D4.2.4

Consolidated Integrated Services Design and Implementation Report

Version 1.0

WP 4.2 Mobile Network Middleware Architecture, Design and Implementation

Dissemination Level: Public

Lead Editor: Fredrik Solsvik, Telenor

14/08/2007

Status: Final

SIXTH FRAMEWORK PROGRAMME

PRIORITY IST-2002-2.3.1.18

Information Society

Grid for complex problem solving

Proposal/Contract no.: 004293

Projectsite: www.mobilegrids.org

This license allows you to

- to copy, distribute, display, and perform the work
- to make commercial use of the work

Under the following conditions:

Attribution. You must attribute the work by indicating that this work originated from the IST-Akogrimo project and has been partially funded by the European Commission under contract number IST-2002-004293

No Derivative Works. You may not alter, transform, or build upon this work without explicit permission of the consortium

- For any reuse or distribution, you must make clear to others the license terms of this work.
- Any of these conditions can be waived if you get permission from the copyright holder.

This is a human-readable summary of the Legal Code below:

License

THE WORK (AS DEFINED BELOW) IS PROVIDED UNDER THE TERMS OF THIS CREATIVE COMMONS PUBLIC LICENSE ("CCPL" OR "LICENSE"). THE WORK IS PROTECTED BY COPYRIGHT AND/OR OTHER APPLICABLE LAW. ANY USE OF THE WORK OTHER THAN AS AUTHORIZED UNDER THIS LICENSE OR COPYRIGHT LAW IS PROHIBITED.

BY EXERCISING ANY RIGHTS TO THE WORK PROVIDED HERE, YOU ACCEPT AND AGREE TO BE BOUND BY THE TERMS OF THIS LICENSE. THE LICENSOR GRANTS YOU THE RIGHTS CONTAINED HERE IN CONSIDERATION OF YOUR ACCEPTANCE OF SUCH TERMS AND CONDITIONS.

1. Definitions

a. **"Collective Work"** means a work, such as a periodical issue, anthology or encyclopedia, in which the Work in its entirety in unmodified form, along with a number of other contributions, constituting separate and independent works in themselves, are assembled into a collective whole. A work that constitutes a Collective Work will not be considered a Derivative Work (as defined below) for the purposes of this License.

b. **"Derivative Work"** means a work based upon the Work or upon the Work and other pre-existing works, such as a translation, musical arrangement, dramatization, fictionalization, motion picture version, sound recording, art reproduction, abridgment, condensation, or any other form in which the Work may be recast, transformed, or adapted, except that a work that constitutes a Collective Work will not be considered a Derivative Work for the purpose of this License. For the avoidance of doubt, where the Work is a musical composition or sound recording, the synchronization of the Work in timed-relation with a moving image ("synching") will be considered a Derivative Work for the purpose of this License.

c. **"Licensor"** means all partners of the Akogrimo consortium that have participated in the production of this text

d. **"Original Author"** means the individual or entity who created the Work.

e. **"Work"** means the copyrightable work of authorship offered under the terms of this License.

f. **"You"** means an individual or entity exercising rights under this License who has not previously violated the terms of this License with respect to the Work, or who has received express permission from the Licensor to exercise rights under this License despite a previous violation.

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 & implementation
WP 4.2	**Mobile Network Middleware Architecture, Design and Implementation**
Task 4.2.1	**Integrated service platform and middleware for NGG**
Dependencies	**This deliverable depends on and has influence on (to some extent) work carried out in WP 3.1, 3.2, 4.1, 4.3, 4.4, 5.1, 5.2.**

Contributors:	*Reviewers:*
Vicente Olmedo (UPM) Víctor A. Villagrá (UPM) Fredrik Solsvik (Telenor) Hasan (University of Zurich) David Hausheer (University of Zurich) Cristian Morariu (University of Zurich) Peter Racz (University of Zurich) David Lutz (USTUTT) Patric Mandic (USTUTT) Juan E. Burgos (TID) Bienvenido Gómez (TID)	Internal review by WP4.2 participants. Review by Akogrimo partners external to WP 4.2: Nuno Inacio (IT Aveiro) Kleopatra Konstanteli (NTUA) Francesco D'Andria (ATOS) Alfonso Sánchez-Macián (IT-In) Guiseppe Laria (CRMPA) Annalisa Terracina (DM)

Approved by: QM

Version	Date	Authors	Sections Affected
0.1	14.07.2007	Fredrik Solsvik	Title, version, summary, frontpages and introduction
0.2	14.07.2007	UniZH	Updated the A4C section with the section submitted by Hasan
0.3	14.07.2007	Fredrik Solsvik	Minor changes in the CM section to reflect the small changes since last deliverable

Version	Date	Authors	Sections Affected
0.4	17.07.2007	Hasan	Minor update of A4C section and text formatting
0.5	24.07.2007	Vicente Olmedo	Minor update of SIP PUA section Added SIP SD implementation section
0.6	27.07.2007	Juan E. Burgos	Update of SIP SD implementation section Final edition for external review
0.7	01.02.2007	Juan E. Burgos	Official template Inclusion of review comments related to chapters 2 and (partly) 6
0.8	02.08.2007	Hasan	Update on A4C section taking into account (internal) reviewer comments
0.9	10.08.2007	Juan E. Burgos	Added subsection in Introduction with differences between D4.2.3 and D4.2.4 from some partners Update on Context Manager section taking into account (internal) reviewer comments
0.10	13.08.2007	Juan E. Burgos	Update introduction chapter (some comments from Alfonso and the description on what's new of the remaining components). Update on chapter 6, considering most of the comments from Alfonso
0.11	14.08.2007	Fredrik Solsvik	Finalized document for QM approval
1.0	14.08.2007	Fredrik Solsvik	Final version based on review by QM

Table of Contents

List of Figures

List of Tables

Abbreviations

Term	**Full text**
A4C	Authentication, Authorization, Accounting, Auditing and Charging
AA	Authentication and Authorization
Akogrimo	Access To Knowledge through the Grid in a Mobile World
API	Application Programming interface
AR	Access Router
BT	Bluetooth
BVO	Base Virtual Organization
CA	Certification Authority
CM	Context Manager
CPU	Central Processing Unit
CRMPA	Centro di Ricerca in Matematica Pura ed Applicata
CV	Context Viewer
DA	Directory Agent
DD	Device Discovery
DOW	Description of Work
DTD	Document Type Definition
EAP	Extensible Authentication Protocol
EMS	Execution Management System
EPA	Event Publication Agent
EPR	EndPointReference (WS-N)
ESA	Event Subscription Agent
ESC	Event State Compositor
FAT	Factory Acceptance Test
GPS	Global Positioning System
GrSDS	Grid Service Discovery System
GW	Gateway

Term	Full text
ID	Identifier
IdP	Identity Provider
IETF	Internet Engineering Task Force
IM	Instant Messaging
IMS	Information Management System
IP	Internet Protocol
IPsec	IP Security
JAIN	JAVA APIs for Integrated Networks
JMF	Java Media Framework
JNI	Java Native Interface
LSDS	Local Service Discovery System or Life Signs Detection System
MAC	Media Access Control
MSC	Message Sequence Chart
MT	Mobile Terminal
ND	Network Discovery
NGG	Next Generation Grid
OGC	Open Geospatial Consortium
OGSA	Open Grid Services Architecture
OS	Operating System
OWL	Ontology Web Language
PA	Presence Agent
PAA	PANA Authentication Agent
PaC	PANA client
PANA	Protocol for carrying Authentication for Network Access
PC	Personal Computer
PDA	Personal Digital Assistant

Term	Full text
PIDF	Presence Information Data Format
PKI	Public Key Infrastructure
PS	Presence Server
PUA	Presence UA
QoS	Quality of Service
RA	Registration Authority
RDD	Resource Description Document
RFC	Request For Comment
RFID	Radio Frequency Identification
RMI	Remote Method Invocation
RPID	Rich Presence Information Data
RSDD	Required Service Description Document
RTT	Round-Trip Time
SA	Service Agent
SAML	Security Assertion Markup Language
SAT	System Acceptance Test
SD	Service Discovery
SDA	Service Discovery Agent
SDD	Service Description Document
SDS	Service Discovery Subscriber
SER	SIP Express Router
SIdP	Shibboleth Identity Provider
SIMPLE	SIP for Instant Messaging and Presence Leveraging Extensions
SIP	Session Initiation Protocol
SLA	Service Level Agreement
SLO	Service-level Objective
SLP	Service Location Protocol

Term	Full text
SOAP	Simple Object Access Protocol
SP	Service Provider
SPA	Service Publication Agent
SQL	Structured Query Language
SSO	Single Sign-On
SW	Software
TID	Telefonica I+D
TLS	Transport Layer Security
TN	Telenor
UA	User Agent
UAProf	User Agent Profile
UDDI	Universal Description, Discovery and Integration
UNIZH	University of Zurich
UPM	Universidad Politecnica de Madrid
UpnP	Universal Plug and Play (Microsoft)
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USTUTT	University of Stuttgart
UTM	Universal Transverse Mercator
VNC	Virtual Network Computing
VO	Virtual Organization
VRA	Violation-Report-Answer
VRR	Violation-Report-Request
WAYF	Where-Are-You-From
WLAN	Wireless Local Area Network
WP	Work Package
WS	Web Services

Term	**Full text**
WS-N	WS Notification
WSDL	Web Services Description Language (XML format for describing network services)
XML	Extensible Markup Language

Executive summary

This document describes the functionality implemented as part of the network middleware layer of the Akogrimo architecture. It is an updated version of [D4.2.3] Final Integrated Services Design and Implementation Report and should be considered an updated version of that deliverable.

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 (services in the proximity of the end user)
- Presence, identity 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 Akogrimo [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 Work Packages

- 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

1. Introduction

Scope

The purpose of this document is to describe functionality developed within Akogrimo Work Package (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 for some components. The described functionality will serve as input for WP51. This document is an updated version of D4.2.3, Final Integrated Services and Implementation Report and should be considered as such.

While SIP Presence and Identity Management subsections remain unchanged (chapters 2 and 5 respectivelly), the A4C section has been updated to reflect the final implementation status. Main updates include the implementation details of the A4C client and server, the complete interface specification, the updated A4C database description, the description of the graphical user interface of the A4C server, and security considerations. In Chapter 5 – Context Manager several changes have been done: updated implementation status, minor changes in interfaces section (describing GetUserEPR method), RFID software section (mentioning Tag Reader which is used in the latest demonstration) and tools section (renamed the Location Simulator from RFIDDummyServer). Finally, implementation details have been added to section 6 regarding SIP Service Discovery.

This document is meant as a complete report including the latest work in the Work Package.

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.

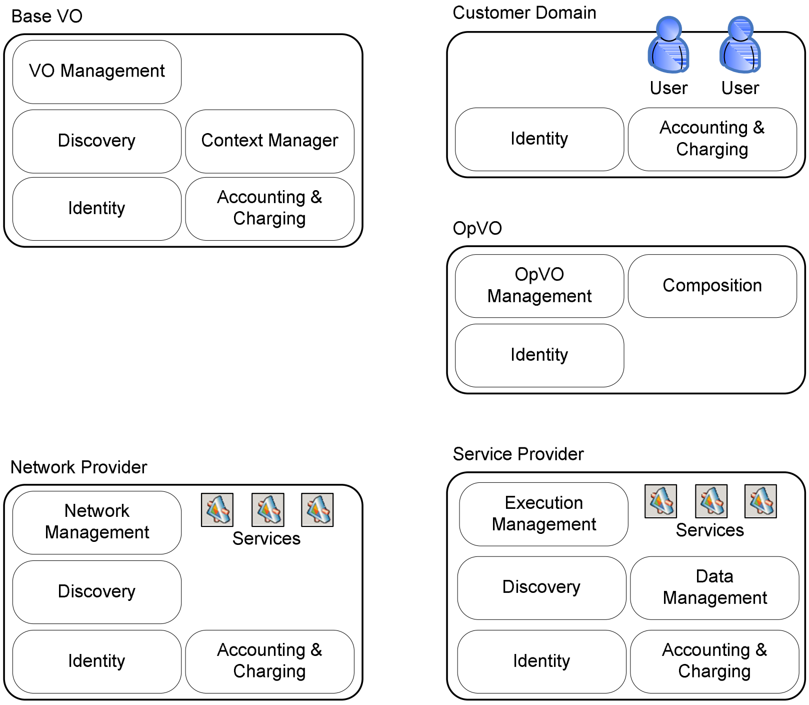


Figure 1 WP42 modules in a logical Akogrimo architecture

Figure 1 provides a logical overview of the Akogrimo architecture. The focus of the Akogrimo project is on these elements, and is described in deliverables from WP 3.1, WP 4.1, WP 4.3, WP 4.4 and this one.

Network Middleware Layer

The purpose of the Network Middleware layer is to provide a set of basic infrastructure functions including cross-layer Authentication, Authorization, Accounting Auditing and Charging (A4C), presence and context management, and semantic service discovery. Implementation of functionality in this layer is also the topic of this document. The different modules are shown in Figure 2. In this document we describe design and implementation for the Mobile Network Middleware layer. This description focuses on the following parts:

- Session Initiation Protocol (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 DS and GrSDS - Grid Service DS) Registration of and search functionality for local (close to the user) and Grid services

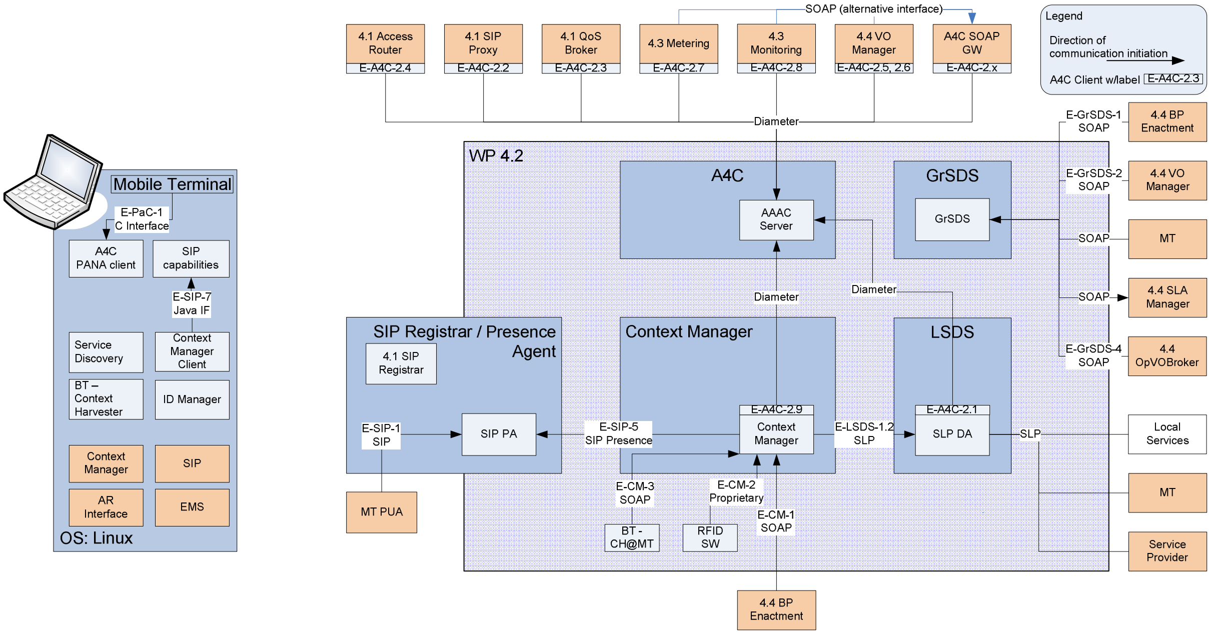


Figure 2 WP42 modules in a technical Akogrimo architecture with interfaces

2. SIP Presence handling

As explained in D422 [D4.2.2], SIP is one of the chosen mechanisms to provide context information within the scope of the Akogrimo project. The SIP Presence handling framework, as part of the generic context provisioning architecture, is depicted in the following figure, where the SIP interactions domain has been highlighted:

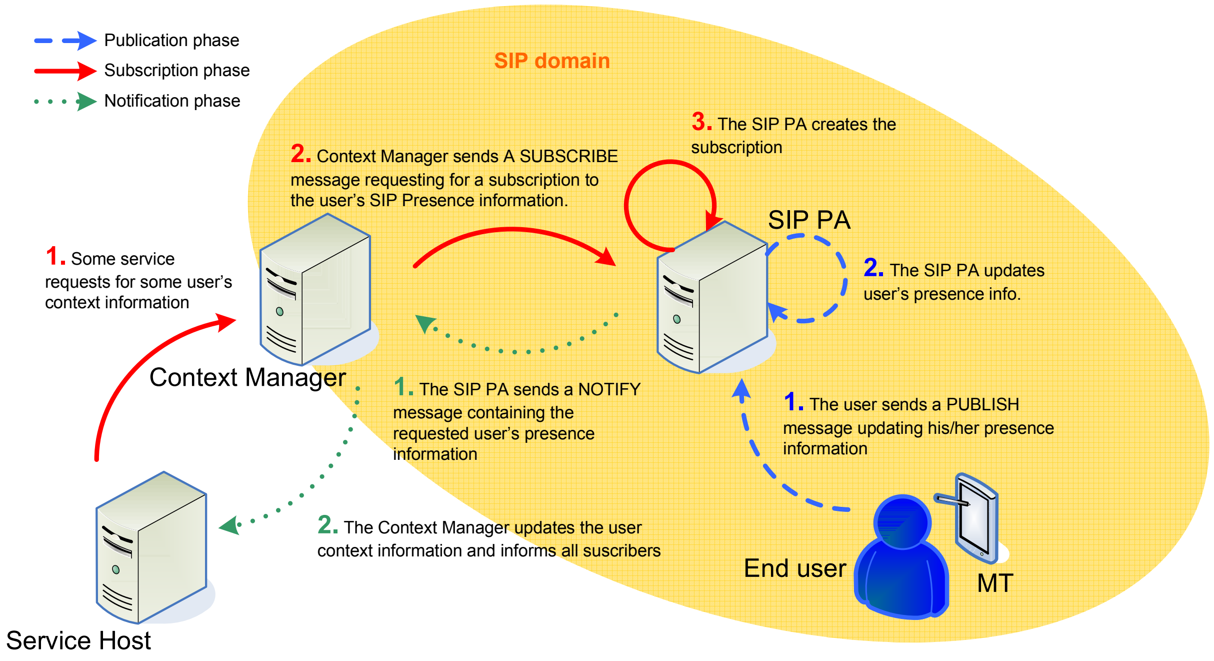

Publication phase
Subscription phase
Notification phase
SIP domain
3. The SIP PA creates the subscription
2. Context Manager sends A SUBSCRIBE message requesting for a subscription to the user's SIP Presence information.
SIP PA
1. Some service requests for some user's context information
2. The SIP PA updates user's presence info.
Context Manager
1. The SIP PA sends a NOTIFY message containing the requested user's presence information
1. The user sends a PUBLISH message updating his/her presence information
2. The Context Manager updates the user context information and informs all suscribers
End user
MT
Service Host

Figure 3 SIP presence framework

Each colour/type of line represents a different phase of the presence information flow:

- Publication phase: some presentity (a SIP entity whose presence information is going to be stored and distributed) makes public its presence information.
- Subscription phase: some presence watcher request for presence information.
- Notification phase: the presence watcher is notified with the presentity presence information.

The requirements, basics and most of the architectural details were described in D421 [D4.2.1] and D422 [D4.2.2].

From an architectural and implementation perspective, the SIP presence framework in Akogrimo comprises two main components: the Presence Agent (PA) which is part of the network infrastructures, and a Presence User Agent (PUA) running at the mobile terminals. Additionally, the Context Manager implements a subscription agent based on the SIP Communicator in order to perform the required presence subscriptions. This section is intended to provide a description of the new features that have been added to these components.

