to context handling; when context for a user is requested, the context engine will request all available context for that user.

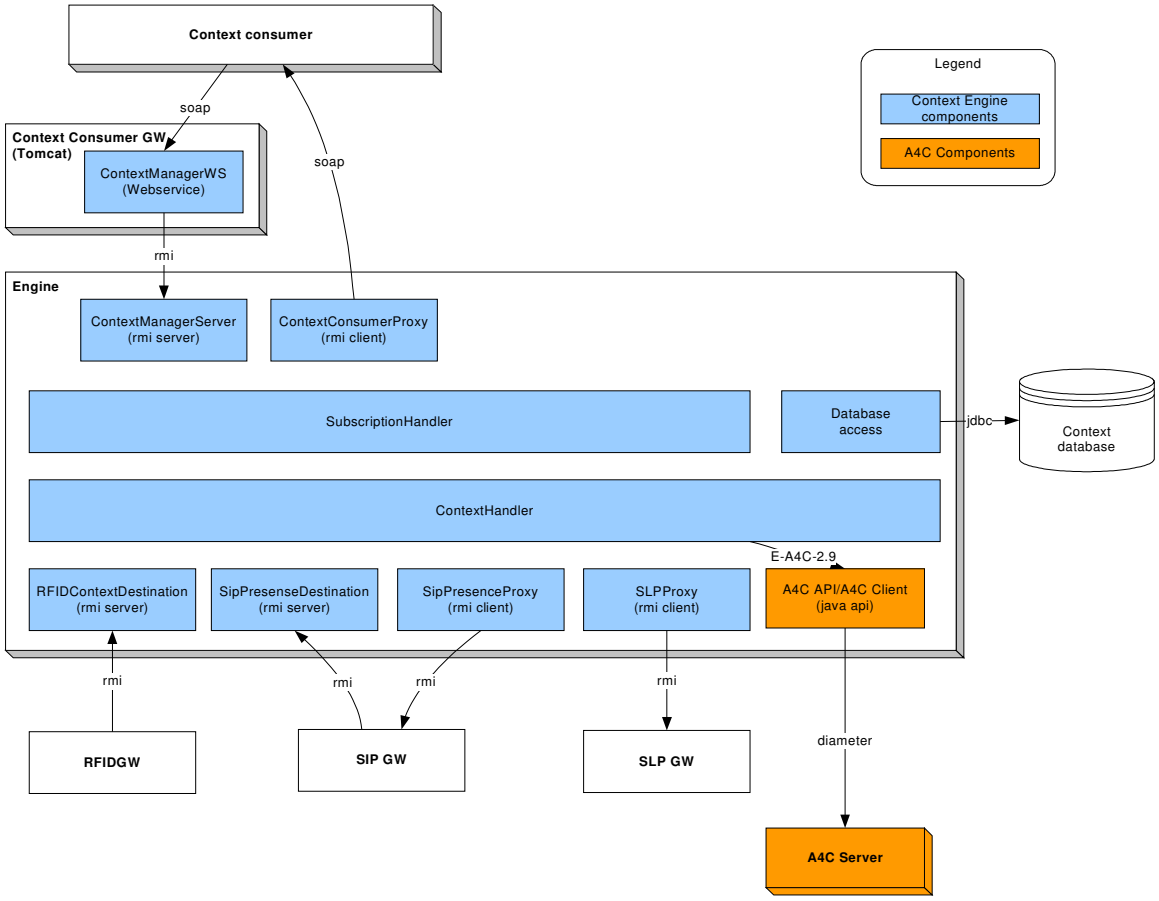


Figure 49 Context Engine – design

Subscriptions from context consumers

When a subscription is received, the following is performed (see Figure 50):

- User profile is requested from A4C
- Request and profile is stored in persistent memory
- Request for SIP presence is sent to the SIP Presence GW

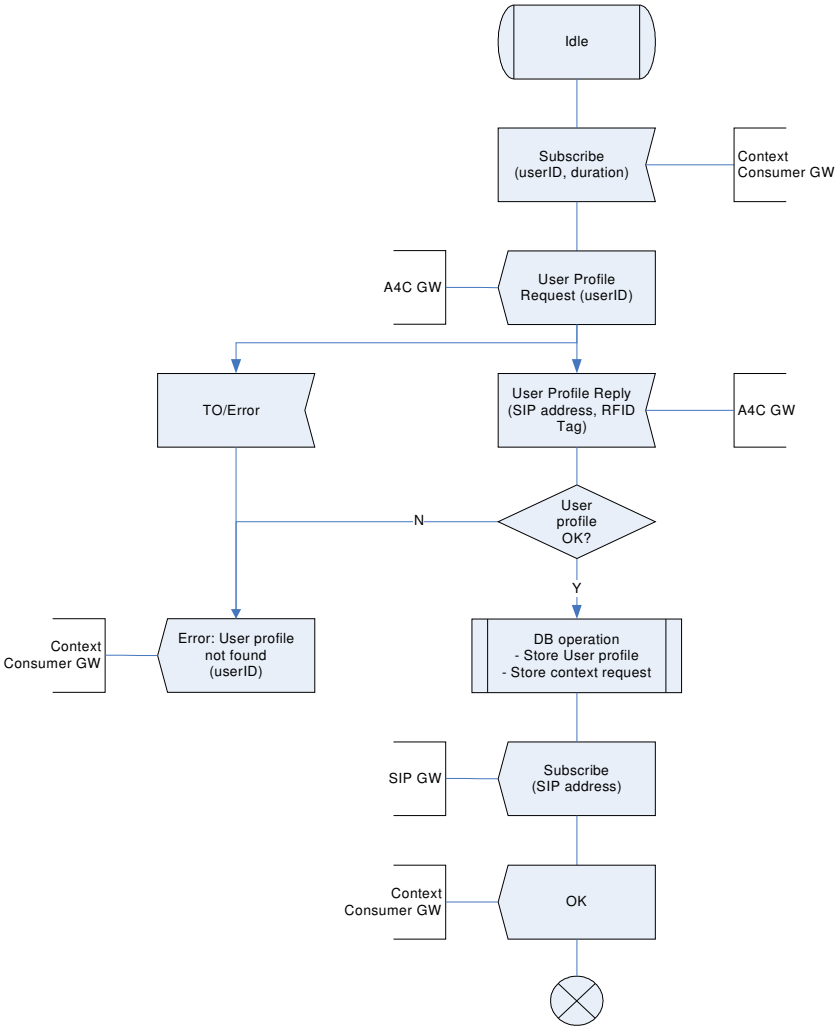


Figure 50 Handling subscriptions – flowchart

Receiving context

Reception of context updates triggers the following behaviour (see Figure 51, Figure 52, and Figure 53):

- When update on SIP Presence is received, the corresponding information (XML scheme) is stored in the context database.
- On updates from RFID GW, Context engine will check if anyone requests context for the associated user. If this is the case, the following occurs
  - Updated position is stored
  - SLP GW is requested for services within range
- After updated context is stored, notification is sent to subscribing context consumers.

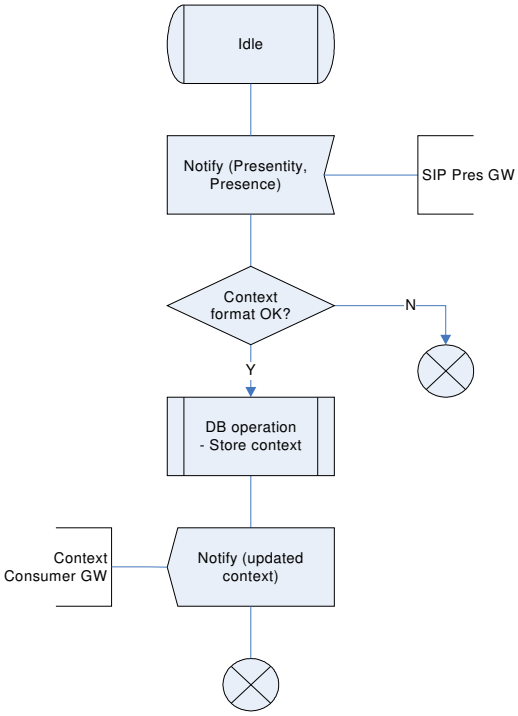


Figure 51 Updated SIP presence

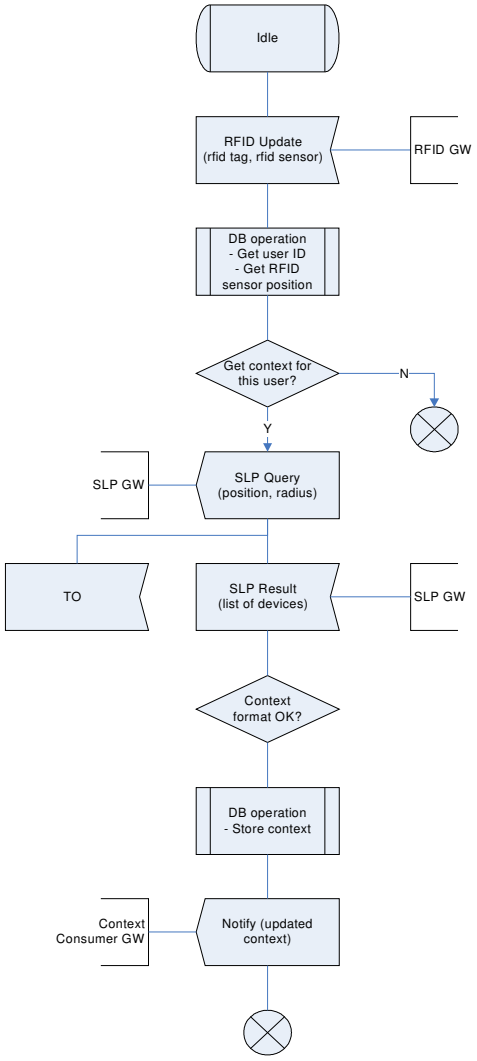


Figure 52 Updated RFID position and availability of SLP services

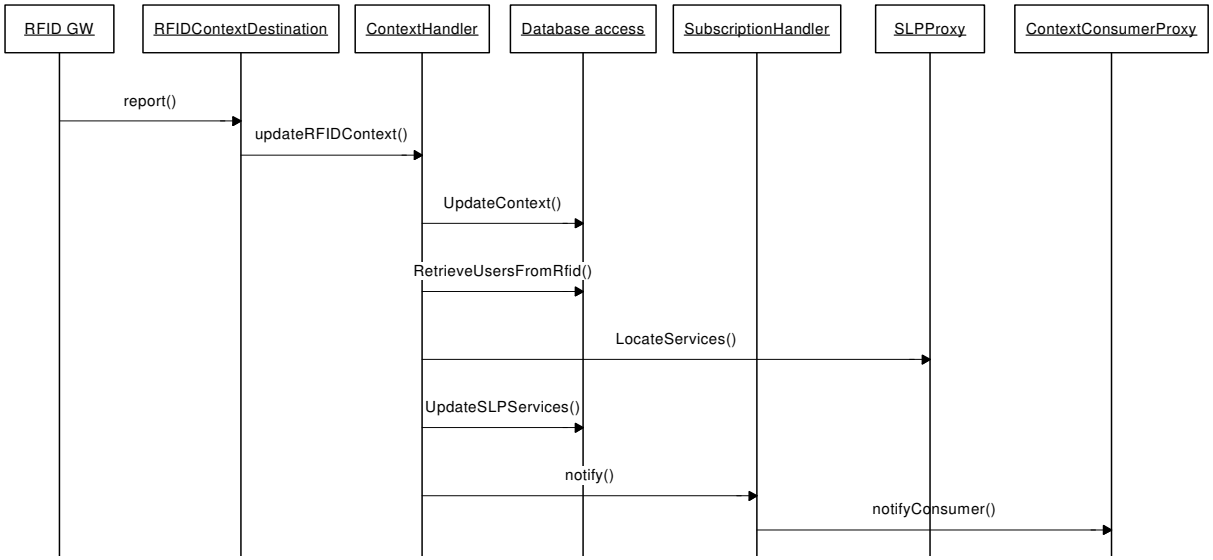


Figure 53 Message Sequence Chart for RFID updates

5.3.5. Context consumer GW

The Context consumer GW handles the communication with Context consumers. Upon receiving a subscription from a Context consumer, it forwards the request to the ContextEngine and returns the status of the subscription. The Context consumer design is illustrated in Figure 49.

- The Context consumer GW is realised as the ContextManagerWS, which is a web service that receives subscriptions.

However, the following components also take part in the communication with a Context Consumer (see Figure 49):

- ContextManagerServer, a RMI server used by ContextManagerWS to forward subscriptions to the ContextEngine.
- ContextConsumerProxy, a RMI client that sends notifications to Context consumers when context changes.

5.3.6. CM Client on MT

The scope of the CM Client on the MT is, in implementation Phase 1, to get the User Agent Profile [UAProf] for the terminal. A method for publishing the UAProf of the terminal will be provided by the SIP PUA implementation. So the part on the mobile terminal is only for making the link to the UAProf available as SIP Presence information.

The information about the UAProf must be utilised by the Context Manager on the network side. When the link is read from the XML document received by the Context Manager a lookup is made to see if the terminal characteristics are in the data storage. If not the UAProf XML document is downloaded and parsed for relevant information.

The following two objects were designed to help with the handling of UAProf information (at the moment in package org.akogrimo.WP42.ContextManager.UAProf):

UAProfInformation

This object holds the following information from a User Agent Profile.

- Vendor
- Model
- UAPrfoURL
- ColorCapable
- ScreenSize
- ImageCapable
- Keyboard
- ScreenSizeChar
- SoundOutputCapable
- TextInputCapable
- VoiceInputCapable

And allows for setting and getting of each value, in addition to getting a string and a XML string version of the information.

UAProfXMLReader

This object utilises the SAX parser and collects the information from a UAProf document. A URL for the document is sent as input and a method allows for collection of a UAProfInformation object containing the relevant information.

5.3.7. Context DB

In Akogrimo, user context will (wrt implementation in Phase 1) consist of

- Presence realized with SIP SIMPLE, described in terms of RPID
- User location realized with RFID technology, described in terms of on UTM (Universal Transverse Mercator)
- Device availability realized with SLP and UAProf, Device properties are described in terms of key-value pairs.

Format: A RPID scheme (see example in Section 5.3.7.2) will be created to include user location and properties for available devices.

5.3.7.1. Database model

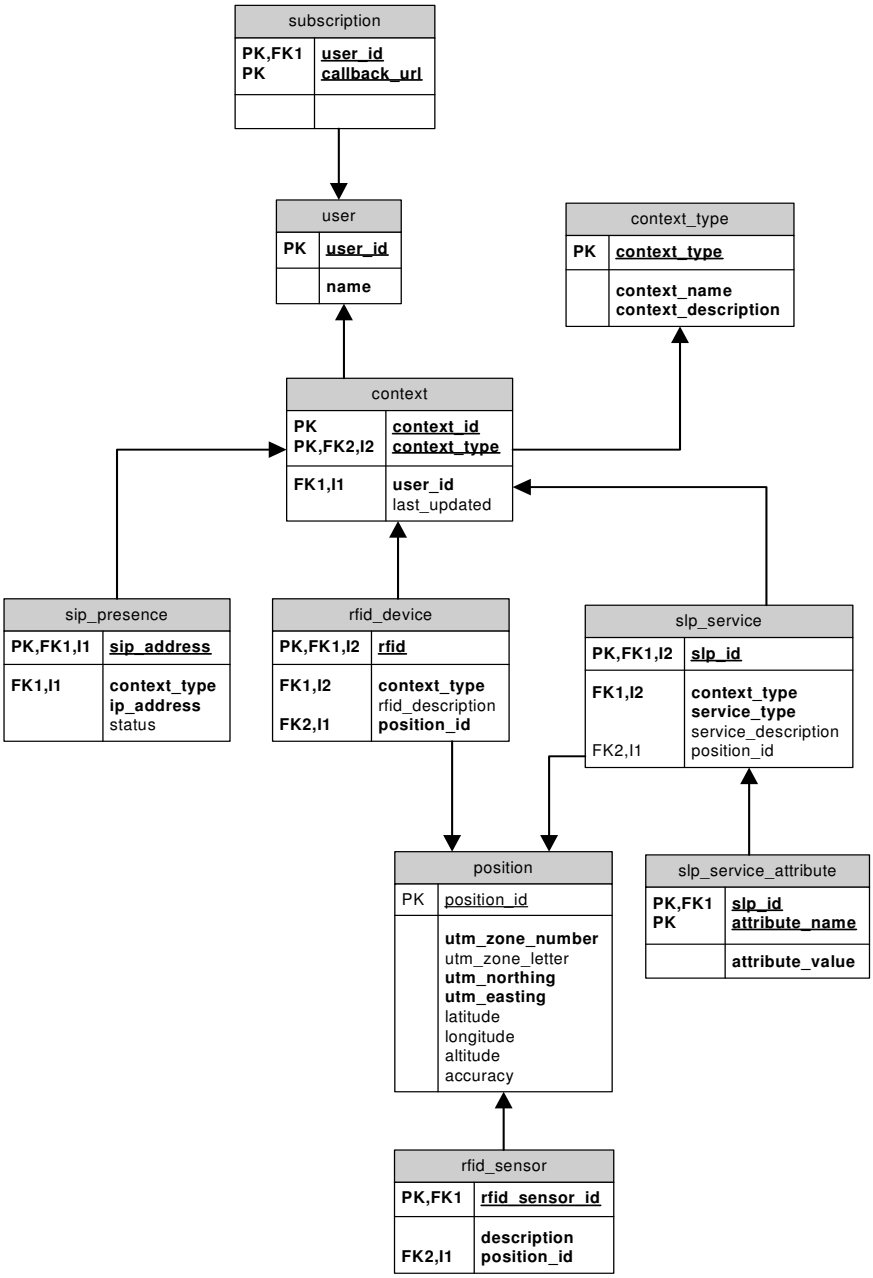


Figure 54 Context DB

Briefly, a *user* is related to his/her context through the *context* table. RFID tags and SLP services are given a position in the *position* table. The tables and the most important attributes are described in Table 43. Description of the context and its values is described in the subsequent sections.

Table 43 Tables and selected attributes

Table	Attribute	Description
subscription	Callback_url	Reference to context consumer
user	user_id	Akogrimo ID
context	context_id	Unique identifier, identical to sip_address, rfid, or slp_id

Table	Attribute	Description
	context_type	from table context_type
context_type	context_type	1 = SIP Presence
		2 = RFID
		3 = SLP
sip_presence	sip_address	SIP-address for the user
	status	XML-form received from SIP PA
rfid_device	rfid	RFID Tag ID for the user
	position	Refers to an item in the *position* table
slp_service	slp_id	Identical to serviceURL, see Table 45 below
	service_type	E.g. printer, display, ssh, …
	position	Refers to an item in the *position* table
slp_service_attribute		Collection of attributes for a SLP service
position	position_id	One ID per registered position
	<attribute>	See description in Section 5.3.7.3.
rfid_sensor		Describes location of RFID sensors