2.1. Presence Agent (PA)

SIP Presence Agent handles SIP presence subscriptions and publications, and generates the corresponding notifications when the status of certain presentity changes. A detailed description of the architecture can be found at D422 [D4.2.2].

SIP Presence Agent has been built as a dynamic C library embedded on SIP Server, which will be implemented starting from the SIP Express Router (an open-source implementation) and will run in a Linux/Ubuntu machine; however, an alternative implementation that has a separate server hosting the Presence Agent – based on the application server being evaluated to support the SIP Service Discovery Service – is also being considered. Some of the advantages of this alternative implementation would be:

- Availability of a Java implementation (since it would be implemented as a Java servlet instead of a C library).
- Code reutilisation (since SIP Service Discovery is also based on the SIP event framework).
- A better alignment with the IMS architecture.

2.1.1. Requirements

The following table summarises the status of the functional/non-functional requirements described in D422. Note that some of them have been enhanced, while others have been deprecated, as a result from the conclusions we have obtained from the first cycle.

Table 1 SIP PA – Functional requirements

Req. nr	Description	Status [1]
F 1	**Presence/Context document handling**	
F 1.1	PUBLISH method support	Supported.
F 1.2	Presence Information Data Format (PIDF) format support	Supported. New: document integrity checking support, configurable by the user.
F 1.3	RPID format support	Supported. New: document integrity checking support, configurable by the user.
F 1.4	Default event state (when no info. from the user is available)	Supported. New: configurable by the user
F 1.5	Event State Compositor (aggregation from different sources)	Supported, but unused.
F 1.6	State deltas support (users provide only the information that changed, but the PA maintains the complete state)	Deprecated. It is only an optimisation issue, and implementation efforts have been reallocated to other tasks.
F 1.7	Permanent storage (database support)	It is an optional requirement, just to protect information against a server shutdown. Not implemented.
F 2	**Subscriptions handling**	
F 2.1	SUBSCRIBE method support	Supported
F 2.2	NOTIFY method support	Supported

Req. nr	Description	Status [1]
F 2.3	Presence Authorisation policies (who is allowed to request presence information)	Supported, configurable by the PA administrator.
F3	**Generic**	
F 3.1	Fully RFC compliance (all possible response codes for all possible "abnormal" situations)	Work in continuous progress. Several bugs were detected and fixed.
F 3.2	Access to the stored information (publications and subscriptions) using the SER command-line tool	**Added**, for monitoring purposes.
N	**Non-functional**	
N1	Running on Linux (Ubuntu) on a PC platform	Done
N2	Performance improvement (efficiency, speed)	Mostly done.

[1] Status row references only the status for the implemented solution (based on SER and co-located with the SIP Sever)

2.1.2. Design

The detailed description regarding the SIP Presence Agent design provided at D4.2.2 [D4.2.2] is still valid. For clarification purposes, we have included also here the figure describing the internal SIP Server architecture, showing 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).

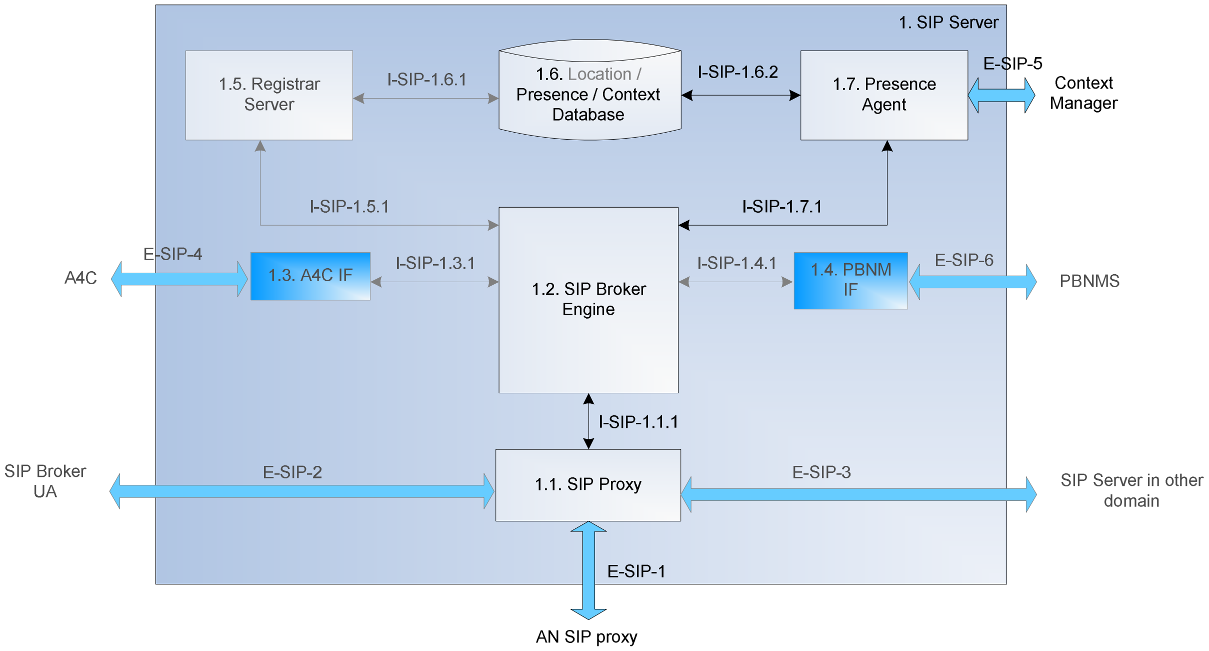


Figure 4 The Akogrimo SIP Server and the embedded SIP Presence Agent

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 of information 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 embedded SIP proxy, play some roles in the WP42 interactions

The SIP Broker Engine, as the core part of the SIP Server, implements the interconnection logic between the different SIP entities as well as the interfaces with other network entities. 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 as a listener on 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 sub-module 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.

Note that even if the Presence Agent was not co-located with the SIP Server (as the alternative implementation based on the application server is evaluating) the proposed design would not be significantly modified, as Figure 5 describes; the SIP Server's embedded SIP proxy would deliver all SIP presence-related requests and responses to the external PA using the external SIP interface being used to forward all requests that the SIP Server is not going to process (E-SIP-3). Obviously, current internal I-SIP-1.6.2 and I-SIP-1.7.1 would disappear.

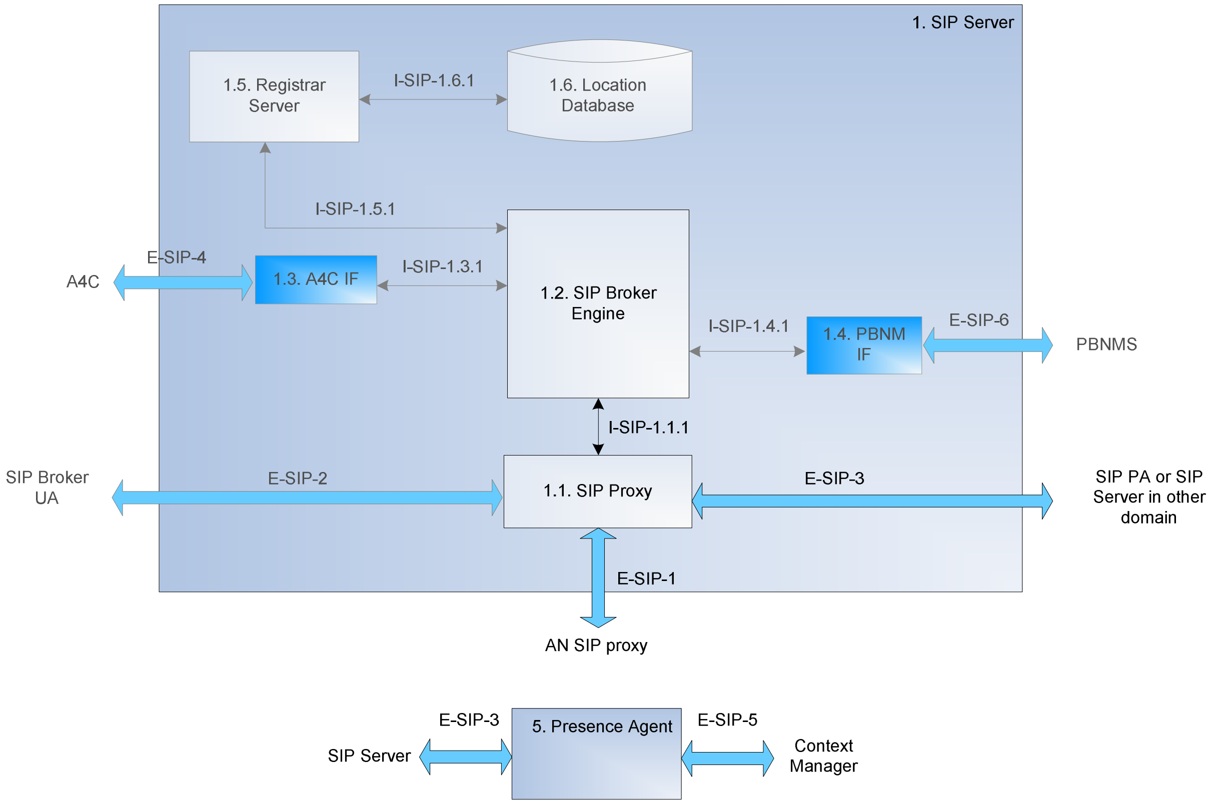


Figure 5 The Akogrimo SIP Server and the external SIP Presence Agent

2.1.3. Interfaces

In general, there are no significant changes from the description provided in D4.2.2; however, some considerations should be taken into account. Next table summarises them:

Table 2 SIP PA interfaces (embedded on the SIP Server)

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	SIP watcher in Context	SIP presence/context subscriptions (SIP watcher interface, SUBSCRIBE and NOTIFY methods)	SIP

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
WP4.2 [E-SIP-5]	Manager 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 PUA in another component 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

The interface I-SIP-1.6.2 (a C API offered by the Presence/Context Database to the Presence Agent library) for storage and retrieval of persistent publications and subscriptions has not been implemented yet, accordingly to the fact that this is an optional requirement.

The implementation of the watcher interface of E-SIP-5 (subscriptions and notifications of presence status to end-user terminals, or another entity containing a SIP User Agent (UA), like the SIP Broker) will have a strong dependency with the adopted decision regarding authorisation policies. It is necessary to adopt some mechanism in order to restrict which subscriber can access to the presence information of some presentity. Several alternatives were evaluated (e.g. the implementation of RFC 3857 [Ros04-2], "A watcher information Event Template-Package for the Session Initiation protocol" to let the user to adopt a final decision on the subscriptions. In this way, the PA could "learn" dynamically the allowed watcher list of a user; or an extension to the presence document, by adding a list of allowed users). Finally, we adopted a solution based on the maintenance of allowed watcher lists preloaded in the server, in a user basis, since it is the easiest to implement that fits all the Akogrimo project requirements.

If we consider having a separate PA from the SIP Server, the list of interfaces are reduced to a couple of standard SIP interfaces.

Table 3 SIP PA interfaces (external to the SIP Server)

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	SIP Server	SIP presence document forwarding (PUBLISH) *-Input: UserID, Presence Event State*	SIP

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol
[E-SIP-3]		*-Return: OK/Error* SIP presence/context subscriptions forwarding (SUBSCRIBE and NOTIFY methods) *-Input: UserID (AoR, default SIP URI)* *-Return: Presence Event State*	
PA WP4.2 [E-SIP-5]	PUA in MT PUA in another component 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

2.1.4. Implementation

As mentioned before, the current version of the Akogrimo PA has been built as a SER module named esc.so. ESC is the acronym of Event State Compositor, which is a generalisation of the Presence Agent for any kind of event as defined in RFC 3903 [Nie04].

SER 0.8.14 already included a version of a Presence Agent (named pa.so), but it is incomplete and experimental (e.g with no support to SIP PUBLISH). For that reason, a new a complete library was developed from scratch.

The library exports some new functionality to be invoked from the ser.cfg configuration file. Next table summarises exported functions.

Table 4 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.

A detailed description of the Presence Agent implementation, as well as the detailed description of the processing of the incoming SIP PUBLISH and SUBSCRIBE requests can be found in section 3.4.1 of D422 [D4.2.2]. No relevant changes have been introduced.

The first significant add-on from the description provided at D.4.2.2 is the extension of the SER command-line utility (serctl command) to check the status of the publications and subscriptions. It is a command-line utility which can perform the management tasks needed to operate SER: watching server status, show in-RAM users. We have extended it in order to monitor the current status of the presence documents and subscriptions.

Most significant changes are related to the several configurable parameters the library offers. Considering the new additions, the current configurable parameters are:

- The detail level of the logs (for debugging purposes a traces mechanism – independent from the SER traces framework – has been included. This allows the library debugging without "interferences" from the rest of the SER code).
- The periodic timeout the library checks the validity of both publications and subscriptions.
- The maximum permissible value for subscription validity.
- The maximum permissible value for publication validity.
- A filename containing the "default" presence document (to be sent when no information from the presentity were available).
- The mechanism to validate an incoming presence document (no validation at all, only xml validity checking, validation using a DTD [XML] or validation using an xmlSchema [xmlSchema]). Related to the last two options, a file containing the DTD or xmlSchema document for validation use can be also specified. Note that that a DTD makes it possible to validate the structure of relatively simple XML documents, but that's as far as it goes. It can't restrict the content of elements, and it can't specify complex relationships or provide support for future extensions on the xml definition. For these reasons, xmlSchema is the preferred option.
- A filename containing the presence authorisation policies (who can access to the presence information of certain URIs, in a per-URI basis; entities with full access can be also especified).

The alternative version of the SIP Presence Agent being evaluated basically consists of a Java servlet which implements the required functionality. This servlet have been deployed in a services container (or Application Server, following the IMS terminology) based on a modified Tomcat that comprises a JAIN SIP-based SIP UA, and is fully compliant with the SIP Servlet API specification [JSR116]. This alternative implementation was evaluated because the usage of the Application Server is being considered as the basis for the SIP Service Discovery framework and most of the code components could be re-used (both SIP Presence and SIP Service Discovery are based on the SIP Event Framework).

2.2. Presence User Agent (PUA)

Akogrimo's SIP Presence User Agent is a key component of the SIP Presence infrastructure. This infrastructure is in charge of collecting and delivering the presence and context information needed in several parts of the Akogrimo platform, i.e. the Context Manager or the SIP Broker.

The SIP PUA allows applications to publish desired presence and context information about the user in the Akogrimo SIP infrastructure, as well as to subscribe to other users' information. Thus, the SIP PUA is built from the combination of a publication agent and a subscription agent.

The following sections describe the SIP PUA's requirements and design, as well as its interfaces with other components and implementation details.

2.2.1. Requirements

The following tables show the progress of each of the functional and non functional requirements presented in D4.2.2. Requirements can be divided into two main blocks, F1 and F2, which are related with **publication** and **subscription** capabilities, respectively.

Table 5 SIP PUA – Functional requirements

Req. nr	Description	Status
F 1	**Presence/Context document handling**	
F 1.1	PUBLISH method support	Supported.
F 1.2	Publication refreshment	Supported.
F 1.3	Publication update	Supported.
F 1.4	Unpublication (publication removal)	Supported.
F 1.5	PIDF format support	Partially supported.
F 1.6	RPID format support	Partially supported.
F 1.7	Event State Composition (aggregation from different sources)	Deprecated, since it was not used in the use cases.
F 1.8	Publication expiration handling	Supported.
F 2	**Subscriptions handling**	
F 2.1	SUBSCRIBE method support (sender)	Supported.
F 2.2	Subscription refreshment	Supported.
F 2.3	Subscription update	Supported.
F 2.4	Unsubscription (subscription removal)	Supported.
F 2.5	Subscription expiration handling	Supported.
F 2.6	NOTIFY method support (receiver)	Supported.

As shown in the table, basic publication functionality is fully supported. F1.2 and F1.3 are implemented as an interface to allow publication of information from the different applications residing in the machine that make use of the SIP PUA to expose presence and/or context information. Since PIDF and RPID define a huge amount of attributes and parameters, this interface has been analyzed and designed carefully to come up with a solution characterized by the compromise between flexibility and maintainability. F1.4 is related to the capability of several applications to publish presence data in such a way that the complete information for a user is correctly composed from these partial descriptions.

The functionality needed for subscription to users' presence information has also been implemented and tested. Furthermore, some enhancements have been performed to allow transparent management of multiple simultaneous subscriptions. Even it has not already been used in Akogrimo testbeds, it is also possible for a user to subscribe to information for another user. Only the SIP Broker uses this functionality.

Table 6 SIP PUA – Non-functional requirements

Req. nr	Description	Status
N 1	Running on Linux (Ubuntu) on a PC platform	Done.
N 2	Running on PDA platform.	Done.

2.2.2. Design

The SIP PUA has undergone a complete redesign since D4.2.2 in response to the deep redesign performed in the underlying SIP UA. The SIP PUA has been adapted to the new interfaces and its functionality improved.

In spite of this redesign, the overall architecture remains unchanged. In this architecture, the logic was contained within a single subcomponent called "Presence/Context Logic", as depicted in the figure below.

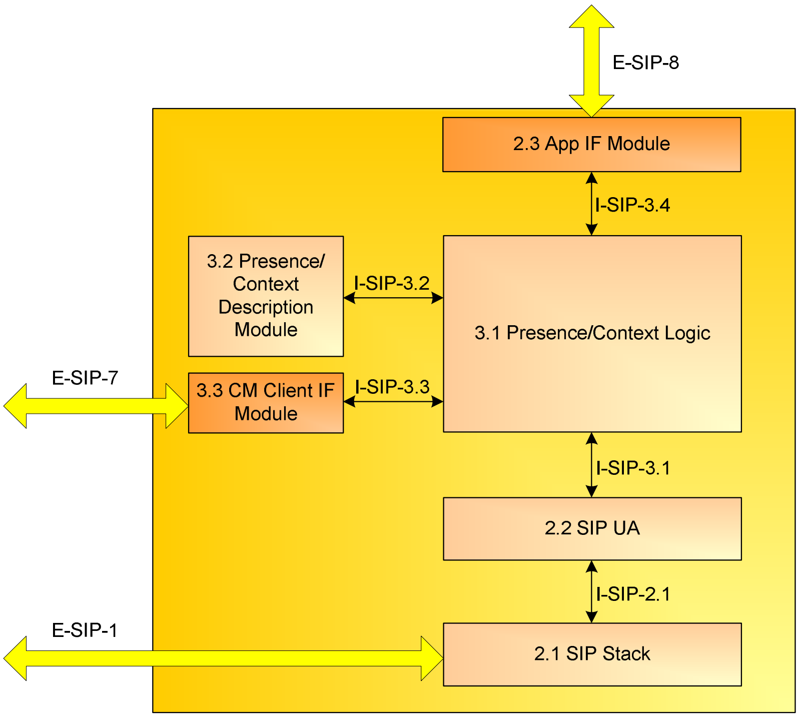


Figure 6 SIP Presence User Agent design in the Mobile Terminal

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 Context Manager (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.

As stated before, the Presence/Context Logic subcomponent holds the logic needed for presence publication and subscription. This component was implemented monolithically for phase 1, where requirements were not as demanding as they are in subsequent phases. The enhancements to SIP PUA's functionality and the redesign of the SIP UA have led to a new design perspective for this subcomponent. This approach is depicted in the next figure.