5.3.7.2. Presence

The below example of SIP presence taken from RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF) [Schu05].

```
The example below describes the presentity 'pres:someone@example.com', which has a SIP
contact, 'sip:someone@example.com', representing a service.  It also has a device contact, as
an email box.  The presentity is in a meeting, in a public office setting.  The 'until'
information indicates that he will be there until 5.30 pm GMT.  The presentity also has an
assistant, sip:secretary@example.com, who happens to be available for communications.
   <?xml version="1.0" encoding="UTF-8"?>
   <presence xmlns="urn:ietf:params:xml:ns:pidf"
   xmlns:p="urn:ietf:params:xml:ns:pidf:person"
   xmlns:d="urn:ietf:params:xml:ns:pidf:device"
   xmlns:rs="urn:ietf:params:xml:ns:pidf:status:rpid-status"
   xmlns:rt="urn:ietf:params:xml:ns:pidf:rpid-tuple"
   xmlns:rp="urn:ietf:params:xml:ns:pidf:rpid-person"
   xmlns:rd="urn:ietf:params:xml:ns:pidf:rpid-device"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   entity="pres:someone@example.com"
   xsi:schemaLocation="urn:ietf:params:xml:ns:pidf pidf.xsd
   urn:ietf:params:xml:ns:pidf:status:rpid-device rpid-device.xsd
   urn:ietf:params:xml:ns:pidf:status:rpid-tuple rpid-tuple.xsd
   urn:ietf:params:xml:ns:pidf:status:rpid-person rpid-person.xsd
   urn:ietf:params:xml:ns:pidf:status:rpid-status rpid-status.xsd
   urn:ietf:params:xml:ns:pidf:rpid-tuple rpid-tuple.xsd">
     <tuple id="t0">
       <status>
         <basic>open</basic>
       </status>
       <et:class>assistant</et:class>
       <et:relationship>assistant</et:relationship>
       <contact>sip:secretary@example.com</contact>
       <note>My secretary</note>
     </tuple>
     <tuple id="t1">
       <status>
         <basic>open</basic>
         <rs:privacy>
           <audio/>
         </rs:privacy>
         <rs:idle since="2003-01-27T10:43:00Z"/>
       </status>
       <et:class>sip</et:class>
```

```
    <contact priority="0.8">sip:someone@example.com</contact>
    <timestamp>2001-10-27T16:49:29Z</timestamp>
  </tuple>
  <tuple id="t2">
    <status>
      <basic>open</basic>
      <e:privacy>
        <text/>
      </e:privacy>
      <e:timeoffset>300</e:timeoffset>
    </status>
    <contact priority="0.8">im:someone@mobilecarrier.net</contact>
    <timestamp>2001-10-27T16:49:29Z</timestamp>
  </tuple>
  <tuple id="t3">
    <status>
      <basic>open</basic>
    </status>
    <et:class>mail</et:class>
    <contact priority="1.0">mailto:someone@example.com</contact>
    <note>I'm in a boring meeting</note>
  </tuple>
  <tuple id="t4">
    <status>
      <basic>closed</basic>
    </status>
    <et:service-class>in-person</et:class>
    <note>Closed-door meeting</note>
  </tuple>
  <p:person>
    <p:status>
    <rp:activities>
      <rp:meeting/>
    </rp:activities>
    <rp:class>composed</rp:class>
    <rp:mood>
      <rp:happy/>
      <rp:text>I got my paycheck!</rp:text>
    </rp:mood>
    <rp:place-type until="2003-01-27T17:30:00Z">
      <rp:office/>
    </rp:place-type>
  </person>
</presence>
```

5.3.7.3. Location

Location description is based on UTM (Universal Transverse Mercator) [MLP], which basically is a point in a 2D coordinate system called a zone. The world has been divided into several zones on which one may assume properties of a 2D coordinate system. Read more here: http://www.dmap.co.uk/utmworld.htm. For converting from longitude/latitude to UTM, see this page: http://www.ngs.noaa.gov/cgi-bin/utm_getut.prl

Table 44 Location parameters

Parameter	Explanation
utm_zone_number	A UTM zone is indicated by letter and number. This is the number indicator
utm_zone_letter	UTM zone, number indicator
utm_northing	Unit: Meters. For the northings in the northern hemisphere, the origin is defined as the equator. For the northings in the southern hemisphere, the origin is defined as a point 10,000,000 metres south of the equator
utm_easting	Unit: Meters. For the eastings, the origin is defined as a point

Parameter	Explanation
	500,000 metres west of the central meridian of each longitudinal zone, giving an easting of 500,000 metres at the central meridian
Altitude	Above sea level. Unit: Meters
Accuracy	Radius indicating how accurate the measuring is. RFID is accurate, and has low radius (usually set to 5). GSM position would typically have accuracy = 200. Unit: Meters

Example 1: Trondheim, Norway

Parameter	Value
utm_zone_number	32
utm_zone_letter	V
utm_northing	7035377
utm_easting	569838
altitude	0
accuracy	N/A

Example 2: London, UK

Parameter	Value
utm_zone_number	30
utm_zone_letter	U
utm_northing	5709345
utm_easting	699459
Altitude	20
accuracy	N/A

Example 3: Stuttgart, Germany

Parameter	Value
utm_zone_number	32
utm_zone_letter	U
utm_northing	5404492
utm_easting	513500
altitude	300
accuracy	N/A

5.3.7.4. Device availability and description

Devices/localizable services become available to users in two ways:

- Directly via logging on: By logging on to a SIP UA installed on a device, we assume that the user has direct access to and control over that device.
- Indirectly via location: Assuming that stationary devices/services may be registered with a particular position, one may infer device availability for a user based on the user's position.

Description of device attributes is realised in terms of key-value pairs. See examples in sections below.

User logged on UA on device

For description of attributes, see Section 5.3.6.

Devices/services available in user surrounding

Devices and services are managed using Service Location Protocol (SLP), where attributes are registered by a system administrator. Location will be included in the set of properties. User location, which is obtained by means of RFID positioning technology, may in turn be used when searching SLP for devices in the surrounding (see Figure 55).

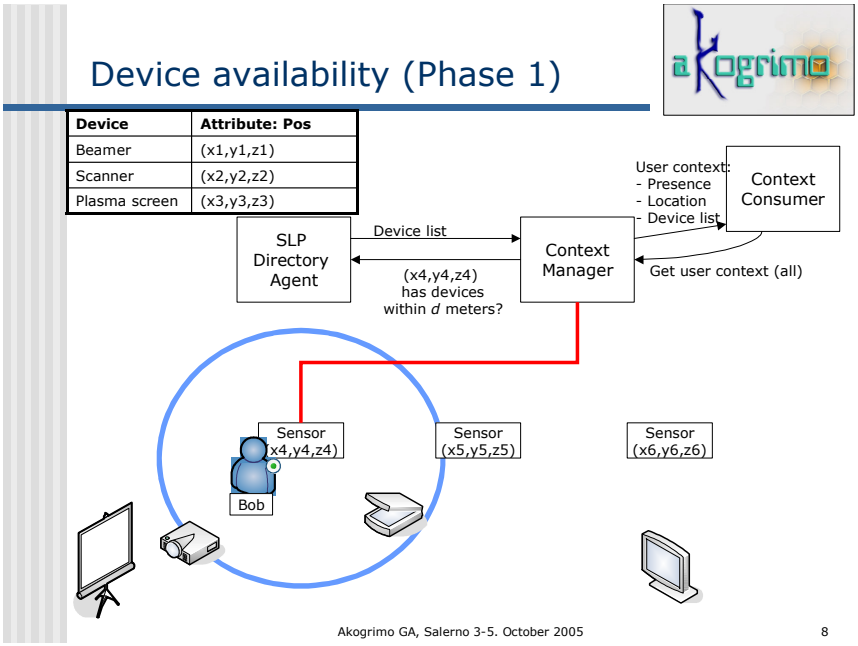


Figure 55 Relating user and device/service location

Table 45 Property description

Field	Description
serviceURL	Where to localize the service. A Service URL may be of the form "service:"<srvtype>"://"<addrspec>"
Service Type	Each type of service has a unique Service Type string.
Lang	Services MAY be registered in multiple languages.
Scope	A set of services, typically making up a logical administrative group.

Attributes	<List of key-value pairs> (attr1=vaule1), (attr2=value2), …

Table 46 Property example

Field	Sample values
URL	service:printer:lpr://igore.wco.ftp.com/draft
scope-list	Development
Lang. Tag	en
Attributes	(Name=Igore),(Description=For developers only), (Protocol=LPR),(location-description=12th floor), (Operator=James Dornan ¥3cdornan@monster¥3e), (media-size=na-letter),(resolution=res-600),x-OK

Table 47 Attributes relevant for Akogrimo

Attribute	Comment
Resolution	Exact description (1024x768, 1280x1024) or indicative description (high, medium, low)
Bandwidth	Max bandwidth, i.e. not a network status indicator. Exact description (128kb, 384kb, 1Mb, 10Mb) or indicative description (high, medium, low)
utmNorthing, utmEasting, altitude, utmZoneLetter, utmZoneNumber	Physical location
SIP address	For using a device in a SIP sessions. Value may be plasma1@trondheim.telenor.com

5.4. Implementation

This section gives a brief description of implementation matters. At the time of writing the software described has not reached a mature state, hence it is not appropriate to give a detailed description here.

5.4.1. Service Locator Protocol (SLP) Gateway

The SLP GW implementation consists of the following parts:

- **SLPGateway,** a RMI server implemented in java that handles SLP requests made by the ContextEngine.
- **OpenSLP (java)**, a java implementation of the communication with the SLP DA. To be able to access the C programming API offered by openslp 1.3, this class has a native function (private native void SLPRequest). Callback functions are an integral part of the SLP. Such callback functions will be called by the library when it is ready to report the

status or results of an operation API. OpenSLP (java) implements functions that are called by the corresponding callback function in C.

- **OpenSLP (C program)**, an implementation in C of the corresponding native function (Java_org_akogrimo_wp42_contextmanager_slpgateway_jni_OpenSLP_SLPRequest). The C program receives different SLP requests and invokes the proper method in the openslp API. Finally, when openslp API invokes the callback function specified in C, the progam calls the corresponding callback method in java to return results.

5.4.2. RFID

An outline of the RFID system is shown in Figure 44. The implementation is based on

- Hardware: RFID tags and RFID sensors from third party equipment providers
- Software:
  - Drivers for communication with RFID sensors. Drivers exist both for Windows and Linux platforms
  - RFID Server software: Java-based, Telenor proprietary software for handling signalling conversion between RFID sensors and RFID GW.
  - RFID GW: Java-based software developed for the Akogrimo project. Handles the communication between RFID Server and Context Engine

5.4.3. SIP Presence GW

The SIP Presence GW is realised in Java. It mainly consists of two parts:

- **SIP Communicator adapter**, which implements interfaces to SIP Communicator. It also implements an RMI interface to communicate with Context Engine.
- **SIP Communicator**, a Java-based implementation of a SIP User Agent. SIP Communicator is developed on top of JAIN-SIP-RI and JMF (Java Media Framework).

5.4.4. Context Engine

The context engine is implemented in Java. The main components are shown in Figure 49. A brief description follows:

- ContextManagerServer is a RMI server that handles requests from ContextManagerWS
- ContextConsumerProxy is a RMI client that sends responses to the context consumer
- SubscriptionHandler will store context requests, and is also responsible for handling context notification sent back to the consumer
- ContextHandler is the main component. It is responsible for

    - interactions with A4C
    - requesting context when necessary
    - handling context collected from the context sources,
    - storing context in the database (using the component Database Access)
    - notifying SubscriptionHandler when context changes occur
- RFIDContextDestination handles communications with RFID-related systems
- SipPresenceDestination sends requests to the SIP Presence GW via RMI
- SipPresenceProxy receives responses from the SIP Presence GW via RMI
- SLPProxy handles communications with SLP GW
- Database access provides an interface to available database operations

5.4.5. Context consumer GW

The Context consumer GW is implemented in java. The main components are the following:

- ContextManagerWS, a web service implemented in java on Tomcat/Axis that receives subscribtions. SOAP services are deployed into a Tomcat server while Axis is essentially a SOAP engine.
- ContextManagerServer, a RMI server that offers an interface to the ContextManagerWS to interact with the ContextEngine.
- ContextConsumerProxy, a RMI client that sends notifications to Context consumers when context changes.

5.4.6. CM Client on MT

5.4.6.1. Package UAProfHandler - Data definition

Enumeration CapableInfo (inner class of UAProfInformation)

Used to show if a capability is present on the terminal, Boolean can not be used because the possibility that the information is not available in the UAProf XML document.

Enumeration Value	Description
Yes	The capability is present
No	The capability is not present
NotDescribed	There is no description of the capability in the UAProf

Class UAProfInformation

Used to store information parsed from a UAProf XML document. The structure can be enhanced in the future if we need more information.

Element Name	Element Type	Description
Vendor	String	The vendor for the terminal
Model	String	The Model of the terminal
UAProfURL	String	Link to the UAProf for the terminal
ColorCapable	CapableInfo	In the terminal has a colour capable screen
ScreenSize	String	The size of the screen in pixels (i.e. 240x320)
ImageCapable	CapableInfo	If the terminal can display images
Keyboard	String	The type of keyboard on the terminal
ScreenSizeChar	String	The size of the screen in characters (i.e. 20x15)
SoundOutputCapable	CapableInfo	If the terminal can play sound
TextInputCapable	CapableInfo	If the terminal can receive text input
VoiceInputCapable	CapableInfo	If the terminal can receive voice input

Method descriptions:

The main usage from **UAProfInformation** will be the two methods:

String toString() returning a string representation

String toXMLString() returning a XML formatted string representation

class UAProfXMLReader extends DefaultHandler

This class extends the DefaultHandler from SAX and utilises methods from it to pars a XML document.

The constructor receives the URL for the XML document to be parsed.

Method descriptions:

The main method for use is the following:

UAProfInformation GetUAProfInformation (UAProfDocument)

The Get UAProfInformation parses the UAProf XML file and collects information that can be used by the Context Manager. This information is stored in a UAProf structure.

For the first implementation it is considered that the input document is well formed without errors. The parsing will pick one and one element to gather and populate the UAProfInformation Structure. If a element is not present the value will be "N/A" String and NotDescribed CapableInfo.

5.5. Testing

This section defines a limited set of tests for verification of basic functionality. The tests relate to requirements to be implemented in phase 1 (i.e. by PM 19).

Table 48 Test cases

Test nr.	Req. nr	Description	Phase 1	Phase 2
	F 1	**Gather context (end user information)**	**X**	**X**
	F 1.1	Presence	X	
	F 1.2	Position	X	
	F 1.2.1	Position – RFID based	X	
T F1.2.1		Presumptions: • Proper RFID Tag data resides within A4C • There are two RFID Sensors with different positions • Proper RFID Sensor data resides within CM Test and verification steps: 1. Swipe RFID Tag over RFID Sensor 1 2. Verify that User position is updated in the context database 3. Verify that corresponding context update is received by the context consumer 4. Swipe RFID Tag over RFID Sensor 2. Repeat steps 2. and 3.		
	F 1.2.2	Position – based on e.g. GPS or WLAN		(X)
	F 1.3	Terminal capabilities for users terminal	X	
T F1.3		Presumptions: • Mobile terminal is switched off Test and verification steps: 1. MT is switched on, SIP UA starts 2. Verify that correct information is received by CM in SIP Presence 3. Verify that information provides sufficient basis to describe MT attributes Additional details may be found in Appendix		
	F 1.4	Discover services in user's proximity		
	F 1.4.1	Terminal-based discovery (discovery performed by user's terminal)		
	F 1.4.2	Centralised, directory-based discovery (match users position with services for that position)		
T F1.4.2		Presumptions: • Three services are registered with SLP • User passes RFID Sensor within range of two of the services Test and verification steps: • Verify that RFID update propagates to CM and triggers SLP request		

Test nr.	Req. nr	Description	Phase 1	Phase 2
		• Verify that CM receives attributes for the two services • Verify that corresponding context update is received by the context consumer		
T F1.4.2				
	F 2	**Distribute context (to context consumers)**	**X**	
	F 2.1	Offer subscription for user context	X	
	F 2.1.1	Possibility to limit context scope (specify specific parts of context to subscribe to, e.g. only location)	(X)	X
	F 2.2	Query (single response) for user context	(X)	X
	F 2.2.1	Possibility to limit context scope (specify specific parts of context to subscribe to, e.g. only location)	(X)	X
	F 2.3	Ontology-based queries		X
	F 2.4	WS-based interface for context consumers	X	
	F 2.4.1	Mechanism for returning context: Regular WS with callback	X	
	F 2.4.2	Mechanism for returning context: WS Base Notification or WS Pubscribe	(X)	X

6. Grid Service Discovery

6.1. Interfaces

Table 49 Grid Service Discovery Server (GrSDS) – Interfaces

Interface	Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
E-GrSDS-1	GrSDS	BPE (WP4.4)	Search and Fetch a Grid Service.	SOAP
E-GrSDS-2	GrSDS	BVO Manager (WP4.4)	Publish/Deregister Service	WSDL/SOAP
	SLA Manager (WP4.4)	GrSDS	GrSDS interacts with the SLA-Access to retrieve the SLA from published SLA-Template and associate them to the interface of service published in GrSDS	SOAP
E-GrSDS-4	GrSDS	OpVOBroker (WP4.4)	Look for services to carry out a certain WF	SOAP

6.2. Requirements

Grid Service Discovery – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 6	**Gather Service information**	**X**	**X**
F 6.1	Grid Service Description		X
F 6.1.1	Service provider description		X
F 6.1.2	SLA description		X
F6.1.3	Other parameters description		X
F 7	**Service publishing**		X
F7.1	Grid Service Publishing by Service Provider		X
F7.1.1	Publishing without SLA information		X

Req. nr	Description	Phase 1	Phase 2
F7.1.2	Publishing with SLA information		X
F7.2	Grid Service Publishing by BVO Manager		X
F7.2.1	Publishing without SLA information		X
F7.2.2	Publishing with SLA information		X
F8	**Query Service Information**	**X**	**X**
F 8.1	Look up services by pattern matching		X
F 8.2	Look up services by service type		X
F 8.3	Look up services within attribute ranges		X
F 8.4	Look up services by "semantic matching" and ontology based queries		X
F8.5	Look up services by specific SLA parameters		X
F9	**Security**		X
F9.1	**Permits and authentication**		X
F9.1.1	Just allow authenticated Services to publish information		X
F9.1.2	Just allow users to query for services inside their VO or open VOs		X
F9.1.3	Complex permission system (just allowed to query at a give time, or when something else occurred, etc…)		(X)
F9.2	Secure communications (e.g. IPsec)		(X)
F10	**Deregister Services**		X
F10.1	By Service Provider		X
F10.2	By BVO Manager		X
F11	**Updating Services**		X
F 11.1	By Service Provider		X
F11.2	By BVO Manager		X

Req. nr	Description	Phase 1	Phase 2
F12	Robustness in front of Service Provider Mobility		X

(X) Optional requirement

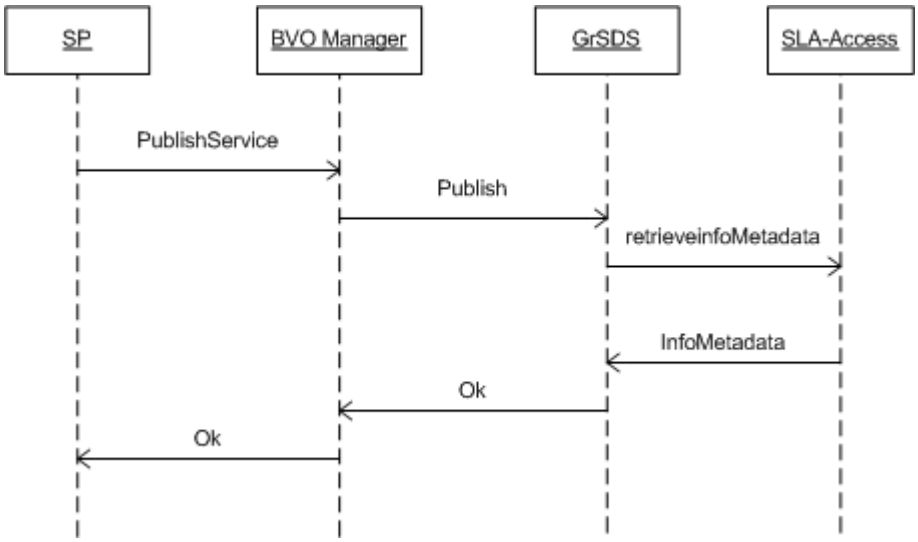
6.3. Design

The GrSDS represents a "static" discovery registry with semantics capabilities where the static description of services (i.e. WSDL description) is stored. Then GrSDS stores the description of Grid Services, i.e., WS-Resources. From now on those terms will be considered equivalent. The two basic process involving GrSDS are the publication and the search of Grid Services (WS-Resources). In both processes the GrSDS interacts with are:

- Publication of Grid Services (WS-Resources)
- Search of Grid Services (WS-Resources)

The publication of Grid services consists mainly on the store of the description of services in the same way as WSDL document are stored in a standard UDDI registry. Furthermore, the publication of Grid Services implies also the process of storing information to the corresponding SLA Template stored physically in the SLA Repository.