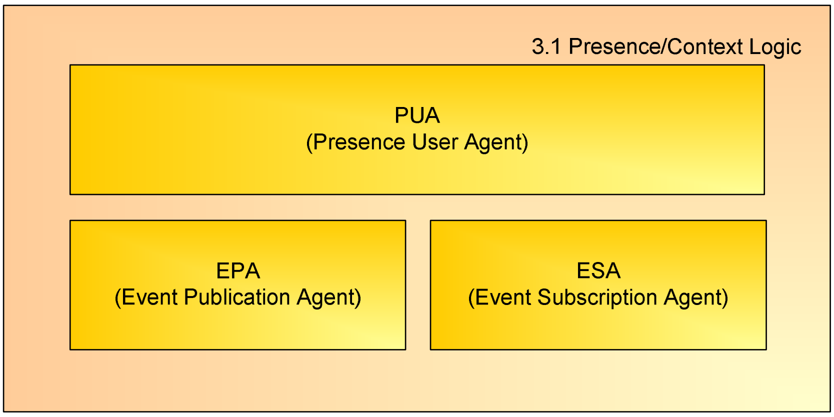

3.1 Presence/Context Logic
PUA
(Presence User Agent)
EPA
(Event Publication Agent)
ESA
(Event Subscription Agent)

Figure 7 New design for the Presence/Context Logic, SIP PUA's core

As can be seen in the figure, the new design basically consists in a modularization of the previous functionality and the addition of the entities needed to satisfy the extra requirements. This approach enhances maintainability and scalability, as functionality is reduced within each individual subcomponent.

Furthermore, the EPA and the ESA provide a generic SIP event handling architecture that eases the deployment of new event packages. In order to support new event types, it is only needed to create new classes that use the services offered by the EPA (for event state publication) and the ESA (for event subscription and notification). These new classes will only need to take care of event-specific particularities.

This architectural and implementative approach is in line with the IETF's specifications related with SIP event frameworks.

2.2.3. Interfaces

As the overall design proposed in D4.2.2 has not been updated, SIP PUA interfaces have not suffered any changes, apart from the interface with the different applications willing to publish presence data. Most of these interfaces are well defined and were implemented and tested in the first prototype. The next table summarizes these interfaces

Table 7 SIP PUA Interfaces

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

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 (PUBLISH, SUBSCRIBE and NOTIFY).	Java Interface
SIP PA (MT – WP4.2) [I-SIP-3.2]	Presence/Context Description Module (MT – WP4.2)	Conversion from applications 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)* *-Return: Operation Result*	Java Interface
SIP PUA (MT – WP4.2) [E-SIP-8]	Applications (MT – WP5.2)	Provide the PUA with user presence information. *-Input: Attributes* *-Return: Operation Result*	Java Interface

Most of these interfaces have already been exhaustively tested in the various demonstrations that have been performed.

About the E-SIP-8 interface, in first and second phases, the SIP PUA was able to send stored presence documents, but was not able to create new ones from data provided by the applications and the CM Client. From now on, however, the SIP PUA is offering an appropriate interface allowing applications to provide the needed information in the form of attributes defined in PIDF/RPID recommendations. The PUA will then build PIDF/RPID XML documents from the information provided via this interface and send them to the SIP PA in the SIP Server.

2.2.4. Implementation

The SIP Presence User Agent has been implemented as a reusable component in the form of a Java library. The functionality provided to applications using this library can be divided into two main branches: publication and subscription. These two faces are implemented via separated subcomponents to enhance maintainability and scalability. This section will describe its subcomponents in more detail.

Publication functionality is implemented by the Event Publication Agent or EPA for short. This subcomponent handles the status of the publication of the presentity corresponding to the user of the terminal the EPA is running in. This publication status handling involves creating, maintaining, updating and removing the presentity's publication in the SIP Presence Agent running at the SIP Server. Of course, publication creation, update and removal must be done "manually" by the user of the subcomponent, but the user can choose to let the EPA refresh the publication automatically. This automatic, built-in publication refreshment mechanism avoids the need for the user to take care about the periodic messages that must be sent to the SIP PA to let it know that the publication is still valid.

To actually act upon this publication status, the EPA interchanges the needed SIP signalling with the SIP PA. The messages interchanged are defined in RFC 3903 [Nie04]. The EPA offers an event-based interface in order to inform the user about the different results of the operations requested.

The second subcomponent, the Event Subscription Agent or ESA, is in charge of the functionality needed to support subscription to presence information. This functionality is used by the SIP Broker to get information about the users running Web Services. It could also be potentially used by a user to get presence information about any other user.

Subscription to presentities' data involves the handling of the subscription itself, which is carried out in a fashion similar to publication handling, but also the management of incoming notifications from the SIP PA. The ESA offers method to allow subscription creation, refreshment, update and removal. As the EPA does, the ESA also implements automatic management of subscription expiration and refreshment.

The operation of the ESA requires the use of special SIP messages, defined as an extension to the ones specified in the core recommendation. These messages are SUBSCRIBE, for subscription management, and NOTIFY, for notification handling. SUBSCRIBE and NOTIFY are the basis of the SIP-Specific Event Notification Framework, defined in RFC 3265 [Roa02].

Taking advantage of the new SIP User Agent's support for multiple, simultaneous dialogs, the ESA is able to handle several subscriptions at the same time. This is done transparently for the user, who receives an identifier of the subscription when its creation is requested. This identifier will be issued in future requests to indicate which subscription they are being asked for. In order to inform the user about the result of the requested operations, the ESA offers an event-based interface. This interface is also used to deliver to the user the notifications coming from the SIP PA.

The SIP PUA is laid on top of the SIP UA, which manages the needed SIP signalling exchanges. For more information about the SIP UA, please refer to Akogrimo's deliverable D4.1.3 [D4.1.3].

As explained, Akogrimo's SIP Presence Framework is based in SIP's extensions for Event Publication and Notification. However, events in these extensions are defined from a general point of view and are not constrained to presence. With this idea in mind, a similar approach has been used to implement service discovery capabilities using the existing SIP infrastructure. This is known as SIP SD, and it is being analyzed within Akogrimo. The SIP SD infrastructure is very similar to the described infrastructure for presence, with the type of information exchanged being the main difference. These similarities have been taken into account in the design of the EPA and the ESA, so that both of them can be reused in the future SIP SD deployment. A single parameter provided at construction time is needed to use EPA and ESA either for presence or for service discovery.

3. 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. The A4C Server tightly interacts with the SAML Authority to authenticate users and to create IDTokens, which are cryptographic values, identifying the user's authentication assertion. Therefore, the SAML Authority is also described in this section.

3.1. Requirements

Table 8 provides an overview of the key requirements of the A4C component and the current implementation status.

Table 8 A4C - Functional requirements and implementation status

Req. nr	Description	Implementation Status
F 1	**Authentication**	
F 1.1	Authentication of users based on PANA (username/password) for network access.	Implemented
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.	Implemented
F 1.3	Single-sign-on support based on SAML (IDToken).	Implemented
F 1.4	IDToken management, generation and verification.	Implemented
F 1.5	Additional authentication mechanisms.	Not implemented (optional)
F 1.6	Private Authentication generation	Not implemented (optional)
F 2	**Authorization**	
F 2.1	Authorization of network services (i.e. type of access, QoS)	Implemented
F 2.2	Provide QoS Profile to the QoS Broker	Implemented
F 2.3	Provide the generic user profile	Implemented
F 2.4	Provide anonymous/pseudonymous access to different services	Pseudonymous access implemented
F 2.5	Authorize User or Service Provider (SP)	Implemented (used by Context Manager)

Req. nr	Description	Implementation Status
F 3	**Accounting**	
F 3.1	Accounting for network services usage (e.g. volume, QoS)	Implemented
F 3.2	Support of accounting during handover and roaming	Implemented
F 3.3	Accounting for grid services usage, support of grid accounting parameters	Implemented
F 3.4	Accounting for compound services, support of service hierarchy	Implemented
F 3.5	Multi-domain support, correlation of accounting records from different domains related to the same service	Implemented
F 4	**Auditing**	
F 4.1	Manage, record and store auditing events (e.g. SLA violation events received from the EMS)	Implemented
F 4.2	Non-repudiation of service usage	Obsolete (optional)
F 4.3	Auditing of A4C messages exchanged between domains	Obsolete (not a task of A4C)
F 5	**Charging**	
F 5.1	Charge calculation for service consumption based on accounting records and tariff profiles	Implemented
F 5.2	Support of service-specific and user-specific tariff profiles	Implemented
F 5.3	Charging for compound services, support of service hierarchy	Implemented
F 5.4	Multi-domain support for the charge calculation	Implemented
F 5.5	Charging record exchange between domains	Not implemented (the alternative, i.e., the exchange of accounting records, is implemented)
F 5.6	Single bill support, create bills for users	Implemented

Req. nr	Description	Implementation Status
F 6	**Generic**	
F 6.1	Manage and store user profiles	Implemented
F 6.2	Manage and store tariff profiles	Implemented
F 6.3	Manage and store accounting records	Implemented
F 6.4	Manage and store charging records	Implemented
F 6.5	Session support, binding A4C tasks together	Implemented
F 6.6	Secure communication (IPSec, TLS), protect inter-domain communication	Provided by OpenDiameter

Table 9 provides the same information for the SAML Authority.

Table 9 SAML Authority – Functional requirements and implementation status

Req. nr	Description	Implementation Status
F 1	**IDToken Management**	
F 1.1	IDToken generation (e.g. user@akogrimo.org)	Implemented
F 1.2	Verification of the IDToken and provision of the SAML authentication assertion	Implemented
F 1.3	SAML attribute assertion generation	Implemented
F 1.4	Signature and encryption of IDToken	Implemented
F 1.5	Signature and encryption of SAML Assertions	Implemented
F 1.6	Provision of a token with delegation support	Not implemented (optional)

3.2. A4C Overview and Interfaces

Figure 8 provides an overview of the 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]. The A4C Client Library provides an API either in C++ or in Java. There is an optional interface between A4C Server and SLA Repository, which enables the retrieval of charging information if it was negotiated dynamically. The A4C Server also interacts with other A4C Servers in support of authentication, authorization, accounting, auditing, and charging.

Akogrimo uses the Security Assertion Markup Language (SAML) [SAML] to provide the IDToken. The SAML Authority assists the A4C Server in the authentication process and provides IDToken management and verification. The SAML Authority interfaces directly with the A4C Server in the form of request and response messages.

For more details on the deployment of the A4C infrastructure refer to [D4.2.2].

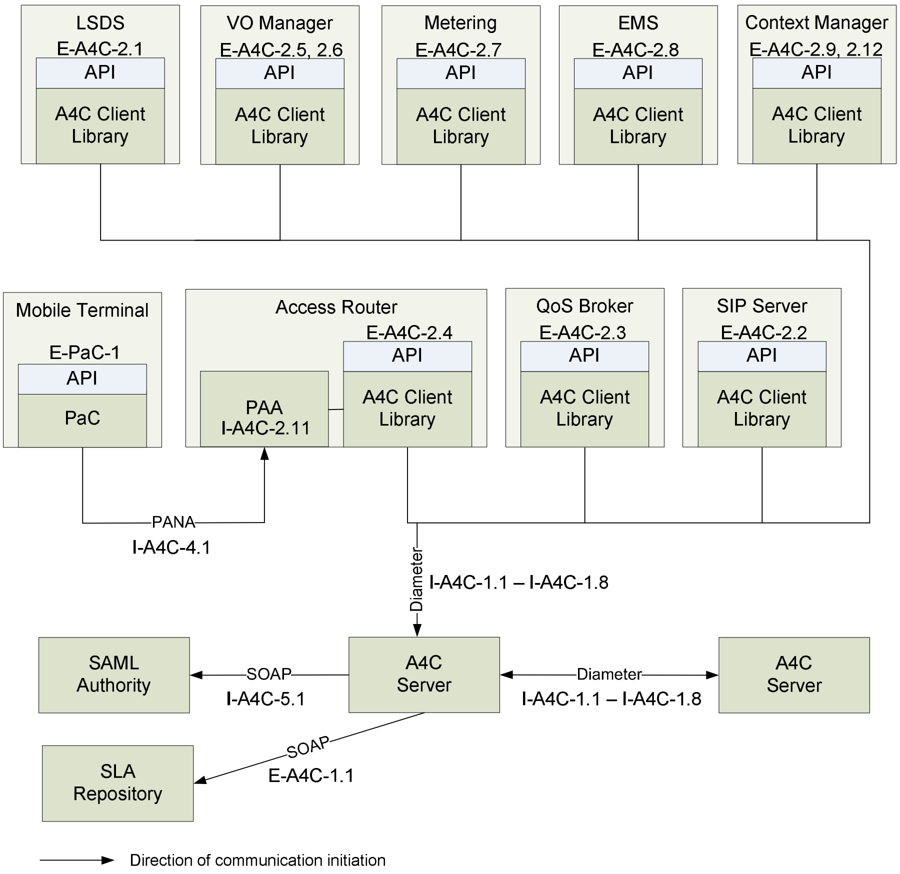


Figure 8 Overview of A4C interfaces

Table 10 provides a complete list of all interfaces of the A4C infrastructure. The table also includes references to the description of the interfaces. Interfaces, that have not changed, are described in detail in [D4.2.2]. New interfaces and updates in the interface specification are described in the subsequent sections.

For the implementation architecture of the A4C Client and Server refer to [D4.2.2].

Table 10 A4C Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Interface/Protocol	Reference/Status
A4C Server (WP4.2) [I-A4C-1.1] – [I-A4C-1.8]	A4C Client (WP4.2)	Provide A4C tasks	Diameter	D4.2.2/unchanged
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	D4.2.2/unchanged

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Interface/Protocol	Reference/Status
A4C Server (WP4.2) [I-A4C-1.1]	A4C Client (WP4.2)	User authentication and transmission of IDToken, public userIDs, and VO parameters. *-Input: loginName, password* *-Return: OK/Error. In case of OK, IDToken, public userIDs, and VO parameters are transferred.*	EAP/Diameter	Section 3.4/new
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	D4.2.2/obsolete (updated by the A4C client PAA interface)
A4C Client (WP4.2) [I-A4C-2.11]	PAA (WP4.2)	User authentication and transmission of IDToken, public userIDs, and VO parameters. *-Input: loginName, password* *-Return: OK/Error. In case of OK, IDToken, public userIDs, and VO parameters are transferred.*	C++ API	Section 3.4/ updated
PAA (WP4.2) [I-A4C-4.1]	PaC (WP4.2)	User authentication and transmission of IDToken, public userIDs, and VO parameters. *-Input: loginName, password* *-Return: OK/Error. In case of OK, IDToken, public userIDs, and VO parameters are transferred.*	PANA	Section 3.4/new
PaC (WP4.2) [E-PaC-1]	MT (WP4.1)	User authentication and transmission of IDToken, public userIDs, and VO parameters. *-Input: loginName, password* *-Return: OK/Error. In case of OK, IDToken, public userIDs, and VO parameters are transferred.*	Java API	Section 3.4/new
A4C Client (WP4.2) [E-A4C-2.4]	Access Router (WP4.1)	Exchange of user authentication related data, i.e., notification of the AR about user login, logout, access termination, authentication result, and the AR can request access termination.	C++ API	Section 3.4/new
A4C Client (WP4.2) [E-A4C-2.4]	Access Router (WP4.1)	Accounting: *-Input: UserID, sessionID, serviceID, accounting data* *-Return: OK/Error*	C++ API	D4.2.2/unchanged
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	D4.2.2/unchanged
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	D4.2.2/unchanged
A4C Client (WP4.2) [E-A4C-2.9]	Context Manager (WP4.2)	Retrieve generic user profile: *-Input: UserID* *-Return: generic user profile*	Java API	D4.2.2/unchanged

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Interface/Protocol	Reference/Status
A4C Client (WP4.2) [E-A4C-2.12]	Context Manager (WP4.2)	Authorize access to context information *-Input: serviceRequestorID, objected* *-Return: true/false*	Java API	Section 3.4/new
A4C Client (WP4.2) [E-A4C-2.6]	VO Manager (WP4.4)	Authentication based on IDToken: *-Input: UserID, IDToken, serviceID* *-Return: OK/Error, new IDToken*	Java API	D4.2.2/unchanged
A4C Client (WP4.2) [E-A4C-2.5]	VO Manager (WP4.4)	Retrieve service profiles. *-Input: userID, serviceID* *-Return: serviceProfile/Error.* Publish service profiles. *-Input: userID, serviceID, serviceProfile* *-Return: OK/Error.*	C++/Java API	Section 3.4/ updated
A4C Client (WP4.2) [E-A4C-2.7]	Metering (WP4.3)	Accounting: *-Input: UserID, sessionID, serviceID, accounting data* *-Return: OK/Error*	Java API	D4.2.2/unchanged
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	D4.2.2/obsolete (interface between A4C client and EMS is used instead)
A4C Client (WP4.2) [E-A4C-2.8]	EMS (WP4.3)	Auditing of service events *-Input: sessionID, eeventID, eventData, (SLAContractID)* *-Return: OK/Error*	Java API	Section 3.6/new
SLA Repository (WP4.4) [E-A4C-1.1]	A4C Server (WP4.2)	Retrieve a SLA contract for getting the charging specification. *-Input: SLAContractID* *-Return: SLA-Contract/Error*	SOAP	Section 3.5/new
A4C Client (WP4.2) [E-A4C-2.x]	A4C SOAP Gateway	Provide a SOAP based interface for A4C tasks.	Java API	D4.2.2/obsolete
A4C SOAP Gateway	Akogrimo Component (WP4.3, WP4.4)	Access A4C services.	SOAP	D4.2.2/obsolete
A4C Client (WP4.2) [E-A4C-2.1]	LSDS (WP4.2)	Authorize User or SP	C++ API	D4.2.2/unchanged

3.3. SAML Authority Overview and Interfaces

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. Figure 9 shows the integration of the SAML Authority in the A4C system architecture.

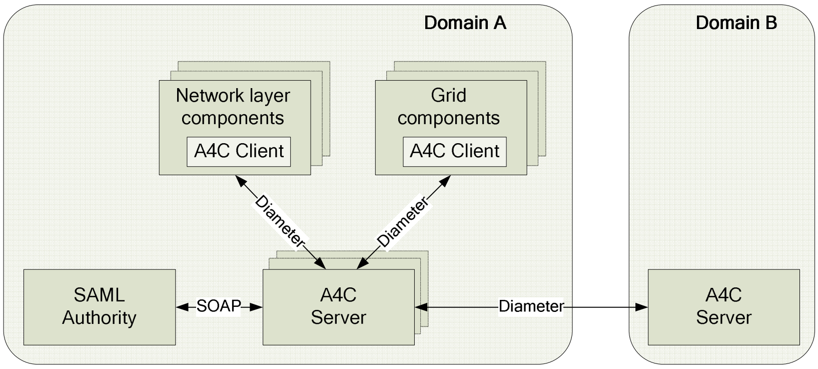


Figure 9 SAML Authority in the context of A4C Architecture

More information related to the SAML Authority can be found in chapter 4 ("Identity Management").

Table 11 SAML Authority Interfaces

Server: Component (Layer) [Interface Label]	Client: Component (Layer)	Purpose	Protocol	Reference
SAML Authority (WP4.2) [I-A4C-5.1]	A4C Server (WP4.2)	Generate IDToken. *-Input: userID* *-Return: IDToken/Error* Validate IDToken. *-Input: IDToken* *-Return: OK/Error* Generate SAML Attribute Assertion (not used) *-Input: Attributes, serviceID, userID* *-Return: AttributeAssertion/Error*	SOAP	D4.2.2

3.4. Authentication and Authorization

Authentication and authorization within the A4C include different aspects. Network authentication is based on PANA and EAP, and it is described in Section 3.4.1. The A4C Server is also responsible for storing user and service profiles, which is described in Section 3.4.2. Additionally, the A4C Server provides an interface to the Context Manager in order to authorize context request. This is described in Section 3.4.3.