The components involved on the publication process can be seen in the following sequence chart:

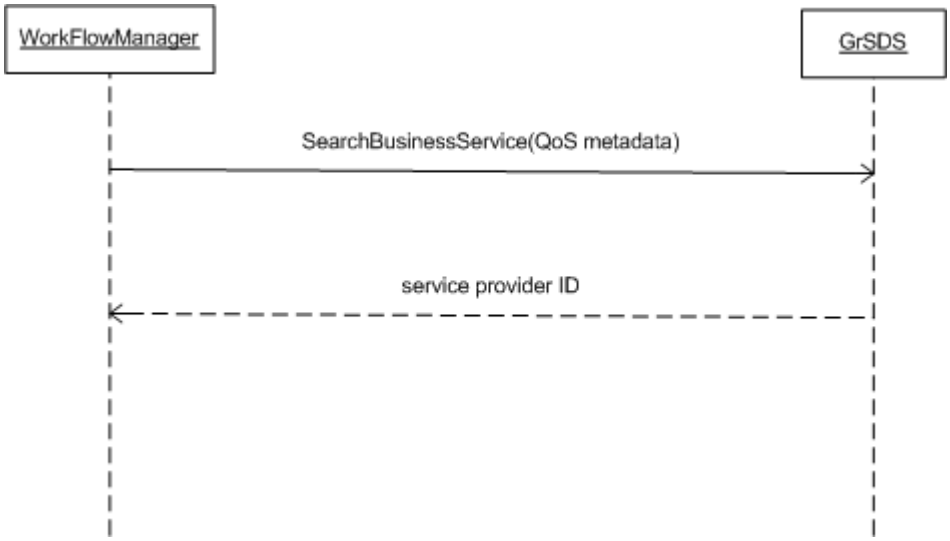


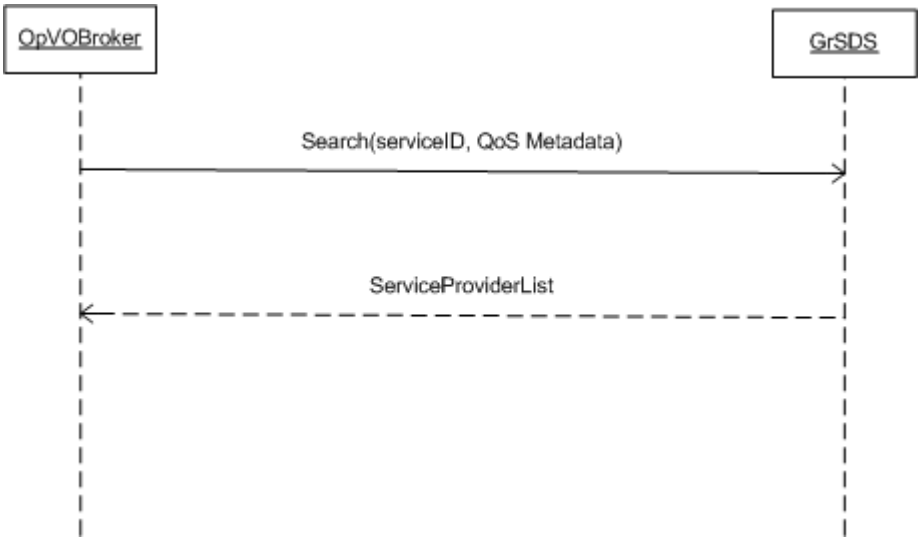
The Service provider (SP) will publish its service through the BVO Manager once it has been subscribed and registered as a participant in the Base Virtual Organisation (BVO). The SP has to provide a contract template (i.e. SLA template, where are collected QoS info, charging policies and service requirements) associated to each service and published against a SLA-Template repository previously to the proper publication of the Grid Service.

Hence, the SLA Template associated to the service should be inserted within the SLA Template Repository before the publication process has started regarding the GrSDS. Together with the WSDL document adapted to Grid Services, a reference to the corresponding SLA Template has to be supplied in order to let the GrSDS request to the SLA-Access for some basic information/parameters enclosed in the SLA Template. This approach has been chosen in order to avoid massive searches from GrSDS to SLA-Access during searching process of Grid Services. So GrSDS should also stored this basic SLA information together with the Grid Service description already stored.

Regarding the process of Grid Service search, two different components can look up for services within the GrSDS in the process of acquiring the service. Those components are the WorkFlowManager (WFManager) and the Operational Virtual Organisation Broker (OpVOBroker).

The corresponding simple sequence diagrams are:





This description represents an overall view of how other components of the architecture interact with GrSDS component. Some important features like semantics have not been addressed yet at this stage, since the design and implementation phase will be mainly concentrated on next development phase.

6.4. Implementation

The implementation of this component will be based on the well-known and established UDDI specifications. UDDI can be considered as the start point especially regarding the registry part of the GrSDS component. In addition a more fitted publication component has to be developed over UDDI in order to be able to store Grid Services (not just stateless Web Services). Moreover, semantic functionality should also be added, especially in the WS-Resource search process.

Regarding technologies, good candidates are WSRF .NET for both the publication and searching components, since up to now it seems to be the technology better developed. However, a java approach is not fully discarded. It would mainly depend on the state of the art at the moment of initiating developments.

The sequence of developments will be as follows:

- Publication module of Grid Services without semantics and without considering SLA information.
- Searching module of Grid Services without semantics and SLA information.

A second iteration will be done in order to include SLA information within both processes. Finally some semantics capabilities might be implemented depending on the project requirements.

After each iteration tests and integration effort will have to be done I order to validate and improve all the developments carried out.

6.5. Testing

Up to now, no specific testing cases have been described due to that the design and implementation phase has been delayed to next iteration. Nevertheless, it can be said that testing for GrSDS can be split out in two set of tests whether they correspond to the Publication or to the Searching process.

6.5.1. Testing for publication process

Next, we include a general list of tests that have to be carried out in order to verify the component.

Test. nr	Description	Phase 1	Phase 2
T1	**Publication process**		**X**
T 1.1	Request of a standard stateless web Service (standard WSDL)		X
T1.2	Request of a WS-Resource (WSDL)		X
T1.3	Request to SLA Access of SLA specific data		X
T1.4	Storing process of the Grid Service Description.		X
T1.5	Storing process of the SLA data		X
T1.6	Integration of the SLA info into the WS-Resource Description.		X

6.5.2. Testing for searching process

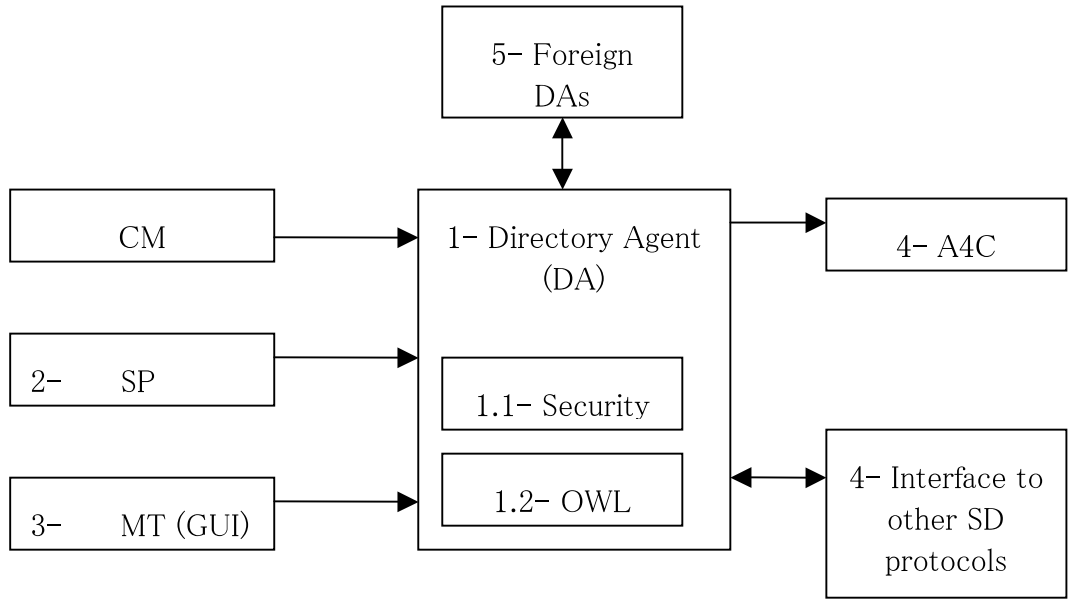
Test. nr	Description	Phase 1	Phase 2
T2	**Searching process**		**X**
T 2.1	Generic Search of a standard web Service.		X
T2.2	Generic Search of a Grid Service without SLA information.		X
T2.3	Parametric search of standard stateless Web Services.		X
T2.4	Parametric search of Grid Service.		X
T2.5	Parametric search of Grid Service with SLA information		X
T2.6	Semantic Search		

7. Local Service Discovery

7.1. LSDS Interfaces

Table 50 Local Service Discovery Server (LSDS) – Interfaces

Interface label	Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
I-LSDS-1.1	DA	MT	Direct search of Services	SLP
E-LSDS-1.2	DA	CM	Search of Local Services for user Context	SLP
	A4C	DA	Authorize User or SP	Diameter
I-LSDS-2	SP	DA	Publish/Deregister/Update Service	SLP
I-LSDS-5	DA	DA	Communicate with other DA	mSLP



7.1.1. CM-DA

This interface will help the Context Manager aggregate context information for a certain user. The Context Manager will then be able to look for services in a certain area surrounding the user. Filters will be used so that the CM can look up certain kinds of services and restricting the network traffic produced by this activity to a minimum. A function call that should be supported could be e.g.:

- SLPFindSrvsU:

Function: Find services and their descriptions restricted to a certain user and area.

Parameters: (scope, filter, user_identifier, area)

7.1.2. SP-DA

Service Providers (SP) will have to be able to advertise their services and their concrete location. In addition to this a security mechanism must exist that restricts the access to the Directory Agent (DA) to all SPs that have not been authenticated. To this end a SAML token will have to be provided by the SP in order to be authorized to use the DA. Lifetimes are suggested to be adjusted reasonably depending on the type of service used. That is, static services will have longer lifetimes than services that are on the move. Examples of function calls for this interface are:

- SLPReg

Function: Registration of a service, or update of a service description.

Parameters: url, lifetime, srvtype, attrs, location, SAML token

- SrvDeReg

Function: Deregistration of a service

Parameters: url, (SAML token)

7.1.3. MT-DA

An interface will be providing a user or modules inside the MT, with means to localize services by their own. Probably a GUI and an API can be provided with functions such as:

- SLPFindSrvsTypes

Function: Show services Types available

Parameters: SAML token

- SLPFindSrvs ->

Function: Query service in a certain area

Parameters: attrs, area, scope, SAML token

- SLPFindAttrs

Function: Query characteristics of a certain service

7.1.4. DA-A4C

In order for the LSDS to authorize a SP or a user to make use of its services the DA will contact the A4C to confirm that they are authenticated.

- Authorize

Function: Check if a user is authenticated.

Parameters: SAML token

7.1.5. DA-Other SD

The LSDS will aggregate other SD technologies such Zeroconf or the Bluetooth SDP. By doing this, a much wider spectrum of devices can be tackled. To this end, an interface will have to be provided.