3.4.1. Network Access Authentication

Network authentication is an important function within the A4C architecture. It supports the authentication of users for network login based on username and password. Network authentication is an important task the user has to go through prior to any usage of any

Akogrimo service. The PANA Client (PaC) in the Mobile Terminal (MT) supports the authentication of the user against the A4C server of the user's home domain. The Access Router (AR) acts as a PANA Authentication Agent (PAA) towards the MT and uses the authentication service of the A4C Server to receive an authentication decision. After successful authentication the user gains network access via the AR and the user is also assigned an IDToken that is used for authentication against other Akogrimo services in the AkogrimoVOs later on. The IDToken is based on SAML [SAML] and issued by the SAML Authority of the *home A4C Server*. Besides the IDToken, the user also receives information about his public userIDs and the VO parameters of subscribed services.

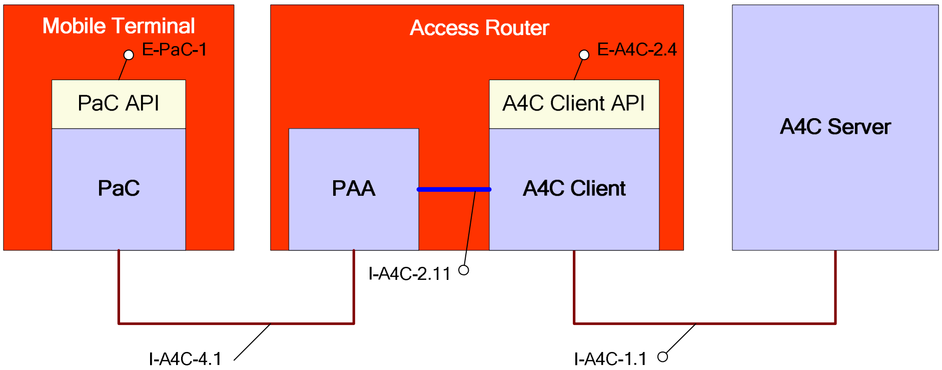


Figure 10 Components and Interfaces for Network Authentication

Figure 10 shows the network components participating in the network authentication process and the interfaces between them. Authentication is based on the Extensible Authentication Protocol (EAP) in which authentication messages are encapsulated. The A4C architecture uses MD5 based authentication with username and password for user authentication. However, EAP supports various authentication methods that can be integrated in the A4C architecture. The communication between the MT and the AR is established via the Protocol for carrying Authentication for Network Access (PANA) [For05], while between AR and A4C Server the Diameter protocol is used. EAP messages are transferred between MT and A4C Server transparently for the AR. The AR forwards EAP messages transmitted between MT and A4C Server in both directions, i.e. the AR takes EAP messages received in a PANA message from the PaC and forwards it to the A4C Server in a Diameter message; and it takes EAP messages received in a Diameter message from the A4C Server and forwards it in a PANA message to the PaC. This way the communication between the three entities continues as part of the authentication method on the top of EAP. Once the A4C Server decides on the authentication of the user, it issues a Success/Failure message. On the basis of this, the user is either given access to the network or his request is rejected.

The MT includes the PaC library that provides all authentication related functions via the PaC API over the E-PaC-1 interface. Over this interface the MT can initiate the user authentication and after successful authentication it receives the IDToken, the public userIDs, and VO parameters, which are transferred in EAP-Notification messages to the MT. The public userIDs are displayed by the MT and the user can select the one that he wants to use for the subsequent service access (cf. Section 3.4.2 for further details). The VO parameters include the service and VO name the user is subscribed to, the User Agent (UA) URI where the user can access the service, and an optional text based description (cf. Section 3.4.2 for further details).

The AR includes the PAA and the A4C Client libraries that assist the user to authenticate and provides a communication interface to the AR over the E-A4C-2.4 interface. Between the PAA and A4C Client there is an internal interface (I-A4C-2.11) that is not visible to the AR. The A4C Client API in the AR enables the AR to request the termination of a current network access session and it is also used to notify the AR about the authentication process, user login, user logout, and access termination.

The I-A4C-4.1 interface provides the PANA based communication between MT and AR. The I-A4C-1.1 interface enables the communication between AR and A4C Server via the Diameter protocol. Both interfaces are A4C internal.

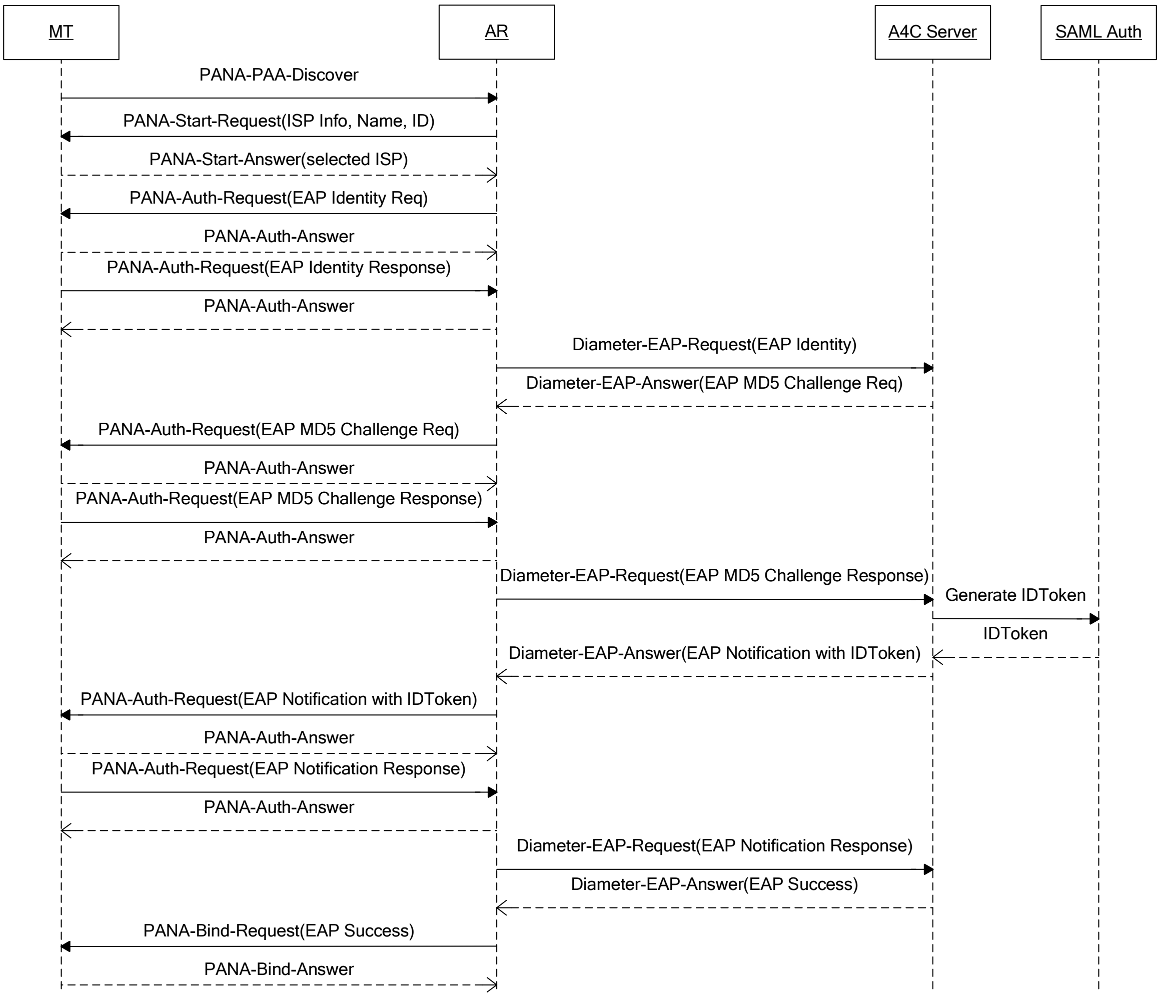

MT
AR
A4C Server
SAML Auth
PANA-PAA-Discover
PANA-Start-Request(ISP Info, Name, ID)
PANA-Start-Answer(selected ISP)
PANA-Auth-Request(EAP Identity Req)
PANA-Auth-Answer
PANA-Auth-Request(EAP Identity Response)
PANA-Auth-Answer
Diameter-EAP-Request(EAP Identity)
Diameter-EAP-Answer(EAP MD5 Challenge Req)
PANA-Auth-Request(EAP MD5 Challenge Req)
PANA-Auth-Answer
PANA-Auth-Request(EAP MD5 Challenge Response)
PANA-Auth-Answer
Diameter-EAP-Request(EAP MD5 Challenge Response)
Generate IDToken
IDToken
Diameter-EAP-Answer(EAP Notification with IDToken)
PANA-Auth-Request(EAP Notification with IDToken)
PANA-Auth-Answer
PANA-Auth-Request(EAP Notification Response)
PANA-Auth-Answer
Diameter-EAP-Request(EAP Notification Response)
Diameter-EAP-Answer(EAP Success)
PANA-Bind-Request(EAP Success)
PANA-Bind-Answer

Figure 11 Message Sequence for Network Authentication

The message sequence for successful user authentication based on MD5 is shown in Figure 11. The mobile terminal acts as a PaC that can communicate with the PAA component in the AR via the PANA protocol. The A4C Client component in the AR communicates with the A4C server via the Diameter protocol. The communication starts with the PANA-PAA-Discover message sent by the MT to find available ARs. The AR sends a PANA-Start-Request containing information about the network provider, such as its name and ID. In turn, the MT selects a network provider from the list and sends its selection back to the AR in the PANA-Start-Answer. These messages are part of the handshake phase.

After the handshake, the AR issues a PANA-Auth-Request to the MT, containing an EAP message requesting the identity of the user. The PANA-Auth-Answer is sent back as an acknowledgement and afterwards the MT sends the user's identity in the PANA-Auth-Request, which is acknowledged by the AR. The AR forwards the user's identity to the A4C server in the Diameter-EAP-Request message. The A4C server replies with a Diameter-EAP-Answer, containing an EAP MD5 challenge. The challenge is forwarded to the MT in the PANA-Auth-

Request. The MT replies to the challenge in the PANA-Auth-Request containing an EAP MD5 Challenge Response. The AR forwards the EAP message with the answer to the challenge to the A4C server, which in turn can verify the identity of the user.

If the authentication is successful, the A4C server requests the generation of an IDToken from the SAML Authority. Then, the A4C Server sends the IDToken, the user's public identities, and VO parameters to the AR in an EAP Notification that is encapsulated in the Diameter-EAP-Answer. The EAP Notification is forwarded to the MT in the PANA-Auth-Request message. After the EAP Notification is acknowledged, the A4C server sends the EAP Success to the AR, which is forwarded to the MT in the PANA-Bind-Request. The MT replies with a PANA-Bind-Answer, which closes the authentication process.

In order to support network access authentication, the A4C infrastructure provides several interfaces that are described in the following sections.

3.4.1.1. A4C Internal Interfaces (I-A4C-1.1, I-A4C-2.11, I-A4C-4.1)

The A4C Internal Interfaces I-A4C-1.1, I-A4C-2.11, and I-A4C-4.1 provide the transfer of authentication related data between A4C Client and A4C Server via Diameter, between A4C Client and PAA, and between PaC and PAA via PANA, respectively. These interfaces enable the transmission of username and password to the A4C Server and the retrieval of IDToken, public userIDs, and VO parameters. Refer also to [D.4.2.2].

3.4.1.2. Authentication API for the MT (E-PaC-1)

The authentication interface provided by the PaC library is used within the MT for user authentication during the network access request. The authentication API provides the following class and methods:

- ***class Authenticator (loginName, password)***

  The ***Authenticator*** class provides the authentication functions for the MT. The class is initiated by the user's login name and his password. When it is instantiated, it automatically makes one authentication request.

  Methods of the Authenticator class:

  - ***void setUsername(char* username)***

    Changes the username used for authentication.

  - ***void setPassword(char* password)***

    Changes the password used for authentication.

  - ***int getResultCode()***

    Returns the authentication result: OK/Err.

  - ***void reAuthenticate()***

    Makes an authentication request.

  - ***char* getToken()***

    Returns the IDToken received from the A4C Server.

  - ***int getNumberIDs()***

    Returns the number of public IDs received for the authenticated user.

- ***char* getPublicID(int i)***

  Returns the i-th public ID.

- ***char* getServiceID(int i)***

  Returns the VO associated with the i-th public ID.

- ***char* getServiceAccess(int i)***

  Returns the user agent URI from the VO attached to the i-th public ID.

3.4.1.3. Authentication API for the AR (E-A4C-2.4)

The A4C Client API in the AR enables the communication between AR and A4C for authentication purposes. The AR can request the A4C Client to terminate the network connection, e.g., for administrative reasons. Via the interface the A4C Client can notify the AR about the outcome of the authentication process (i.e. success or failure) and it can also inform the AR if the user logs out or the network access connection is terminated due to other reasons (e.g., loss of wireless signal).

3.4.2. Authentication based on the IDToken

The A4C Server supports authentication based on the SAML IDToken. An Akogrimo Component, e.g., the BaseVO Manager, 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 an AA-Answer (AAA) to the Akogrimo Component.

The message sequence chart of the IDToken based authentication is shown in Figure 12.

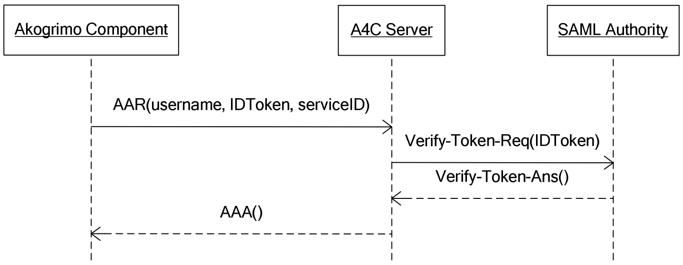


Figure 12 Message Sequence for IDToken based Authentication

3.4.2.1. IDToken based Authentication API (E-A4C-2.2/2.6)

The IDToken based authentication API provides the interface to perform authentication based on the SAML IDToken. The authentication API provides the following class and methods:

- ***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.

3.4.3. Identity and Profile Management

One of the most important tasks of the A4C infrastructure is to verify a user's identity before granting him access to services. Whilst knowing the identity of a user is the main requirement for service accounting and charging, for privacy purposes users often prefer to use different digital identities when accessing different services. The A4C infrastructure supports the multi-domain provisioning of services by offering a federated identity mechanism that allows Single Sign-On (SSO) and provides support for virtual user identities. A virtual user identity is created whenever a user wants to use a service of a service provider without revealing his real identity to that provider. The main requirement for providing SSO and virtual identity support is to have trust agreements between different administrative domains.

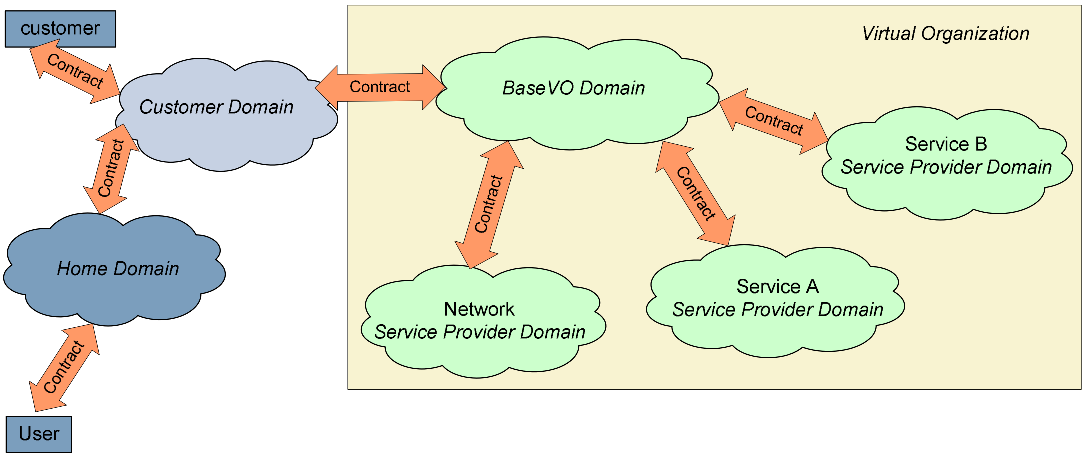


Figure 13 Akogrimo example of trust establishment

As the A4C Servers are the components that verify user identities, trust relationships between domains can be created by having trust relationships between the A4C Servers of the respective domains. A trust relationship between two A4C Servers implies that messages, authentication decisions, accounting, auditing and charging records are accepted from the trusted partner A4C server.

Federated identity management is supported by integrating SAML within a Diameter-based A4C infrastructure. Besides the SSO support, SAML also provides support for secure retrieval of identity attributes. More details on federation identity can be found in Section 4.

The main role of the A4C Server in the Identity and Profile management process is the provisioning of user identities and profiles. All the identities of a user are stored in the A4C Server in the home domain of the user. After a successful authentication the A4C Server sends all the public identities of the users together with a list of VOs in which the user is registered to the mobile terminal. Based on this information, the user can choose the identity he wants to use.

Services to which users are subscribed may require the storage of a service profile. Such a profile would contain information specific for the given service (e.g. language in which the user prefers the service to be delivered, or the default codec to use for voice encoding). As the service profiles are service specific, it is up to the service to define what the corresponding service profile should contain. Service profiles are stored transparently in the A4C Server of the service providers who deliver those services.

```
<?xml version="1.0" encoding="UTF-8"?>
<GUP>
        <userID>alice@akogrimo.org</userID>
        <RfidTag>D2C7F651B28804004659255641102102</RfidTag>
        <SipUri>alice@akogrimo.org</SipUri>
</GUP>
```

Figure 14 Generic User Profile

Figure 14 shows an example of a Generic User Profile. Such a profile is bound to a public identity and is used to store attributes specific to that public identity. The example here contains only information required by the e-Health scenario, but it can be extended with other parameters.

An example of a service profile can be found in Annex C.1. The presented example shows a network QoS profile for user alice@akogrimo.org.

In order to store and retrieve profiles, the A4C infrastructure provides one interface that can be called using an A4C Client. This profile management interface is described in the next section.

3.4.3.1. Profile Management API (E-A4C-2.5)

The ID and Profile Management interface is described in [D4.2.2], Section 4.2.2. In addition to the previous description, two new methods are required for the interface E-A4C-2.5 that will be used by the VO to publish and query service profiles:

- ***serviceProfile serviceProfileRequest(userID, serviceID)***

  The ***serviceProfileRequest()*** method is used for retrieving the service profile of a user.

- ***void serviceProfilePublish(userID, serviceID, serviceProfile)***

  The ***serviceProfilePublish()*** method is used for publishing a service profile for a given user in the A4C Server.

3.4.4. Authorization of Context Request

Access to context information for users needs to be protected so only authorized services can have access to the context of a specific user. The interface presented here, although currently only used by the Context Manager, might be used in the future also by other components to request authorization. For that, service specific modules need to be implemented in the A4C Server in order to support service-specific authorization policies.

The authorization for access to context information will be performed by checking whether a requester identified by a unique identifier may have access to the context of an object identified as well with a unique identifier. The authentication of the two identifiers (checking whether the requester really is who he claims to be and if the context really belongs to the right object) will be performed internally by the Context Manager by using certificates. More details can be found in section 5.3.5.3.

In order to support authorization of context request, the A4C infrastructure provides one interface that can be called using an A4C Client. This interface is described in the next section.