7.2. Requirements

Table 51 Service Discovery – Functional requirements

Req. nr	Description	Phase 1	Phase 2
F 1	**Gather Service information**	**X**	**X**
F 1.1	Service Description	X	
F 1.1.1	Service provider description	X	
F 1.1.2	Location description	X	
F 1.1.3	Service description: Extended Characteristics		(X)
F 1.1.4	Description of how to access each service	X	
F 1.2	Directory based publishing	X	
F 1.3	Ad-hoc publishing (No directory)		(X)
F 1.4	Service publishing	X	
F 1.4.1	Service Publishing by Service Provider	X	
F 1.4.2	Service Publishing by External User		
F 2	**Query Service Information**	**X**	**X**

Req. nr	Description	Phase 1	Phase 2
F 2.1	Look up services by pattern matching	X	
F 2.2	Look up services by service type	X	
F 2.2	Look up services within attribute ranges	X	
F 2.3	Look up services by location, location range	(X)	X
F 2.4	Look up services by "semantic matching" and ontology based queries		X
F 2.5	Look up services by service provider context		(X)
F 2.6	Look up services using filtering with user context		(X)
F 2.7	Look up service by current availability to users (CM query for the services a user can reach)	X	X
F 3	**Security**		X
F 3.1	**Permits and authentication**		X
F 3.1.1	Just allow authenticated Services to publish information		X
F 3.1.2	Just allow users to query for services inside their VO or open VOs		X
F 3.1.3	Complex permission system (just allowed to query at a give time, or when something else occurred, etc…)		(X)
F 3.2	Secure communications (e.g. IPsec)		(X)
F 4	**Deregister Services**	X	
F 4.1	By Service Provider	X	
F 4.2	By Directory	X	
F 5	**Robustness in front of Service Provider Mobility**	X	X

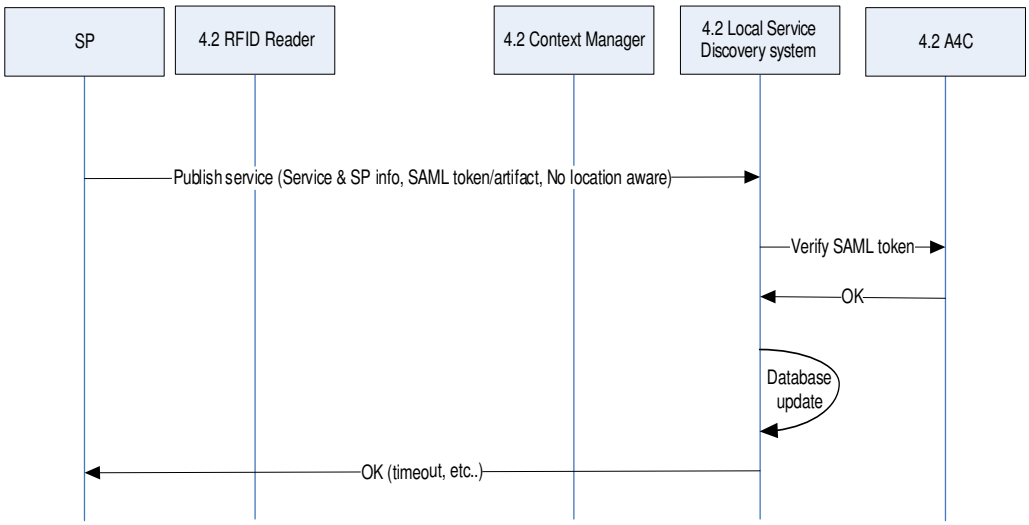
(X) Optional requirement

7.3. Design

7.3.1. Example of Message sequences

In the following example of the message interchange that can take place in the field of LSDS and its relation with other components are shown. These examples show a few different cases in which the utilization of the LSDS could be useful but are not exhaustive whatsoever.

1a) Service Publication (no location aware)

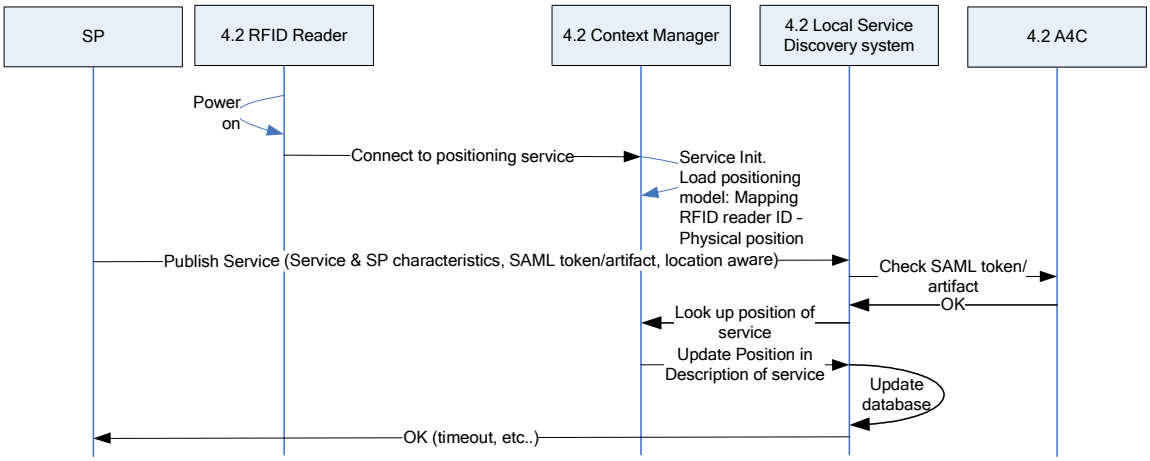


Sending a SAML token and contacting the A4C is skipped once a security association between SP and LSDS is created.

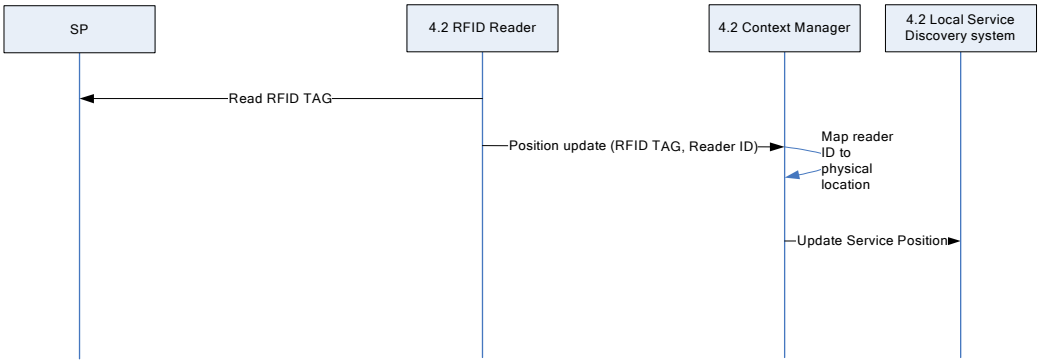
1b) Publish service taking into account position

Purpose: Handle Service positioning based on various technologies.

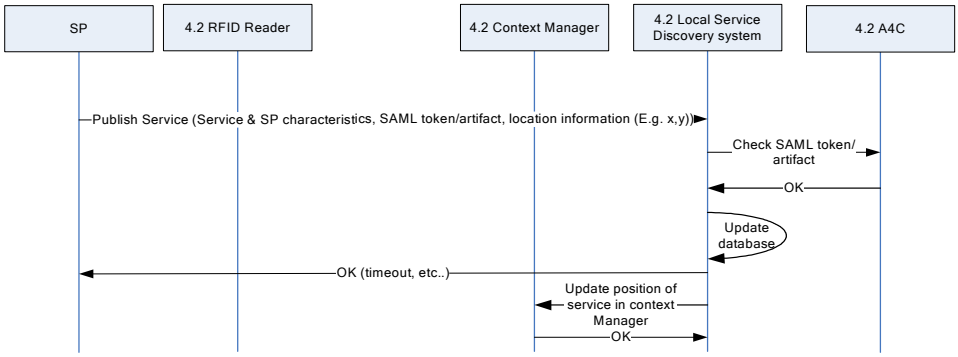
- Location manager centric: the SP is not directly aware of its position. E.g. RFID. This kind of location is not expected to be implemented for now in services. In principle we will be using RFID just to localize people whereas services that require the use of location information are supposed to be aware of their own location by technologies such as GPS.



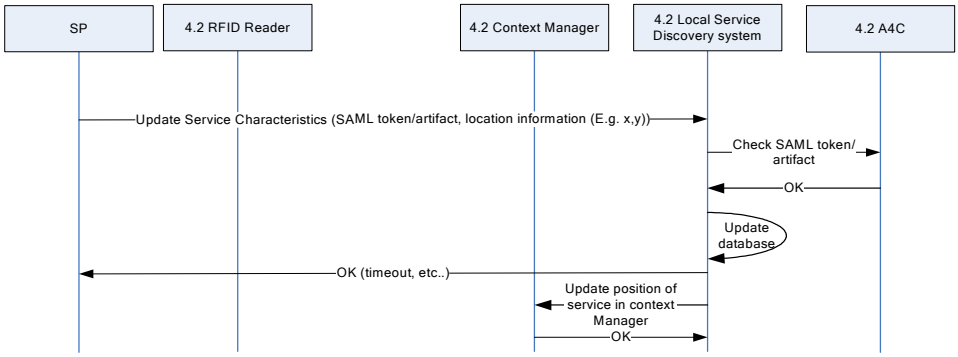
- *Updating RFID position*



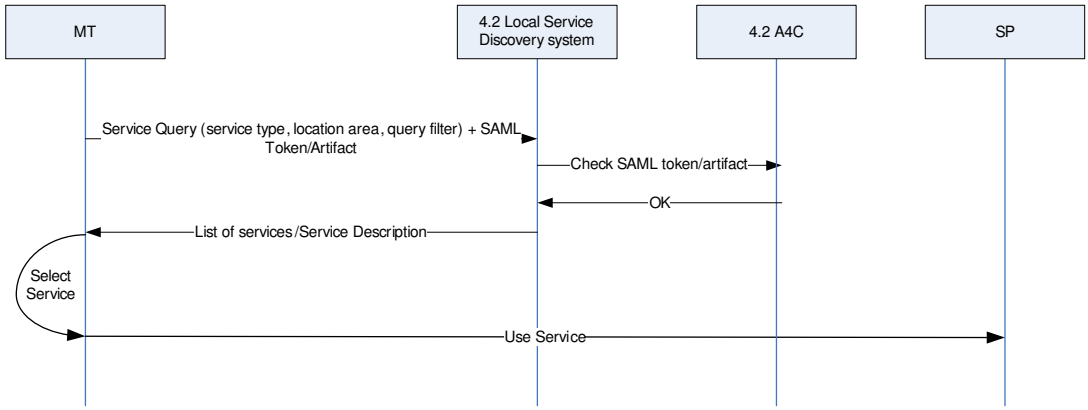
- *SP centric: The SP is aware of its position. E.g. GPS*