3.4.4.1. Context Access Authorization API (E-A4C-2.12)

The Context Access Authorization interface provides means for the CM to verify with the A4C if a particular requestor of a context may have access to that context. For this purpose, the A4C Client class provides one method that can be used to verify authorization:

- ***int authorizationRequest(requesterID, objectID)***
  - *requesterID* – the serviceID of the service who wants access to context information
  - *objectID* – the userID for which context information is requested
  - the method returns 1 if authorization is successful or 0 otherwise.

3.5. Accounting and Charging

The A4C infrastructure supports accounting of service usage both for network services and grid services. Based on the collected accounting records the A4C Server performs charging for service usage. For more details on accounting and charging in A4C refer also to [D.4.2.2].

3.5.1. Multi-Domain Accounting and Charging for Compound Services

The multi-domain service provisioning aspect has an impact not only on the authentication of users, but also on the accounting and charging of compound services. As a VO groups together multiple service providers that agree to work together in order to provide high-level services to users, the accounting and charging mechanisms of these service providers need to support the accounting and charging of all the corresponding service sessions while presenting the user with a single, aggregated bill that hides all the details about the sub-services required for the service execution and delivery.

In Akogrimo a Single-Sign-On (SSO) approach was chosen for reducing the number of contracts with SPs a user has to manage. Based on the contracts between SPs, users may access services in administrative domains with which they do not have any contract. Figure 13 of the previous section above shows a scenario in which a user has a contract with his home domain. Based on the contract/trust relationships between service providers, the user may request services from the BaseVO domain. This services may further aggregate services like Service A or Service B from other SP domains. As the user only maintains a contract with his home domain, he will only accept bills from this one, thus the home domain of a user will act as a *bill aggregator* for all services the user requested.

A contract and trust relation between the BaseVO and service providers assures that whenever a user requests a service from the SP, the SP can send the service charge fee to the BaseVO and expects that the BaseVO pays for that service session.

Such an approach requires the communication of charging records and/or accounting records between administrative domains. An accounting record contains information about the degree of service usage at a given time, while a charging record contains information about the overall service session, like the total amount of resources used and the total charge of the session.

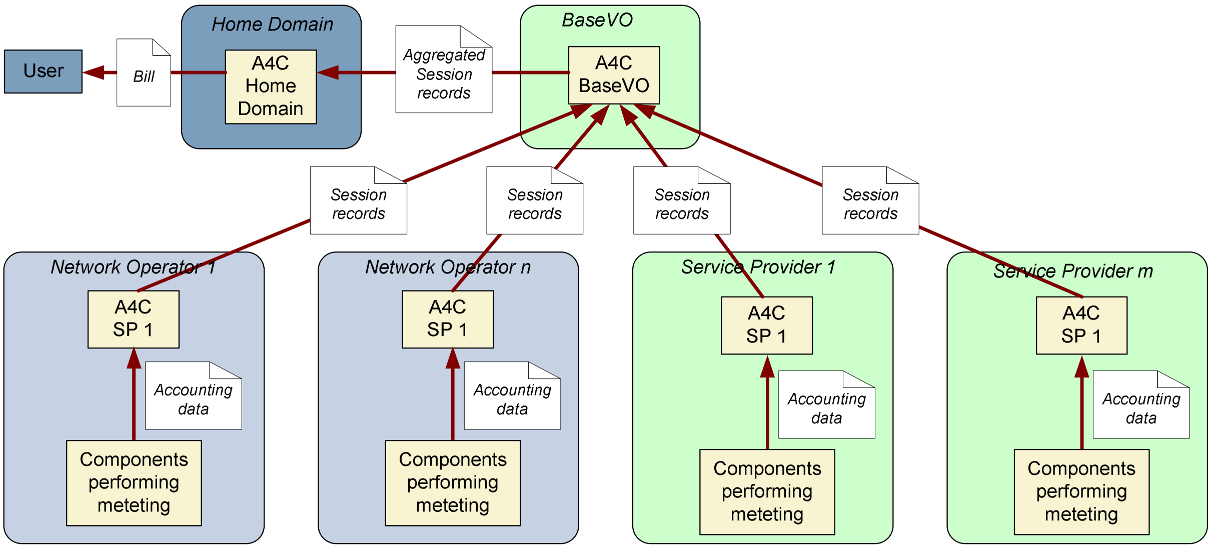


Figure 15 Accounting and charging data flow

Figure 15 shows the accounting and charging data flow in the multi-domain grid environment, where network operators and service providers offer the complete service to the user. In each network operator and service provider domain there is an A4C server that receives and collects service usage data in the form of accounting records from components participating in the service provisioning and performing service usage metering (e.g., the access router in the network operator domain). A4C servers in the participating domains transfer session records to the A4C server in the BaseVO domain, since the BaseVO domain is the central entity managing the service provisioning. Session records can include accounting records and charging records, depending on the business model and contract between the providers. The A4C server in the BaseVO domain sends aggregated session records to the A4C server in the user's home domain. These records enable the home domain to prepare a single bill, containing all charges for accessed services in the grid environment.

In case charging specifications are negotiated dynamically at service request, the session record also contains the SLA-Contract-ID that identifies the SLA established between providers. Based on the SLA-Contract-ID the A4C Server can retrieve the appropriate SLA and charging specification from the SLA Repository and can perform the charge calculation for the given session.

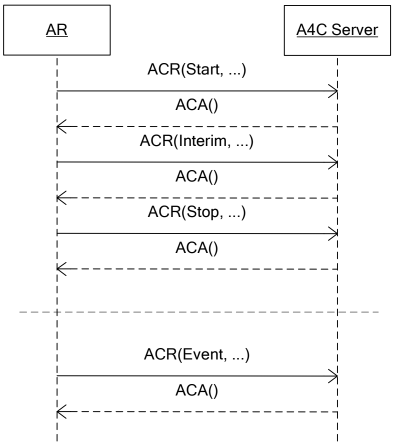


Figure 16 Message Sequence for Accounting

The messages related to accounting are shown in Figure 16. The figure only shows the AR but the same applies for other components as well. 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 kinds 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.

3.5.2. 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.

3.5.3. SLA Interface (E-A4C-1.1)

In order to support the retrieval of charging information from the SLA, the A4C infrastructure has an interface with the SLA Repository. The SLA interface will be used by the A4C Server to retrieve the charging specification in the SLA contract from the SLA Repository. The charging specification is used to calculate the charge for a given service session. In case the charging scheme is negotiated dynamically, accounting records and auditing records have to include the SLA-Contract-ID in order to enable the A4C server to identify the SLA to be applied for the session. Note, that this interface will only be needed if the appropriate charging scheme will be negotiated during the SLA negotiation process. If only the QoS parameters are negotiated, the SLA interface is not needed. In that case the A4C server will have the fixed charging scheme available locally. Charging information will be stored in the database of the A4C Server.

- ***ResultCode retrieveSLAContract(SLAContractID, &SLA)***

  The retrieveSLAContract() method is used by the A4C server to retrieve the SLA contract from the SLA Repository. The following is the list of input parameters and return values of this method:

  - *SLAContractID*: a string representing the identifier of the SLA contract generated for the service session.
  - *&SLA*: an object representing the SLA contract as a result of the method call.
  - *ResultCode*: a boolean representing whether the task was completed successfully or not.

3.6. Auditing

Auditing can generally be defined as an examination of audit trails to ensure compliance with pre-established specifications, including procedures, policies, and agreements or contracts. The auditing objective within Akogrimo is to detect any violations to an SLA and to react according to the SLA. Akogrimo defines reimbursements for affected customers if a provider is not able to fulfill the SLOs specified. With respect to SLA compliance auditing, an A4C server is responsible for collecting, processing, and storing of SLA violation reports obtained from the EMS (Execution Management Service). During charge calculation these reports will be checked and reimbursements will be applied according to the negotiated SLA for the service that was monitored. A4C auditing will not take any further actions such as controlling the service instance, as this is done by the EMS.

Figure 17 shows the sequence of messages to report auditing events to the A4C server. During service provisioning, a monitoring component receives notifications with respect to CPU, storage, and memory usage from the Metering service. In order to reduce traffic, the Metering service sends out a notification only when the difference between the new and old value of a QoS parameter exceeds a certain threshold. The Monitoring service also gets notifications related to network layer from the QoS Broker. The Monitoring service informs SLA Controller when a notification is received.

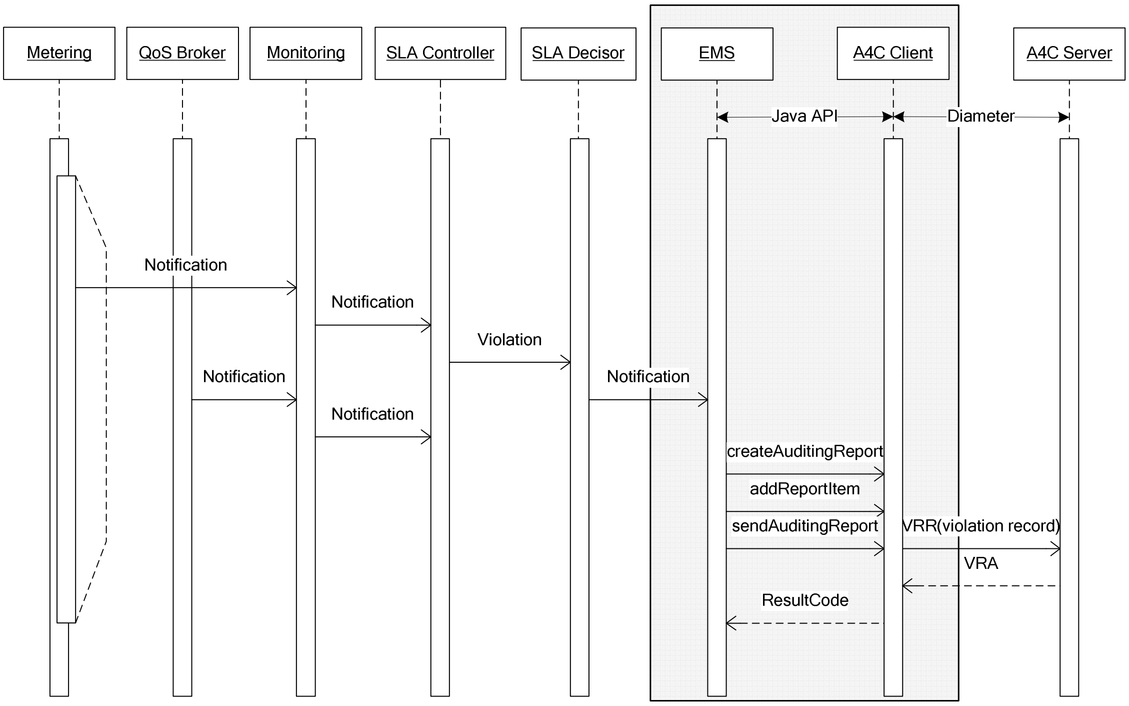


Figure 17 MSC for SLA compliance auditing interactions

The SLA Controller determines whether the service is running under the expected condition, i.e., whether the QoS parameters are within the thresholds defined in the SLA. If a violation occurs, the SLA Controller sends a violation report to the SLA Decisor which contacts the Policy Manager to take the appropriate actions. Afterwards, the SLA Decisor propagates the violation to the EMS through a notification message.

The EMS, having received the notification message, reports the event to the A4C server with the help of an A4C client. The A4C client creates a Violation-Report-Request (VRR) message and sends it to the A4C server. Included in the message are the violation type and the violation degree that was detected as well as the accounting sessionID of the session for which this violation is reported. Possible violation types for a service are stored in the A4C server. The A4C Server responds with a Violation-Report-Answer (VRA).

3.6.1. SLO Violation and Reimbursement

As mentioned in the previous section a reimbursement is applied if an SLO is violated. Each SLO is given an identifier which is called an SLO-ID to allow for later references when reporting a violation event. Table 12 lists and describes only that part of information in an SLA which is relevant for auditing.

Table 12 Reimbursement formulae

Information Item	Type	Example	Remark
Contract (SLA)-ID	String	20061116-01	Each SLA must be uniquely identified by an SLA-ID.
Service-ID	String	VoIP-Gold	An SLA may specify more than one service. Each service is identified by a Service-ID.
SLO-ID	Unsigned Integer	5 (Jitter)	An SLA may specify several SLOs for a particular service. Each SLO is identified by an SLO-ID.
SLO	Expression	Jitter_{max} < 100 ms and Jitter_{avg} < 50 ms	An SLO defines an expression representing a specific condition to be met. In general, this expression is composed of relational and logical expressions. The relational expressions relate QoS parameters with specific target values.
Violation Degree	Unsigned Integer	1 (Severe)	To allow for finer granularity of SLO violation evaluations, degrees of violation are introduced. Each degree corresponds to a particular violation condition.
Violation Condition	Expression	Jitter_{max} > 250 ms or Jitter_{avg} > 150 ms	A violation condition is defined for a particular violation degree.
Base Reimbursement	Unsigned Integer	10 EUR/event	The reimbursement to be applied if there is a violation to the SLO irrespective of the degree of violation.
Reimbursement Weight	Unsigned Integer	150	For each violation degree a weight factor in percentage of base reimbursement is also defined.

Table 13 presents a list of pre-defined SLO-IDs to be used within Akogrimo. The list is not considered to be complete and is extensible to allow for other types of violations.

Table 13 Akogrimo SLO-ID

Akogrimo SLO-ID	Meaning
1	Downlink Throughput
2	Uplink Throughput
3	End-to-end Delay
4	Round-Trip Time (RTT)
5	Jitter
6	Packet Loss
7	CPU Usage
8	Memory Usage
9	Storage Usage

3.6.2. Auditing API (E-A4C-2.8)

The auditing interface E-A4C-2.8 will be used by EMS to inform A4C on detected SLA violations during service provisioning. This interface is implemented as an API providing the following classes and methods:

- ***Class A4CClient***

  An object of this class implements the auditing functionality of an A4C client to communicate with an A4C server. Following public methods are provided:

  - ***AuditingReport createAuditingReport(acctgSessionID)***
    This method is used to create an AuditingReport object. The string parameter acctgSessionID identifies the affected accounting session for which an auditing report is to be generated.

  - ***ResultCode sendAuditingReport(AuditingReport)***
    Calling this method will cause the A4C client to send to the A4C server a VRR message constructed from the object parameter AuditingReport. This method returns immediately if there is an error in sending the VRR message, otherwise, it will return after a VRA message has been received. The value of the ResultCode is an integer denoting whether the task was completed successfully or not. A non-zero value means that there was an error.

- ***Class AuditingReport***

  An object of this class holds information about the violation event. This class provides the following public method:

  - ***int addReportItem(violationType, violationDegree, violationText)***
    Each violation event is described by invoking this method with the respective values of the parameters:

- The parameter *violationType* is an unsigned integer to identify a specific type of violation. It must refer to a pre-defined SLO-ID.
- The parameter *violationDegree* is an unsigned integer denoting the degree of violation which will have impact on the amount of reimbursement to be paid.
- The parameter *violationText* is an information string which will not be processed by A4C, but stored.

3.6.3. A4C Internal Auditing Interface (I-A4C-1.7)

The auditing interface I-A4C-1.7 will be used by an A4C client to report SLA violations to an A4C server. The communication is based on Diameter protocol, whereby a new command code is needed for the definition of VRR/VRA messages.

3.7. Implementation

The implementation description of the A4C infrastructure provides the internal architecture and implementation details of the A4C Server and A4C Client. Additionally, the A4C database structure is described.

3.7.1. A4C Server Internal Architecture

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 18. 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.

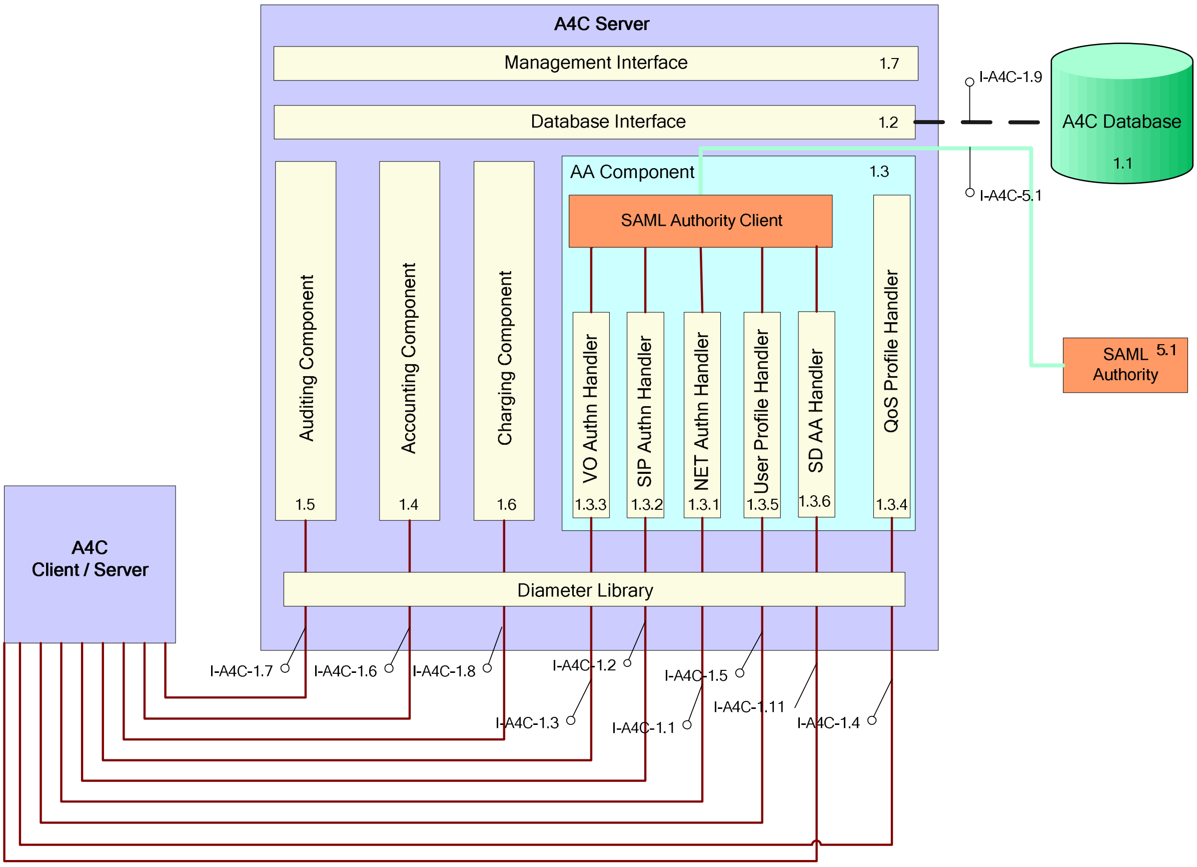


Figure 18 A4C Server Internal Architecture

3.7.2. A4C Server Implementation Details

As Figure 19 shows the main class in the server implementation is `A4CServer`. It uses an `AA_Server`, implemented as a singleton, for dealing with authentication, authorization and auditing messages. The accounting and charging related messages are dealt by an `Acct_Server` also implemented as a singleton. The proper delivery of Diameter messages to the right server class is done by the `A4CServerAllocator` class. The interface to the storage database is the `A4Cdb` class which in Akogrimo is based on *mysql*++ library. The communication between the A4C Server and the SAML Authority is done through the `SAMLAuthorityClient` class. This class is a C++ wrapper to a SOAP client that communicates with the SAML Authority for requesting and validating IDTokens.

The data received by the A4C Server is processed in one of the dedicated message handlers: `NetAuthnMsgHandler` for network-based authentication, `ProfileMsgHandler` for requesting service profiles for users, `VOAuthnMsgHandler` for validating IDTokens during the VO-Login phase, `AuditRecordMsgHandler` for processing violation messages, `AuthzMsgHandler` for authorization of access to context management and `AccountingSessionHandler` for processing accounting records.

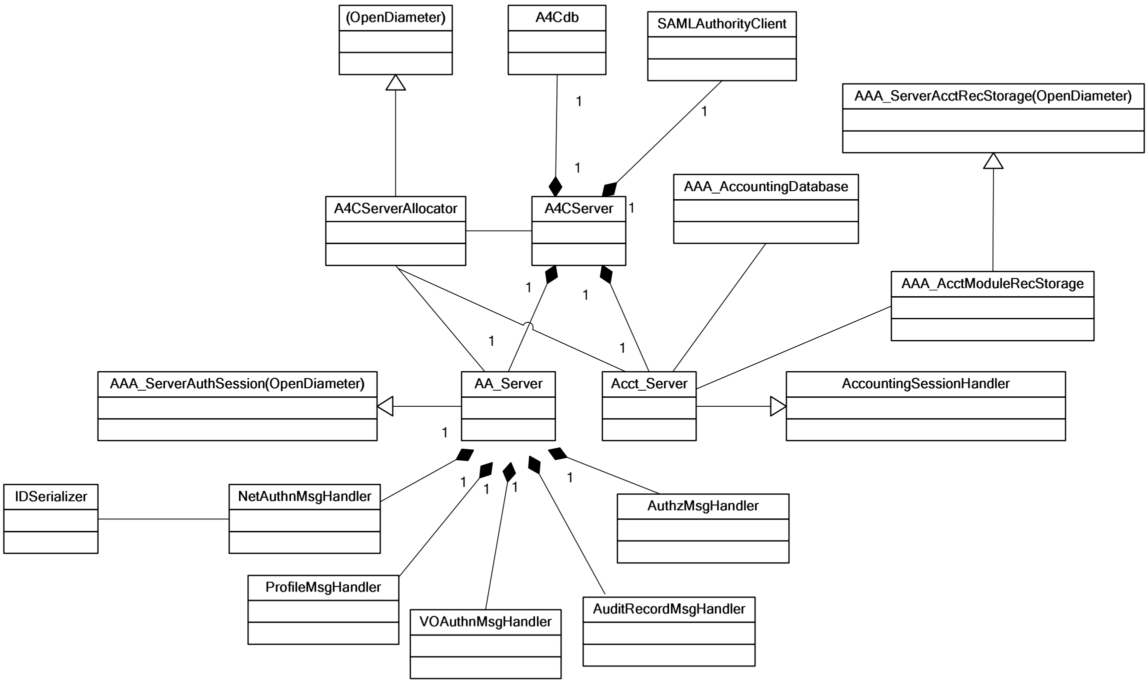


Figure 19 A4C Server - Class Diagram

3.7.3. A4C Client Internal Architecture

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 20. 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].

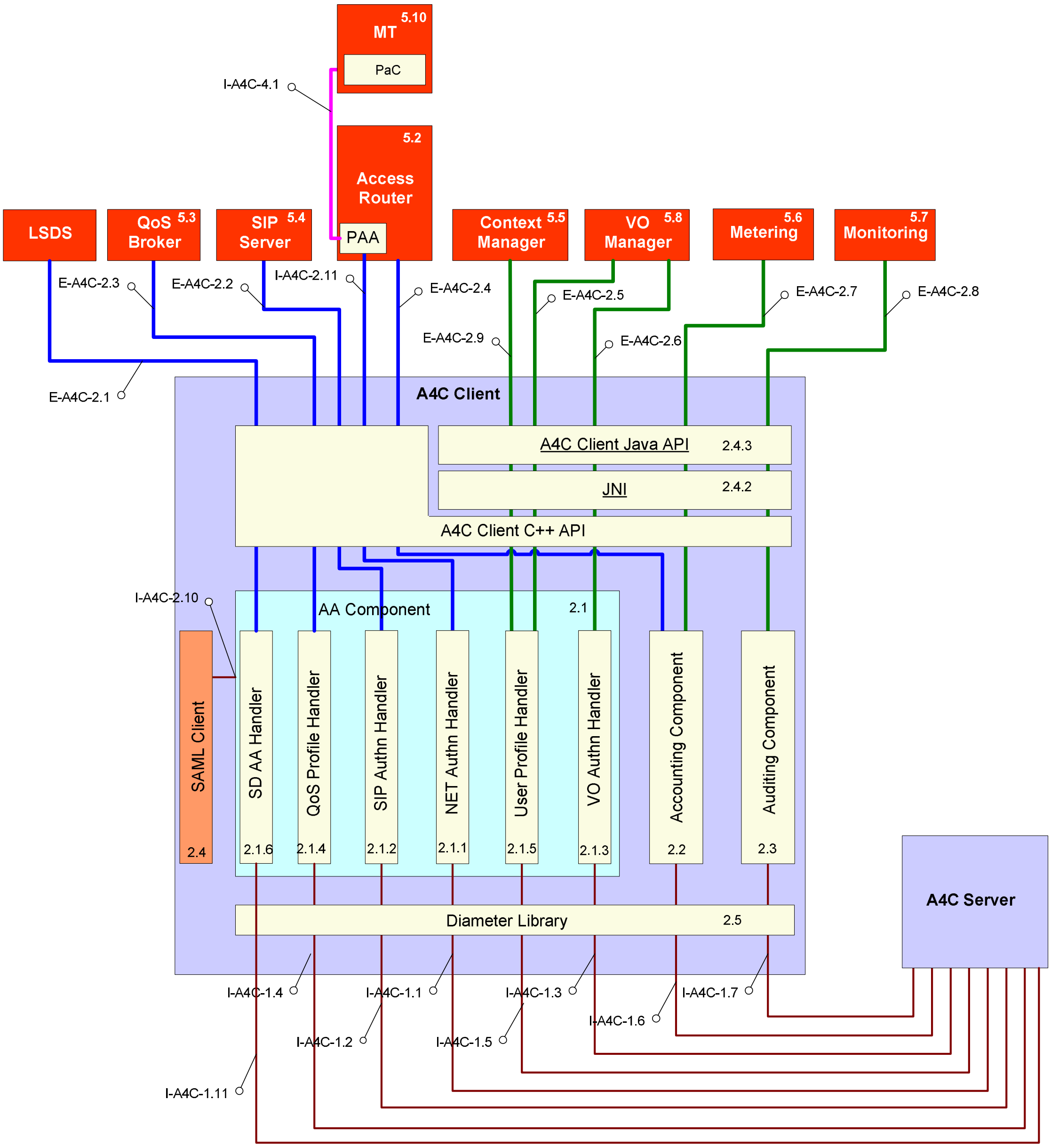


Figure 20 A4C Client Internal Architecture

3.7.4. A4C Client Implementation Details

A4C Client implementation is centered on the A4CClientEngine class as depicted in Figure 21. This class acts as an interface to the different modules that implement the different specific functionalities of the A4C Client:

- `AA_Profile`: used for request of service profiles
- `AA_Authz`: uses for request of service authorization.
- `AA_VOAuthn`: used for request of IDToken validation
- `PAA`: used for instantiating a PANA Authenticator for intermediation of network authentication
- `AuditingReport`: used to send violation reports
- `AccountingSessionManager`: used to create accounting sessions and to send accounting records in these sessions

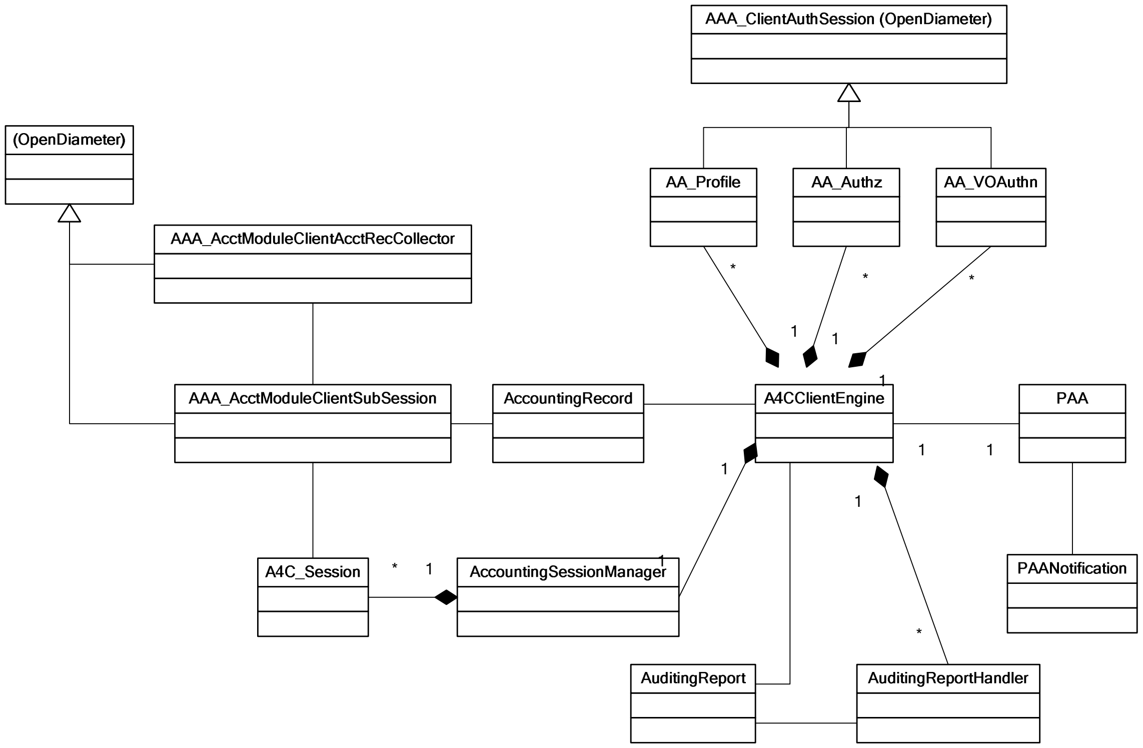


Figure 21 A4C Client - Class Diagram

3.7.5. A4C Database

The structure of the A4C database is shown in Figure 22. In the following the database tables and their fields are explained in detail.

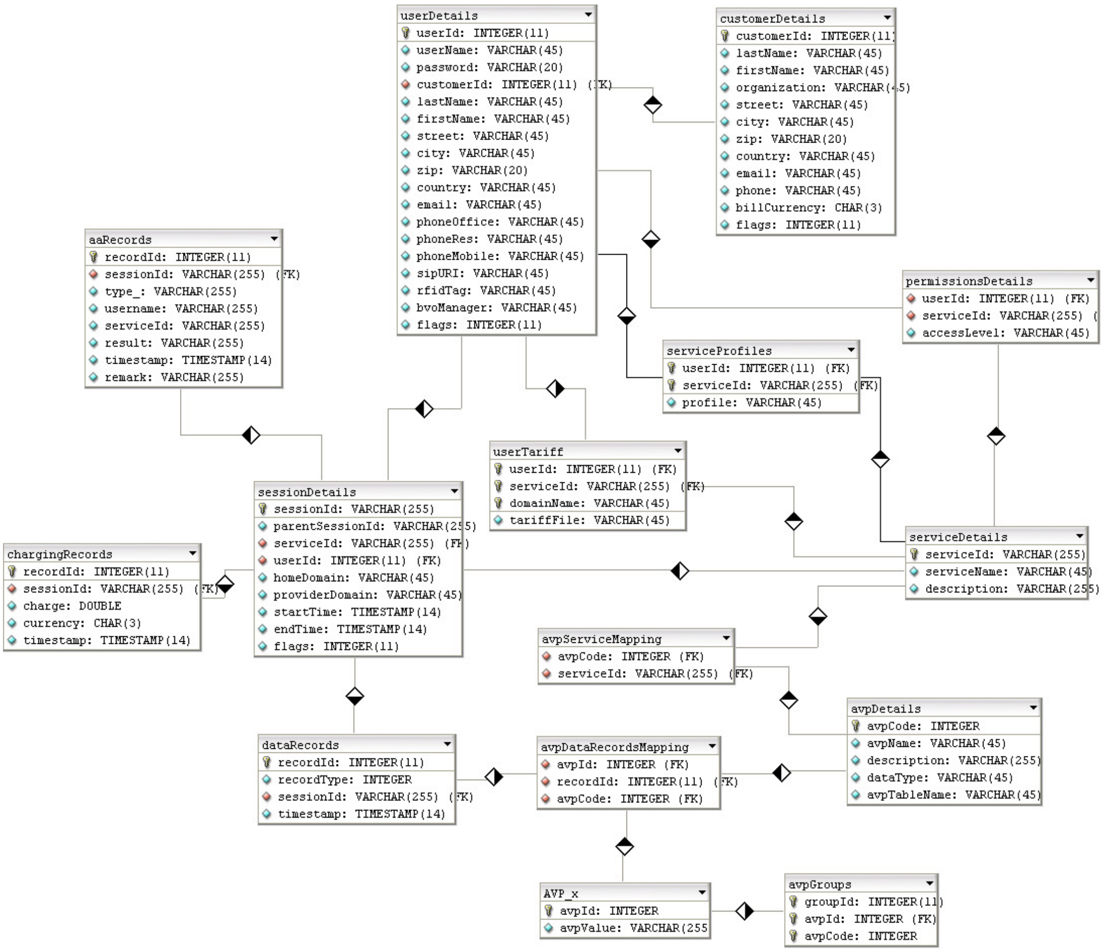


Figure 22 A4C Database Structure

User and customer details

The data fields of the userDetails and customerDetails tables are shown in Table 14 and Table 15 respectively. They include generic attributes like username, password, and personal details.

Table 14 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

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

Table 15 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 16 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 17). 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 25), which is specified by the avpTableName field.

The relation between services and AVPs are defined in the avpServiceMapping table (Table 18), which specifies the AVPs used for a particular service. The serviceProfiles table (Table 19) lists for every user the enabled services with the service profile. The profile defines the network QoS parameters enabled for the user.

The permissionDetails table (Table 20) specifies possible access rights per user and service.

Table 16 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 17 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 18 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 19 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

Table 20 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

Session details

The sessionDetails table (Table 21) is used to store all A4C session related information. 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 defines the state of the session, i.e. running, terminated, charged.

Table 21 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	State of the session, i.e. running, terminated, charged

Authentication and authorization records

The aaRecords table (Table 22) stores authentication and authorization events associated to a session. It specifies the user and service the request was received for. The type field includes the type of the request, e.g., network authentication, IDToken based authentication, profile retrieval. The result of the authentication and authorization, e.g., accept or deny, is stored in the result field. The timestamp specifies the time the record was created. Finally, the remark field can include additional information related to the authentication and authorization, e.g., the IDToken returned.

Table 22 Data fields of the aaRecords table

Data Field	Data Type	Description
recordId	Integer	Identifier of the record
sessionId	Varchar(255)	Reference to the session
type	Varchar(45)	Type of the authentication/authorization event, e.g., network authentication, IDToken based authentication, profile retrieval
username	Varchar(255)	Reference to the user
serviceId	Varchar(255)	Reference to the service
result	Varchar(255)	Result of the authentication/authorization.

Data Field	Data Type	Description
timestamp	Timestamp	Timestamp of the record
remark	Varchar(255)	

Accounting and auditing records

Accounting and auditing records are mapped to several tables in the database. The dataRecords table (Table 23) 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 24), which associates the recordId with avpId and avpCode. The avpGroups table (Table 26) is used to map grouped AVPs.

Table 23 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 24 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 25 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 26 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 27) specifies the tariff file associated to a certain user, service and domain. The tariff file contains the charging specification. The chargingRecords table (Table 28) stores data related to charging records.

Table 27 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 28 Data fields of the chargingRecords table

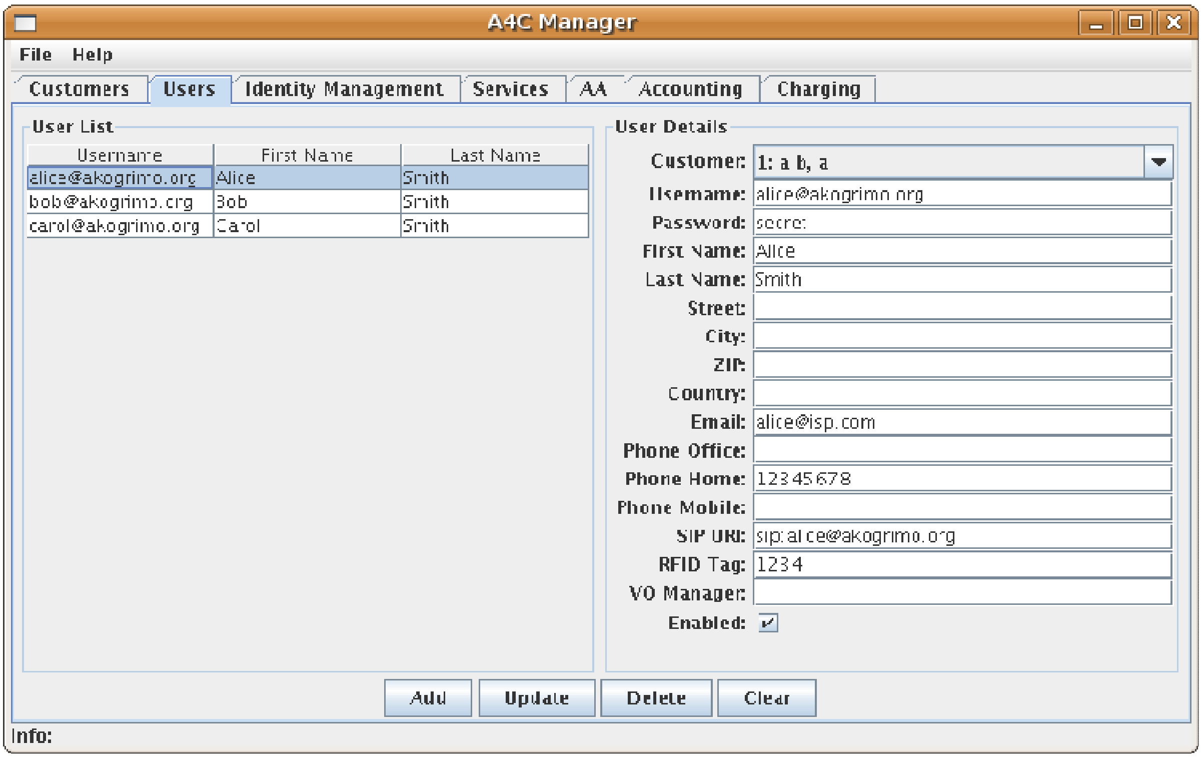
Data Field	Data Type	Description
recordId	Integer	Identifier of the record
sessionId	Varchar(255)	Reference of the session
Charge	Float	Charge for the session

Data Field	Data Type	Description
Currency	Char(3)	Currency used for the record
timestamp	Timestamp	Timestamp of the record

3.7.6. A4C Manager

The A4C Manager provides a graphical user interface to control and configure the A4C server. It enables the management of customers, users and services, the monitoring of authentication results and accounting records, and the performance of charge calculation and billing.

The A4C Manager is implemented as a stand-alone application in Java. It uses the A4C database in order to set, update, and retrieve information related to customers, users, authentication events, accounting and charging records. Figure 23 shows the users view of the A4C Manager. It enables the addition, update and removal of users in the A4C database. A user record includes the username, password, personal data (like name, address, e-mail address, and telephone numbers), the SIP URI, and the RFID tag.