- *Update position*

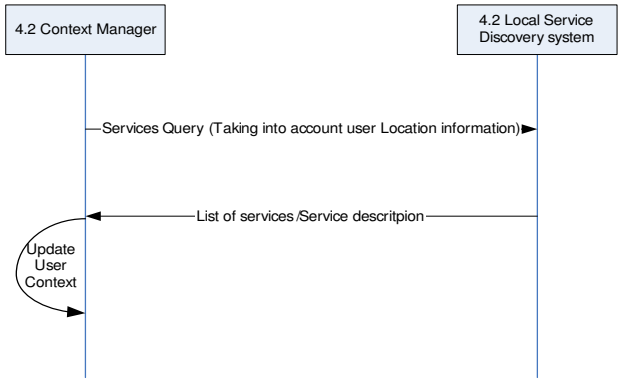


2a)Service Query by a User:

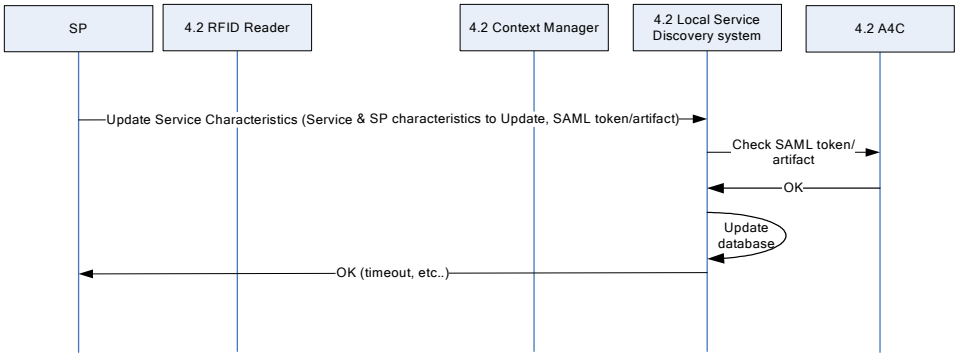


The SAML token and the connection to the A4C is skipped when a security association between MT and A4C already exists.

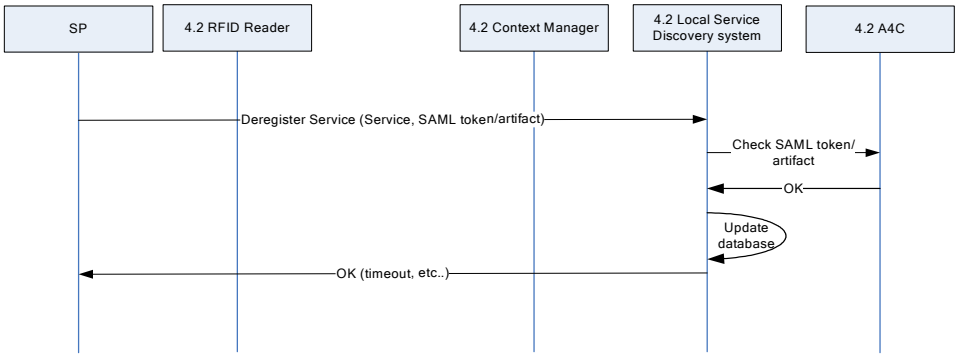
2b) Service Query by the CM



3) Update Service



4) Delete Service



7.3.2. Location awareness

Mobility implies a bunch of new possibilities in the area of network communications. While it may imply some difficulties such as a need for constant reconfiguration, different signal quality, etc… the benefits obtained from it can be overwhelming. In order to obtain the most of these, position awareness can be highly beneficial. Being or not being location aware can be like a boat following its route in the map or going adrift. To this end, it was decided that not only users would be location aware but also services. A user may then be able to localize services in its proximity that best suit his needs.

Several techniques are available in order to provide location information. A location can be described as a room, street and number, town, as coordinates in a map or global coordinates such latitude and longitude – such as the ones given by GPSs – among others. It was decided that we would provide absolute coordinates in the Globe for all the services in Akogrimo for which it makes sense to take profit of localization. We do not rule out the use of other location methods such as relative locations or e.g. room numbers on a building and so forth, since they can be compatible, however a location system based on coordinates is the best way to locate services in a certain area.

Latitude and longitude are quite extensively used by some devices to show a position in the Globe. However this system is not the handiest to calculate areas surrounding a location or calculating distances between two points, as it is based on Polar coordinates. Taking this into account, the Universal Transverse Mercator (UTM) system was developed making the calculation of the distance between two points as easy as applying the

Pythagorean Theorem. By using this system direct area queries can be performed using SLP (cf. Sec. 3.3.3) with standard LDAP filters [How97].

7.3.3. SLP

The Service Location Protocol (SLP) designed by the IETF [Gut99a, Gut99b] has shown to be very suitable to build the core of the LSDS. SLP can work either as distributed system or make use of a Directory Agent (DA) where information about all the services available is stored. The latter configuration is preferred as it is very scalable. In addition, DAs are also able to communicate thanks to an extension of the protocol called mSLP, which makes it very suitable when networks are split or across different administration areas.

7.3.4. Service Description templates

– Templates examples for a beamer (with location awareness) [Gut99].

beamer.0.1.en

 # Name of submitters: P. Mandic <patrick.mandic@rus.uni-stuttgart.de>

 # Others...

 # Language of service template: en

 # Security Considerations

 # TO BE COMPLETED

 # Abstract

 # ---------- template begins here ----------

 # Beamer / screen

template-type = beamer

template-version= 0.1

template-description = The abstract service:beamer provides advertisements for clients searching for screens with adequate characteristics. Concrete types will specialize the service:beamer type for each particular protocol (e.g. vnc)

template-url-syntax= ; Depends on the concrete service type

; To be defined on each concrete template

Resolution = STRING m

 # Indicates resolution types supported by the beamer. Posible values are: high, medium, low

width = STRING O

high = STRING O

 # Indicate the size of the screen in inches. Optional attribute

Bandwidth = STRING o m

 # Indicate which bundles the bandwidth supports. high, medium, low

utm-northing = INTEGER O

utm-easting = INTEGER O

utm-zone-number = STRING O

utm-zone-letter = STRING O

sip-address = STRING

 # sip contact address

 # Non Akogrimo specific attributes will follow here

 # End of Template

beamer-sip.0.1.en

 # Name of submitters: P. Mandic <patrick.mandic@rus.uni-stuttgart.de>

 # COMPLETE WITH THE CONTRIBUTORS LIST

 # Language of service template: en

 # Security Considerations

 # TO BE COMPLETED

 # Abstract

 # PROVIDE A SUMMARY OF THIS DOCUMENT HERE

 # SIP-controlled beamer Concrete Service Type

 # ---------- template begins here ----------

template-type = beamer:sip

template-version = 0.1

template-description = The service:beamer:sip type provides advertisements for clents seeking SIP-controlled beamers

; Example of a posible definition of template-url-syntax (to be defined by the SIP people ;))

; template-url-syntax= "request=" action

; action = "handover" / "remotecontrol"

; service:beamer:sip://beamer123@akogrimo.org/request=remotecontrol

 # All attributes are inherited from the abstract type

 # End of template

7.4. LSDS Implementation

The implementation will take place taking as a base the OpenSLP. OpenSLP is a free implementation of the SLP protocol. The current development version (1.3) is the first version with IPv6 support and therefore is the chosen version to work with.

Since OpenSLP 1.3 is written in C, this language is the programming language used for the LSDS.

7.5. LSDS Testing

Corresponding Test-cases will be developed to validate the correct functionality of the software. In addition to that, some of the scenarios defined by the MSCs from Section

3.3.2 can be corroborated. A test-bed will be build to physically test the interaction with other components.

Annex A. References

[Blu98] L. Blunk, J. Vollbrecht. „PPP Extensible Authentication Protocol (EAP)", IETF RFC 2284, March 1998. URL: http://www.ietf.org/rfc/rfc2284.txt Last visited 27.10.2005

[Cal03] P. Calhoun, J. Loughney, E. Guttman, G. Zorn, J. Arkko. "Diameter Base Protocol", IETF RFC 3588, September 2003. URL: http://www.ietf.org/rfc/rfc3588.txt Last visited 27.10.2005

[Cal05] P. Calhoun, G. Zorn, D. Spence, D. Mitton. "Diameter Network Access Server Application", IETF RFC 4005, August 2005. URL: http://www.ietf.org/rfc/rfc4005.txt Last visited 27.10.2005

[D4.2.1] J. Wedvik et al: "Overall Network Middleware Requirements Report". Akogrimo Deliverable D4.2.1. 01.09.2005, v 1.1. URL: http://www.akogrimo.org/modules.php?name=UpDownload&req=getit&lid=39 Last visited 21.10.2005

[D5.1.2] Akogrimo: "Integrated Prototype". Akogrimo Deliverable D5.1.2. Scheduled for January 2005.

[DaidalosIDToken] B. Weyl et al. " Protecting Privacy of Identities in Federated Operator Environments", Proceedings of the 14th IST Mobile & Wireless Communications Summit, Dresden, 2005. URL: http://www.ikr.uni-stuttgart.de/Content/Publications/View/Frame/FullRecord.html?36435 Last visited 27.10.2005

[For05] D. Forsberg, Y. Ohba, B. Patil, H. Tschofenig, A. Yegin. "Protocol for Carrying Authentication for Network Access (PANA)", IETF Draft, 16 July 2005. URL: http://www.ietf.org/internet-drafts/draft-ietf-pana-pana-10.txt Last visited 27.10.2005

[Gut99a] E. Guttman et al: "Service Location Protocol, Version 2," IETF, RFC 2608, June 1999. URL = http://www.rfc-editor.org/rfc/rfc2608.txt Last visited 27.10.2005

[Gut99b] (former [Per99]) E. Guttman et al, "Service Templates and Service: Schemes" IETF RFC 2609. URL: http://www.ietf.org/rfc/rfc2609.txt Last visited 27.10.2005

[How97] T. Howes: "The String Representation of LDAP Filters", IETF RFC 2254. December 1997. URL: http://www.ietf.org/rfc/rfc2254.txt Last visited 27.10.2005

[IANA05] IANA, RADIUS Types, 20 September 2005. URL: http://www.iana.org/assignments/radius-types Last visited 28.10.2005

[JNI] Java Native Interface (JNI), http://java.sun.com/docs/books/tutorial/native1.1/ Last visited 28.10.2005

[MLP] Open Mobile Alliance (OMA): "Mobile Location Protocol (MLP)". OMA Specification no OMA-LIF-MLP-V3_1-20040316-C. 16.03.2004, ver 3.1. URL: http://www.openmobilealliance.org/release_program/docs/MLP/OMA-LIF-MLP-V3_1-20040316-C.pdf Last visited 21.10.2005

[MySQL] MySQL Database. URL: http://www.mysql.com Last visited 28.10.2005

[Nie04] A. Niemi, "Session Initiation Protocol (SIP) Extension for Event State Publication", RFC 3903, Internet Engineering Task Force, October 2004. URL: http://www.ietf.org/rfc/rfc3903.txt Last visited 28.10.2005

[OPENDIAM] Open Diameter Webpage. URL: http://www.opendiameter.org Last visited 28.10.2005

[openslp] OpenSLP Webpage. URL www.openslp.org Last visited 21.10.2005

[Roa02] A. B. Roach, "Session Initiation Protocol (SIP)-Specific Event Notification", RFC 3265 Internet Engineering Task Force, June 2002. URL: http://www.ietf.org/rfc/rfc3265.txt Last visited 28.10.2005

[Ros04] J. Rosenberg, "A Presence Event Package for the Session Initiation Protocol", RFC 3856 Internet Engineering Task Force, August 2004. URL: http://www.ietf.org/rfc/rfc3856.txt Last visited 28.10.2005

[SAML] OASIS: "Security Assertion Markup Language, SAML", October 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security Last visited 28.10.2005

[SAMLAssertion] P. Hallam-Baker et al. "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML)", OASIS, November 2002. URL: http://www.oasis-open.org/committees/download.php/ 1371/oasis-sstc-saml-core-1.0.pdf Last visited 28.10.2005

[SAMLProfile] E. Maler et al. "Bindings and Profiles for the OASIS Security Assertion Markup Language", OASIS, September 2003. URL: http://www.oasis-open.org/committees/download.php/3405/oasis-sstc-saml-bindings-1.1.pdf Last visited 28.10.2005

[Schu05] H. Schulzrinne, V. Gurbani, "RPID: Rich Presence Extensions to the Presence Information Data Format (PIDF)". IETF Draft, September 2005. URL: http://www.ietf.org/internet-drafts/draft-ietf-simple-rpid-09.txt Last visited 28.10.2005

[SER] SIP Express Router homepage. URL: http://www.iptel.org/ser/ Last visited 28.10.2005

[SIPComm] "SIP Communicator - a Java SIP User Agent built on top of the JAIN-SIP-RI and JMF". Software project at Network Research Team, Louis Pasteur University - Strasbourg, France. URL: http://www.sip-communicator.org Last visited 19/10/2005

[SOAP] D. Box et al., "Simple Object Access Protocol (SOAP) 1.1", World Wide Web Consortium Note, May 2000. URL: http://www.w3.org/TR/SOAP Last visited 28.10.2005

[Sug04] H. Sugano, S. Fujimoto, "Presence Information Data Format (PIDF)". IETF RFC 3863, August 2004. URL: http://www.ietf.org/rfc/rfc3863.txt Last visited 28.10.2005

[UAProf] Open Mobile Alliance (OMA): "WAG UAProf - Wireless Application Protocol". OMA Specification no WAP-248-UAPROF-20011020-a. 20.10.2001. URL: http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf Last visited 21.10.2005

Annex B. Context Manager – details

B.1. Context Consumer interface

B.1.1. ContextManagerWS.wsdl

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

<wsdl:definitions                targetNamespace="http://ws.contextmanager.wp42.akogrimo.org"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://ws.contextmanager.wp42.akogrimo.org"
xmlns:intf="http://ws.contextmanager.wp42.akogrimo.org"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.2.1
Built on Jun 14, 2005 (09:15:57 EDT)-->
 <wsdl:types>
  <schema                        targetNamespace="http://ws.contextmanager.wp42.akogrimo.org"
xmlns="http://www.w3.org/2001/XMLSchema">
   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
   <complexType name="ArrayOf_xsd_int">
    <complexContent>
     <restriction base="soapenc:Array">
      <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:int[]"/>
     </restriction>
    </complexContent>
   </complexType>
  </schema>
 </wsdl:types>
   <wsdl:message name="unsubscribeRequest">
      <wsdl:part name="callbackUrl" type="xsd:string"/>
      <wsdl:part name="userId" type="xsd:int"/>
   </wsdl:message>
   <wsdl:message name="subscribeResponse">
      <wsdl:part name="subscribeReturn" type="xsd:int"/>
   </wsdl:message>
   <wsdl:message name="subscribeRequest">
      <wsdl:part name="callbackUrl" type="xsd:string"/>
      <wsdl:part name="userId" type="xsd:int"/>
      <wsdl:part name="fromDate" type="xsd:dateTime"/>
      <wsdl:part name="toDate" type="xsd:dateTime"/>
      <wsdl:part name="contextTypes" type="impl:ArrayOf_xsd_int"/>
   </wsdl:message>
   <wsdl:message name="unsubscribeResponse">
      <wsdl:part name="unsubscribeReturn" type="xsd:int"/>
   </wsdl:message>
   <wsdl:portType name="ContextManagerWS">
      <wsdl:operation  name="subscribe"  parameterOrder="callbackUrl  userId  fromDate  toDate
contextTypes">
         <wsdl:input message="impl:subscribeRequest" name="subscribeRequest"/>
         <wsdl:output message="impl:subscribeResponse" name="subscribeResponse"/>
      </wsdl:operation>
      <wsdl:operation name="unsubscribe" parameterOrder="callbackUrl userId">
         <wsdl:input message="impl:unsubscribeRequest" name="unsubscribeRequest"/>
         <wsdl:output message="impl:unsubscribeResponse" name="unsubscribeResponse"/>
      </wsdl:operation>
   </wsdl:portType>
   <wsdl:binding name="ContextManagerWSSoapBinding" type="impl:ContextManagerWS">
      <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
      <wsdl:operation name="subscribe">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="subscribeRequest">
            <wsdlsoap:body           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.contextmanager.wp42.akogrimo.org" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="subscribeResponse">
            <wsdlsoap:body           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.contextmanager.wp42.akogrimo.org" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
```

```
      <wsdl:operation name="unsubscribe">
         <wsdlsoap:operation soapAction=""/>
         <wsdl:input name="unsubscribeRequest">
            <wsdlsoap:body           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.contextmanager.wp42.akogrimo.org" use="encoded"/>
         </wsdl:input>
         <wsdl:output name="unsubscribeResponse">
            <wsdlsoap:body           encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
namespace="http://ws.contextmanager.wp42.akogrimo.org" use="encoded"/>
         </wsdl:output>
      </wsdl:operation>
   </wsdl:binding>
   <wsdl:service name="ContextManagerWSService">
      <wsdl:port binding="impl:ContextManagerWSSoapBinding" name="ContextManagerWS">
         <wsdlsoap:address
location="http://localhost:8080/ContextManagerWS/services/ContextManagerWS"/>
      </wsdl:port>
   </wsdl:service>
</wsdl:definitions>
```

B.1.2. ContextManagerWS.java

```
package org.akogrimo.wp42.contextmanager.ws;

import java.net.MalformedURLException;
import java.rmi.Naming;
import java.rmi.NotBoundException;
import java.rmi.RemoteException;
import java.util.Calendar;
import java.text.*;


import org.akogrimo.wp42.contextmanager.rmi.ContextManagerServer;

public class ContextManagerWS {
       private DateFormat mDateFormat;

       public ContextManagerWS() {
              mDateFormat = new SimpleDateFormat("MM-dd-yyyy HH:mm:ss");
       }


       public  int  subscribe(String  callbackUrl,  int  userId,  Calendar  fromDate,  Calendar
toDate, int[] contextTypes) {

       System.out.println("subscribeUser:\n\tcallbackUrl="+callbackUrl+"\n\tuserId="+userId);
              System.out.println("\tfromDate="+mDateFormat.format(fromDate.getTime()));
              System.out.println("\tfromDate="+mDateFormat.format(toDate.getTime()));
              for (int i=0; i<contextTypes.length;i++) {
                     System.out.println("\tContextType="+contextTypes[i]);
              }
              try {
               // Look up the remote object
                     ContextManagerServer     contextManager     =     (ContextManagerServer)
Naming.lookup("//localhost/ContextManagerServer");

                     contextManager.subscribe(userId, callbackUrl);
               return 0;

           } catch (MalformedURLException e) {
              System.out.println(e);
           } catch (java.rmi.UnknownHostException e) {
              System.out.println(e);
           } catch (NotBoundException e) {
              System.out.println(e);
           } catch (RemoteException e) {
              System.out.println(e);
           }
              return 1;
       }

       public int unsubscribe(String callbackUrl, int userId) {
              System.out.println("unsubscribe: callbackUrl="+callbackUrl+", userId="+userId);
              try {
               // Look up the remote object
```

```
                    ContextManagerServer     contextManager     =     (ContextManagerServer)
Naming.lookup("//localhost/ContextManagerServer");

                    contextManager.unsubscribe(callbackUrl);
                return 0;

            } catch (MalformedURLException e) {
                System.out.println(e);
            } catch (java.rmi.UnknownHostException e) {
                System.out.println(e);
            } catch (NotBoundException e) {
                System.out.println(e);
            } catch (RemoteException e) {
                System.out.println(e);
            }
                return 1;
        }


}
```

B.2. Test details

B.2.1. Context manager Client for mobile terminal

The initial testing may be done against the UAProf of the Sony Ericsson P900 UAProf. A small test program utilising the UAProfHandler package. The information in UAProfInformation is displayed on screen if the document is available.

The first test gives the correct UAProf URL and the following shows the information expected in the test. Note test against a local copy of the profile.

Element Name	Expected result	Result
Vendor	Sony Ericsson Mobile Communications	
Model	P900R101	
UAProfURL	http://localhost:8080/P900R101.xml	
ColorCapable	Yes	
ScreenSize	208x320	
ImageCapable	Yes	
Keyboard	OnScreenQwerty	
ScreenSizeChar	20x15	
SoundOutputCapable	Yes	
TextInputCapable	Yes	
VoiceInputCapable	Yes	

The second test gives a wrong URL and an error message should be shown and no information should be displayed about the UAProfInfo