User List

Username	First Name	Last Name
alice@akogrimo.org	Alice	Smith
bob@akogrimo.org	Bob	Smith
carol@akogrimo.org	Carol	Smith

User Details

Customer: 1: a b, a

Username: alice@akogrimo.org

Password: secret

First Name: Alice

Last Name: Smith

Street:

City:

ZIP:

Country:

Email: alice@isp.com

Phone Office:

Phone Home: 12345678

Phone Mobile:

SIP URI: sip:alice@akogrimo.org

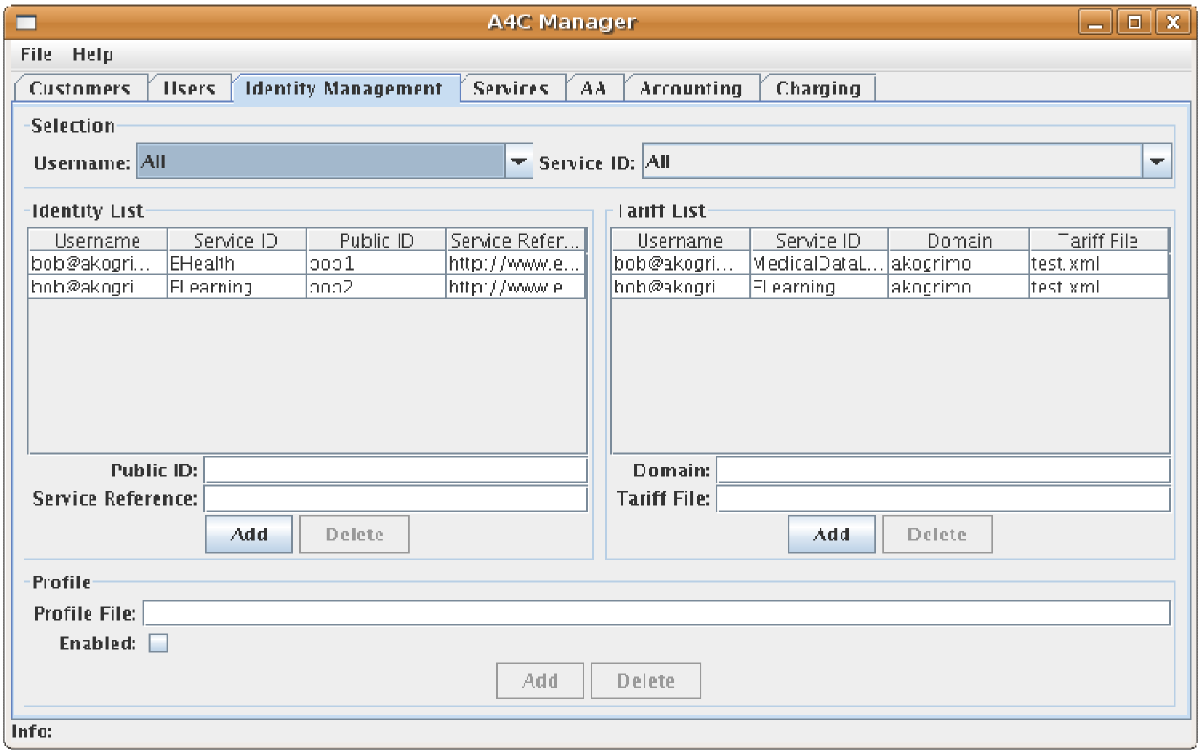
RFID Tag: 1234

VO Manager:

Enabled: ☑

Figure 23 Users View of the A4C Manager

Figure 24 shows the identity management view of the A4C Manager. This view enables the management of public identities, service profiles, and tariff specifications. Public identities can be assigned to a username-service ID pair, where the public identity will be used when the user accesses the given service. Similarly, the service profile can be assigned per user-service pair. Finally, the tariff specification defines the tariff file to be used in the charge calculation. Separate tariff files can be assigned per user, service, and domain.



Selection

Username: All

Service ID: All

Identity List

Username	Service ID	Public ID	Service Refer...
bob@akogri...	EHealth	0001	http://www.e...
bob@akogri	ELearning	0002	http://www.e

Public ID:

Service Reference:

Tariff List

Username	Service ID	Domain	Tariff File
bob@akogri...	MedicalDataL...	akogrimo	test.xml
bob@akogri	ELearning	akogrimo	test.xml

Domain:

Tariff File:

Profile

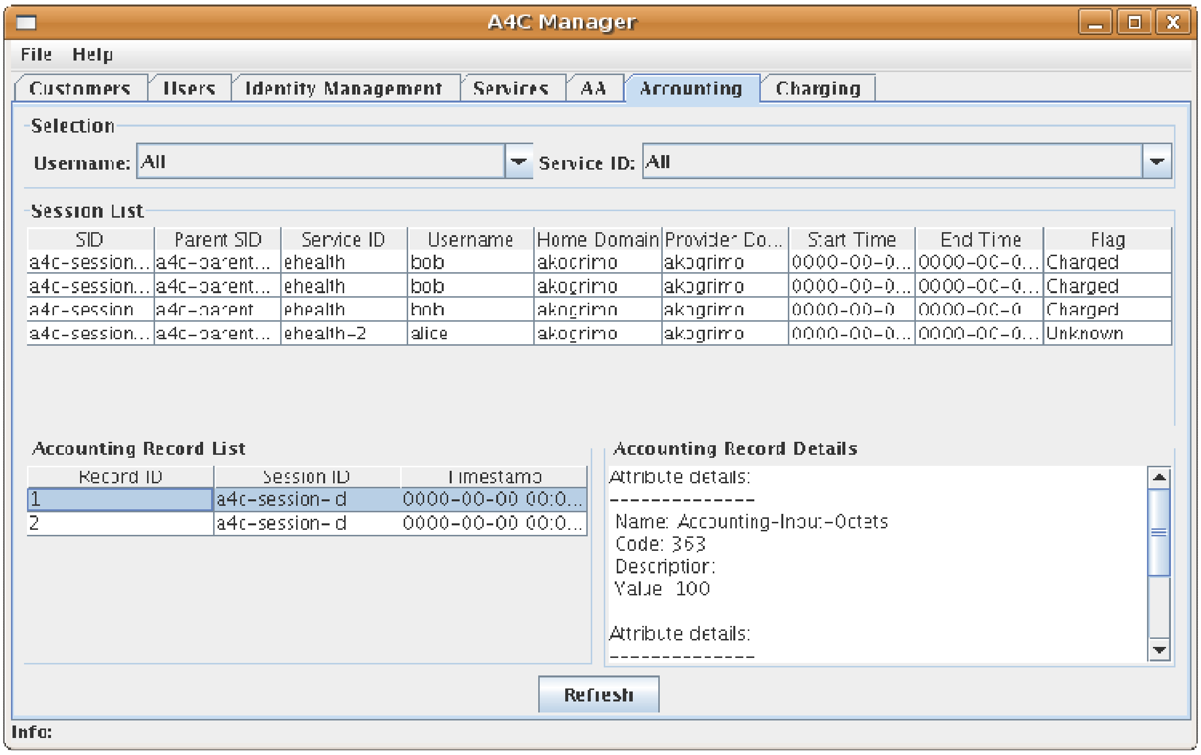
Profile File:

Enabled: □

Info:

Figure 24 Identity Management View of the A4C Manager

The accounting view of the A4C Manager is shown in Figure 25. It enables the monitoring of accounting records. Accounting records are listed per accounting session. In the accounting record details, all AVPs of a selected accounting record are shown. The session list can be filtered per user and service.



Selection

Username: All

Service ID: All

Session List

SID	Parent SID	Service ID	Username	Home Domain	Provider Co...	Start Time	End Time	Flag
a4c-session...	a4c-parent...	ehealth	bob	akogrimo	akogrimo	0000-00-0...	0000-00-0...	Charged
a4c-session...	a4c-parent...	ehealth	bob	akogrimo	akogrimo	0000-00-0...	0000-00-0...	Charged
a4c-session	a4c-parent	ehealth	bob	akogrimo	akogrimo	0000-00-0	0000-00-0	Charged
a4c-session...	a4c-parent...	ehealth-2	alice	akogrimo	akogrimo	0000-00-0...	0000-00-0...	Unknown

Accounting Record List

Record ID	Session ID	Timestamp
1	a4c-session-id	0000-00-00 00:0...
2	a4c-session-id	0000-00-00 00:0...

Accounting Record Details

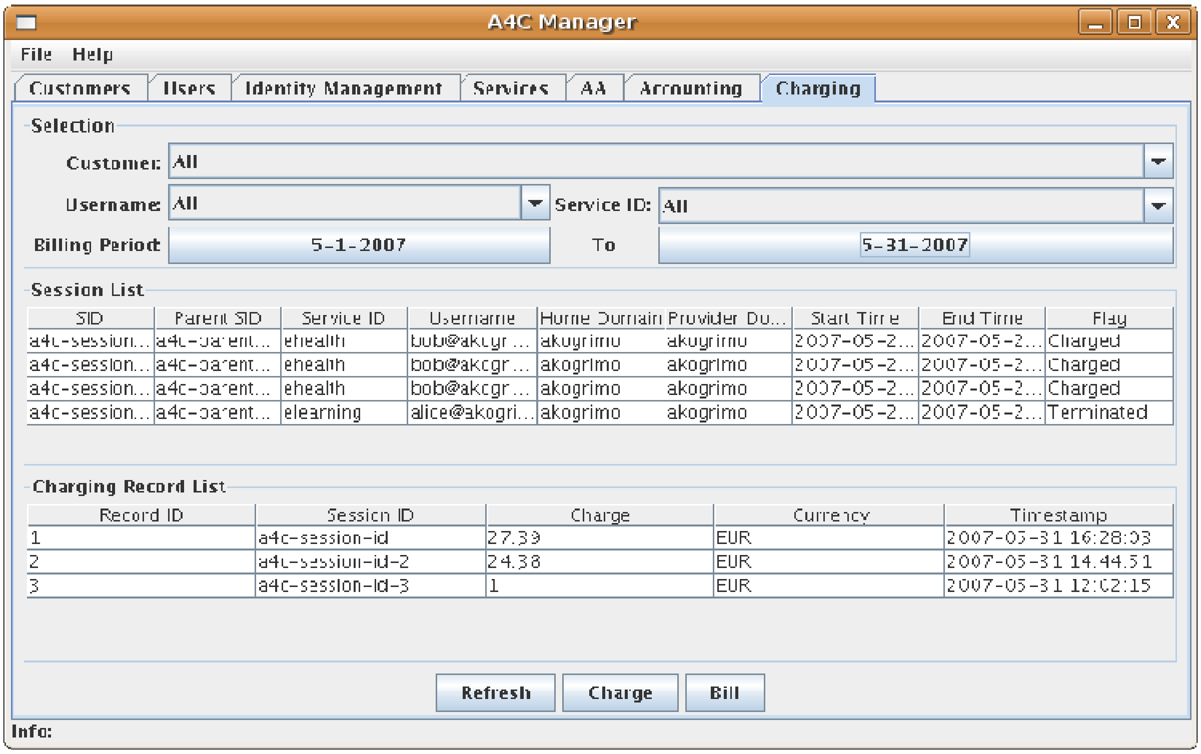
```
Attribute details:
--------------
 Name: Accounting-Input-Octets
 Code: 363
 Description:
 Value  100

Attribute details:
--------------
```

Info:

Figure 25 Accounting View of the A4C Manager

The charging view of the A4C manager is shown in Figure 26. It provides the view of charging records and the possibility to generate bills. Charging records can be filtered by customer, user, and service. For terminated sessions the charge calculation can be performed and a bill can be prepared for the defined billing period. The bill is created per customer separately.



Customer: All

Username: All

Service ID: All

Billing Period: 5-1-2007 To 5-31-2007

SID	Parent SID	Service ID	Username	Home Domain	Provider Du...	Start Time	End Time	Flag
a4c-session...	a4c-parent...	ehealth	bob@akogr...	akogrimo	akogrimo	2007-05-2...	2007-05-2...	Charged
a4c-session...	a4c-parent...	ehealth	bob@akogr...	akogrimo	akogrimo	2007-05-2...	2007-05-2...	Charged
a4c-session...	a4c-parent...	ehealth	bob@akogr...	akogrimo	akogrimo	2007-05-2...	2007-05-2...	Charged
a4c-session...	a4c-parent...	elearning	alice@akogri...	akogrimo	akogrimo	2007-05-2...	2007-05-2...	Terminated

Record ID	Session ID	Charge	Currency	Timestamp
1	a4c-session-id	27.39	EUR	2007-05-31 16:28:03
2	a4c-session-id-2	24.38	EUR	2007-05-31 14:44:51
3	a4c-session-id-3	1	EUR	2007-05-31 12:02:15

Figure 26 Charging View of the A4C Manager

3.7.7. Security Considerations

The A4C is a key component with respect to security, since it provides security mechanisms by itself, namely authentication and authorization. Other functions, like accounting, auditing, and charging work with highly sensitive data. Thus, the provisioning of A4C functions in a secure manner is of key importance. The implementation of A4C is done on top of OpenDiameter framework which has a library implementing Transport Layer Security (TLS). Since message authentication, integrity protection, and encryption between A4C components are implemented in an OpenDiameter library, A4C as an OpenDiameter application does not need to implement its own message transport security. To secure the communication between an (application) service and an A4C Server, the service has to use a separate A4C Client and be assigned a separate certificate. This allows the A4C Server to have separate TLS connections to every service and identify and authenticate the service based on its certificate.

4. Identity Management

Federated Identity Management is a crucial topic in Akogrimo due to the importance of the information conveyed and its implications for all layers.

Since there is more than one entity in Akogrimo which needs access to the information regarding user's identity, a system is designed in a simple and transparent way that fits all needs. Especially in authentication and authorization tasks, the involved components must ensure that they get all necessary information. Therefore a Distributed Federated Identity Management has been developed to fulfil specific requirements for privacy and security.

The following sections explain the concept of the Akogrimo approach and give a brief overview of the identity architecture and the involved components.

4.1. Identity and Profiles

The identity of the user is the key information in the further accounting of service usage and bill creation. The identity of a user was introduced in [D3.1.1] and further explained in [D3.1.3].

The user consists of a set of characteristics that are required to be stored either in the home domain or a service provider domain. Some of these attributes are considered to be of a generic nature and consist of personal user information (name, gender, age, bank account nr., etc.) Another subset of attributes is related to the home domain of a user and contains information like policies to be applied after a user logs in. The personal user information and home domain specific attributes should be stored inside the home domain of a user. A third subset of attributes contains service-specific information. This set of attributes can be stored either in the home domain or in the service provider domain that provides the service. A user profile consists of the set of all characteristics of a user (personal, home domain related, service related). An identity profile in Akogrimo is a subset of the whole user profile which contains just the attributes which are relevant for a service or a group of services. Many identity profiles which are service dependent can match to the same user.

In the normal life, the user identity which is usually presented depends on the type of organization. For instance, the identity profile that an employee presents at work -with attributes like social security number, Health insurance number among others- is different to the profile that a user presents, for example, at a travel agency – with attributes like interesting countries, type of trip…etc.

It is desired in Akogrimo that the user can choose the attributes (excluding demanded attributes) he wants to share within a Base Virtual Organization, which is seen actually as another service or compound service. Depending on several factors such as the content, the purpose and the nature of the VO, he will want to customize the set of attributes that form his identity profile for that service. In Akogrimo, the personalization will be done by the user, at the Base VO registration in a very straightforward way. He will be able to select the attributes that are already stored in the home domain and include other ones at that phase.

In this case, it is considered that the user can have several identity profiles and digital identities – but only one within one Base VO. It is a matter for the user if he wants to use the same identity profile for several VOs.

Table 29 shows relevant terms with its characteristics for identity management and the purpose of the usage and the components involved. A combination of these is used to guarantee pseudonymity.

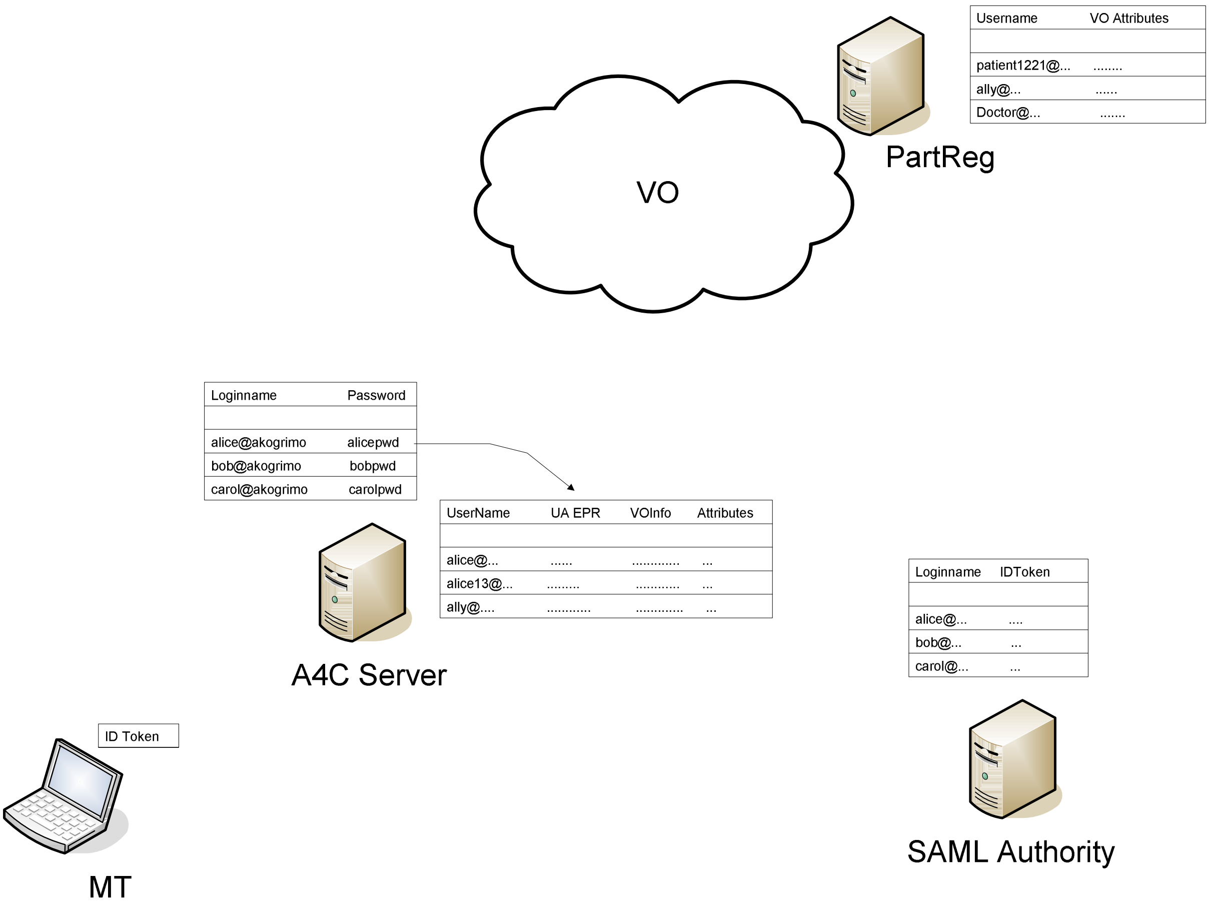
Table 29 Components for identity handling

Name	Type	Purpose	Involved Components
Login-Name	String	Authentication	MT, A4C, SAMLAuthority
Username	String	Authentication	MT, A4C, SAMLAuthority, Participant Registry
IDToken	String	Authentication	MT, A4C, SAMLAuthority
Profile	List of Attributes	Authorization	MT, A4C, Participant Registry
Attribute	String	Authorization	MT, A4C, Participant Registry
Role	Attribute	Authorization	MT, A4C, Participant Registry

4.2. Architecture

There are three main components that handle with Identity Management issues, the A4C Server (include the SAML Authority), the Participant Registry and the Mobile Terminal.

Figure 27 shows a sketch of the components implied in the Distributed Federated Identity Management that are described in this section. All the components are described in detail in other chapters in this document or in other deliverables. Hence they are only characterized in a short way in the following subsections.



Username	VO Attributes
patient1221@...	....
ally@...	......
Doctor@...	....

Loginname	Password
alice@akogrimo	alicepwd
bob@akogrimo	bobpwd
carol@akogrimo	carolpwd

UserName	UA EPR	VOInfo	Attributes
alice@...	......	....	...
alice13@...	....	....	...
ally@....	....	....	...

Loginname	IDToken
alice@...	....
bob@...	...
carol@...	...

Figure 27 Distributed Storage of the Identity

4.2.1. A4C Server

The main tasks of the A4C Server related to Federated Identity Management are to authenticate the user during the initial login and all later authentication phases and to store the generic user profile. When it validates a user during the login–phase the SAMLAuthority is requested to generate an IDToken for him and to issue an authentication assertion. If an authentication is requested from a service the A4C Server received the Identity Token of the user and forwards it to the SAMLAuthority. From the Authority it will receive the login-name of that user so it is able to validate user's identity using the identity overview stored in a database.

For Identity Management purpose it has to communicate with

- Mobile Terminal: Send IDToken after login
- Policy Management System of services and VOs: to receive validation requests
- SAML Authority: Send IDToken and receive SAML Authentication Assertions
- Virtual Organization Manager: To update any changes in user profiles between Participant Registry and A4C Database

Further information to the A4C Server can be found in Chapter 3 and in [D4.2.2] (Chapter 4).

4.2.1.1. SAMLAuthority

The SAMLAuthority is responsible for validating user's identity and generating SAML assertions [SAMLAssertion]. Each time the A4C Server receives a request to validate an Identity Token, the request is forwarded to the SAMLAuthority. The SAMLAuthority responds by sending the related log-in name, thus the A4C Server is able to check if the IDToken and the provided identity match. It can also send identity profiles of the user in a SAML attribute assertion.

- SAML Database: In the SAML Database all Authentication Assertions are stored together with some user related information that is needed to validate the tokens.
- Interface SAMLAuthority – A4C Server: Since all information that is sent by the SAMLAuthority passes to A4C Server the SAMLAuthority needs only the interface to the A4C Server even though the information the Authority produces may be sent to more components.

Further information about the SAMLAuthority and it's interfaces and functions can be fond in [D4.2.2] (Sections 4.3.1, 4.4.3, 4.4.4).

4.2.2. Participant Registry

The Participant Registry is part of a VO and thus not part of the network architecture. It stores all relevant information that user needs inside this VO (service specific profile). The design and implementation of the Participant Registry is not a part of WP4.2 and described in [D3.1.2] (Section 6.4.1), [D4.4.1] (Section 3.1) and [D4.4.2] (Section 2.1.5).

The purpose of the Participant Registry related to Identity Management is to store the profile information required by the VO and provide it to the Policy Management System when requested.

- Interfaces to A4C: The Participant Registry needs to communicate with the user's Home A4C to update profile changes. This is done by communication via the VO manager, since both A4C and Participant Registry have interfaces to that component.

4.2.3. Mobile Terminal

The Mobile Terminal is responsible for the IDToken and thus it stores the user's identity. Each time an authentication is needed it sends the IDToken and the current username to the entity that requests authentication. From there the token is forwarded together with the username to the Home A4C for verification.

- Interfaces to A4C: The Mobile Terminal needs communication with the A4C Server for receiving the IDToken. This is done via an A4CClient at the Access Router.
- Interfaces to Services and Policy Management Systems: For the use of services and for authorization purposes the Mobile Terminal must have an interface to all services and their Policy Management System.

A detailed description of the Mobile Terminal and its interfaces and functions can be found in [D4.1.1] (Sections 2.3.4, 2.4.6, and 3).

Additionally, the Mobile Terminal is also responsible of the module called as the Identity Selector. This module enables the user to select the identity profile used when using a service. The Identity Selector interfaces with the A4C for receiving and sending information about personalization of the identity profiles.

The Identity Selector provides mainly the functionality to:

- Create a new profile: This can occur under different circumstances, for instance, at VO registration the user can create the desired profile for the VO or at any other situation that the user wants another profile. The Identity Selector offers a simple and intuitive Graphical User Interface to the user in order to make the creation as simple as possible.

- Select the desired identity profile for the service: The user just clicks on the Identity Selector GUI that appears at the Mobile Terminal when requesting a service.

Other possibilities can be chosen, like the selection of the default desired identity for the services at one session.

4.3. Communication Schema

Figure 28 shows the communication that is needed for Federated Identity Management in Akogrimo.

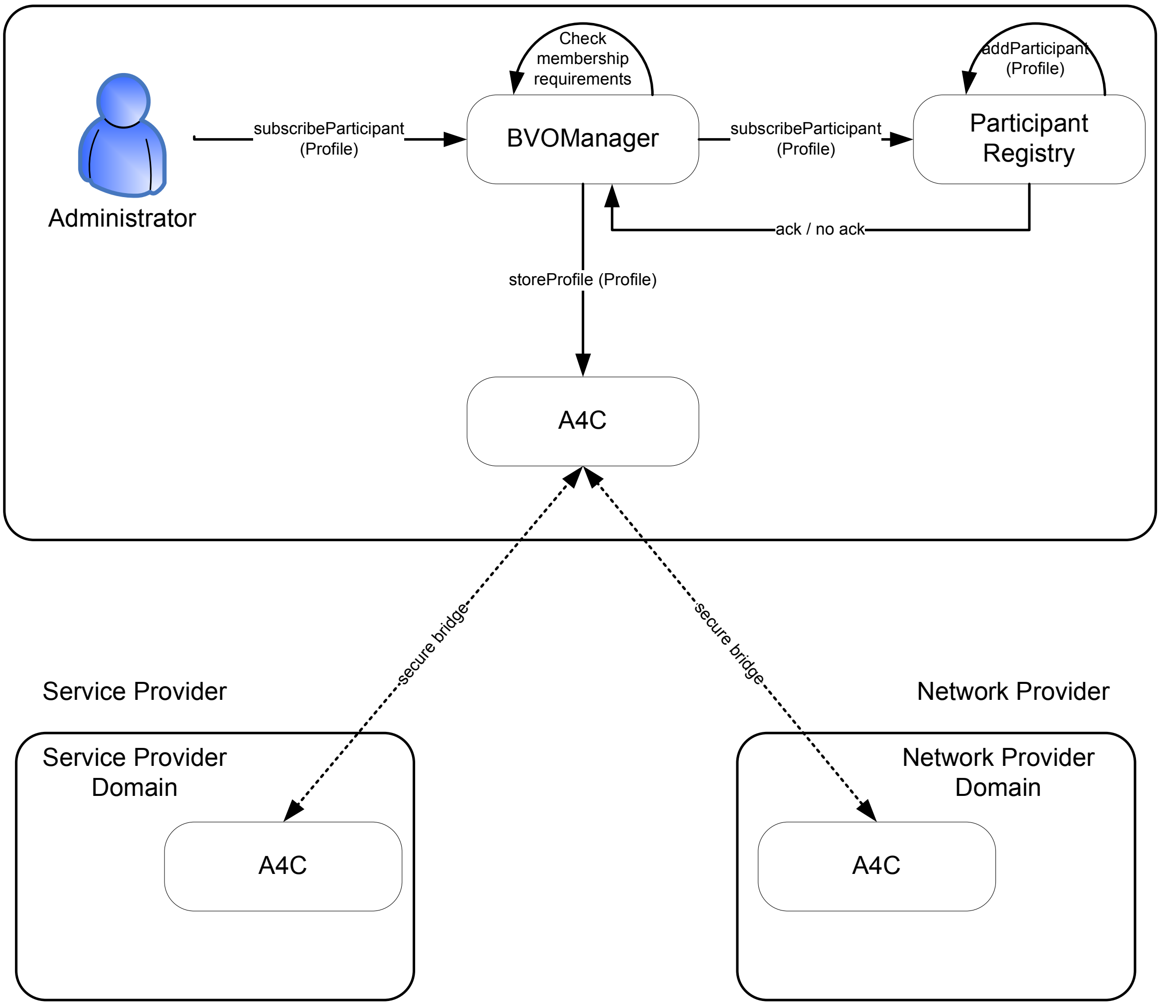


Figure 28 Architecture (A4C)

4.3.1. Authentication

There are two different types of authentication used in Akogrimo, the initial login and other authentications afterwards.

Login

First the user sends his login-name and password to the A4C where it is checked. If the credentials are valid the SAMLAuthority generates an IDToken for him, stores his login-information in the SAML Database and sends a SAML Authentication Assertion to the A4C. After the login-phase the user is provided with a list of possible VOs and usernames he may want to use within the domain. He selects one and is from that point on is only visible with this username.

Authentication after login

If some authentication is needed after the login-phase, the user sends his current username and his IDToken to the entity that needs his authentication from where it is forwarded to user's Home A4C Server. The A4C Server requests the SAMLAuthority to validate the token and receives from the Authority user's login-name. The A4C Server checks if the identity provided by the user matches one of the possible identities that are composed with the login name.

4.3.2. VO Subscription

The VO subscription of a user is an offline process. The user is requested to generate a profile he wants to use within that VO and the required data (e.g. username, personal preferences, VO requirements, etc.) are committed to the Participant Registry where they are stored. To this data any Service specific profile data can be added. The user may chose if he wants to build up a new profile from scratch or if he uses parts of one already stored in his Home A4C.

4.3.3. Authorization

Following the concept of Distributed Federated Identity Management the main components involved in authorization are the Participant Registry (where authorizations done in BVO are stored) and the Home A4C of the user:

When a user needs to be authorized he first must validate his identity by sending the IDToken and his current username to the entity that requires the authorization. There he is first authenticated as described in Section 4.3.1. After that the Policy Management System requests the user profile that is stored in the Participant Registry. Based on the provided profile information the Policy Management System will decide if the user is authorized for the requested action.

4.4. Key Management Infrastructure and Certification Authority

With the use of encryption methods provided by public-key cryptography a communication between partners is more secure. But this encryption method is not protected against malicious acts whether steeling private keys or act as man in the middle during the setup of a communication. To make sure that the holder of a key pair is the entity (user or service) he claims to be, Akogrimo provides a Public Key Infrastructure (PKI).

The PKI consists of at least one Registration Authority (RA) and one Certification Authority (CA). Whenever an entity needs a certificate (X.509) that claims its identity, a certificate request has to be sent to the RA. The RA verifies the identity of the requester by given credentials and forwards the request after a successful verification to the CA. The CA issues the certificate and signs it – it is expected that each member trusts the CA, thus the CA-signature means doubtlessness. To guarantee this doubtlessness it has to be assured that

- Each member trusts the CA
- A policy guarantees a reliable way of validating identities
- A policy guarantees a reliable way of issuing certificates
- The certificates can be removed in case of misuse

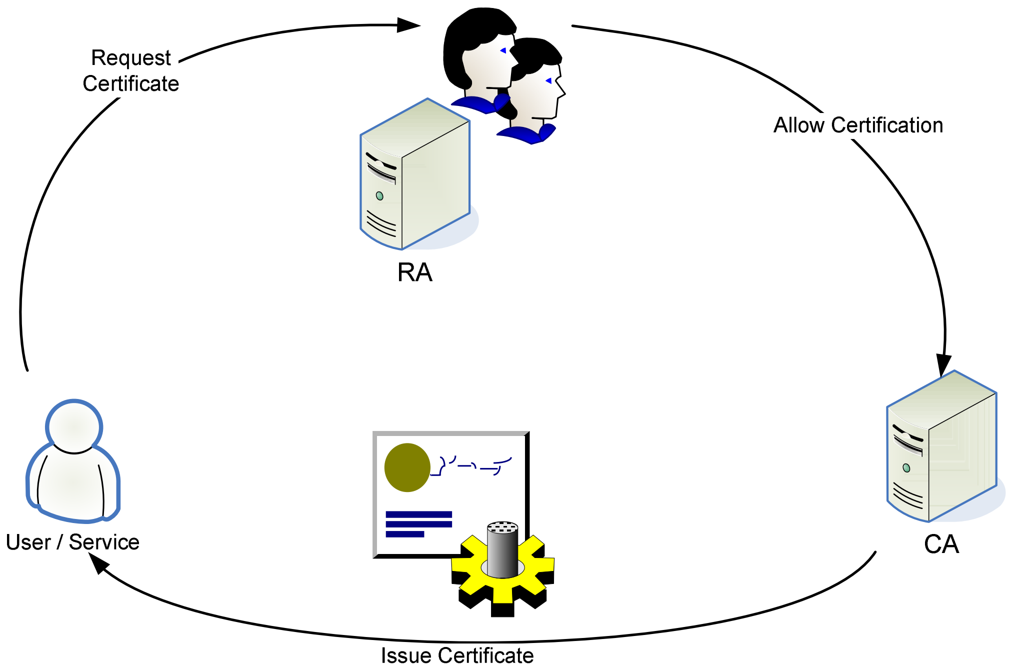


Figure 29 Certificate Issuing

For the fist issue it is considered to provide a root CA that is certified by a worldwide accepted CA (or self signed when each member has trust in this Authority without a certificate from a trusted 3rd party). This Akogrimo-Root-CA will handle certificate requests for entities and especially for other CAs that are intended to be used in Akogrimo. E.g. when each domain owner wants to use his own CA, these CAs have to be certified by the Akogrimo-Root-CA, so that all entities from other domains have the ability to trust the certificates of that CA too. In that way the trust to each CA used in Akogrimo is ensured by inheritance:

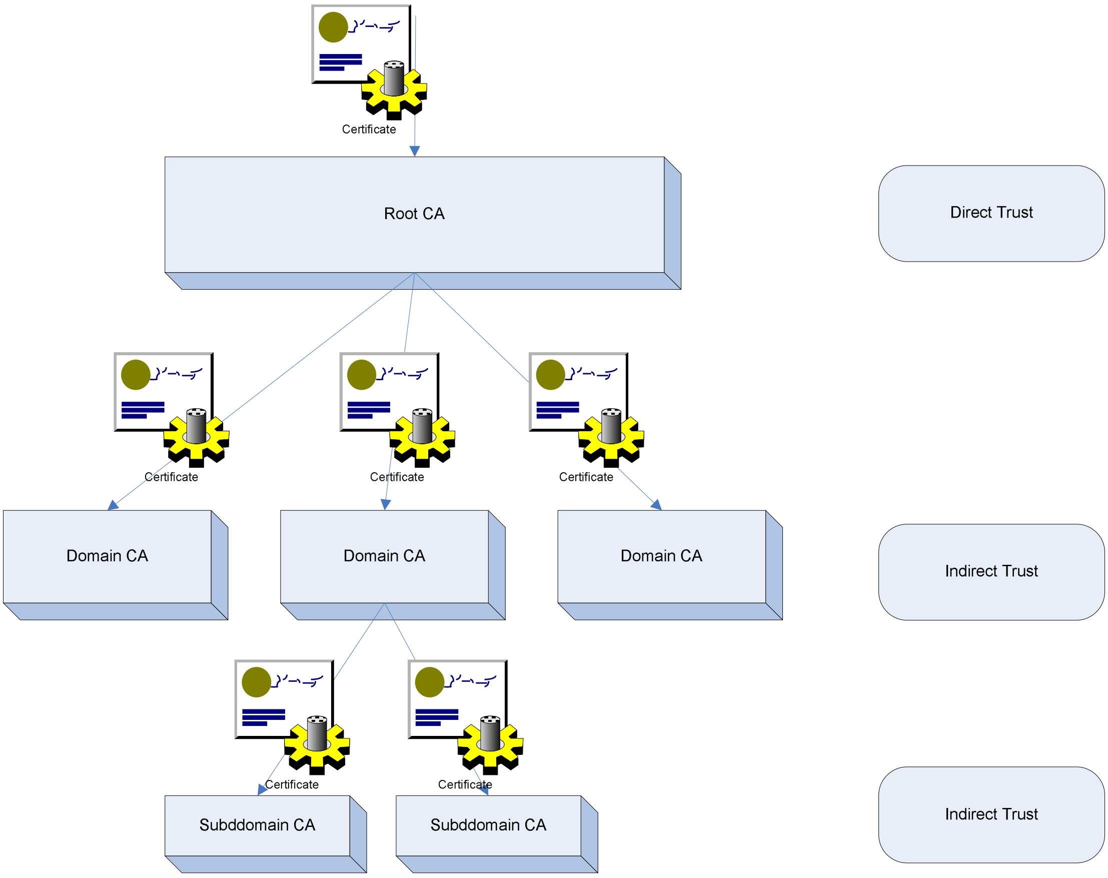


Figure 30 CA Hierarchy

If it is not feasible to establish a Root-CA that is trusted by each member, all CAs in different domains (or in a larger scale if possible) must have certificates from other trusted CAs.

A revocation list is used to verify the status of certificates. When a user reports misuse (or it is detected by other sources) the certificate is revoked and this revocation is issued. Due to constant updates of the revocation list all Akogrimo members are aware that this certificate is not valid any longer.

It is thought that the Home A4C Owner acts as RA for all users subscribed to that domain. The RA function for the services in the VOs can be done by the Owner of the Participant Registry. In that case, the entities that already deal with the identity verification have only another task to do instead of setting-up a new component in the architecture.

For the use of certificates it is proposed that all user-certificates are stored at their Home A4C as part of their profiles and thus transferred to the VO Participant Registry when the user subscribes.

The service-certificates are stored initially at the Participant Registry but transferred to user's Home A4C as important VO information when he subscribes.

This approach guarantees that all certificate information is always up-to-date, since the A4C will receive revocation information for user certificates and the Participant Registry for service certificates.

4.5. Building Confederations

Since the Akogrimo Identity Management is oriented towards the Shibboleth [Shib] approach, it is possible to migrate a user to other federations. With respect to the eduGAIN [eduGAIN] project, which will combine several national federations into one European confederation, the need for Akogrimo to be compatible with these AAIs grows. Thus, the Akogrimo Identity Management concept must allow an easy connection to one of the participating federations.

To show a possible scenario via a thought process, a connection to a Shibboleth federation will be established. The requirements for the Akogrimo Identity Management are as follows:

- Use or provide the schema for the connected Shibboleth federation
- Provide an interface that allows the Akogrimo users a connection to the Shibboleth federation
- Provide an interface that allows the Shibboleth federation users a connection to Akogrimo

The use of the accordant schema is required for the interaction with the SPs of the Shibboleth federation. Thus, it is recommended that a user provides at least all attributes to his Home A4C Server to fulfil the EduPerson [EduPerson] schema or provides them during the migrating process to the Shibboleth federation at the gateway. Only this guarantees a seamless user-migration into the international confederation.

The bridging element between the Akogrimo federation and a Shibboleth federation is split into two parts: The incoming bridge (from Shibboleth to Akogrimo) and the outgoing bridge (from Akogrimo to Shibboleth).

The outgoing bridge is a Shibboleth Identity Provider (SIdP) with a web based portal. The Akogrimo user has to start his browser and to access the website. After providing his IDToken to the bridge, which acts as a service, he will receive a Shibboleth browser artefact from the proxy SIdP. The gateway stores only those attributes that are needed for the Shibboleth federation and are not part of user's Akogrimo–profile.

Figure 31 shows the message flow for an Akogrimo user, that already has a virtual account at the bridge, i.e. the proxy IdP, for using the Shibboleth environment. He sends his Shibboleth artefact, which he has got during the token-conversion from the bridge element, to the SP of that resource he wants to access. The SP discovers by using a Where-Are-You-From (WAYF) the proxy IdP as the user's Home IdP and sends the request for attributes to the bridging element. There the IDToken assigned with the artefact is used to get the user's attributes from his Home A4C-Server. The bridge converts the received Akogrimo assertion into Shibboleth format and sends the attribute assertion to the SP.

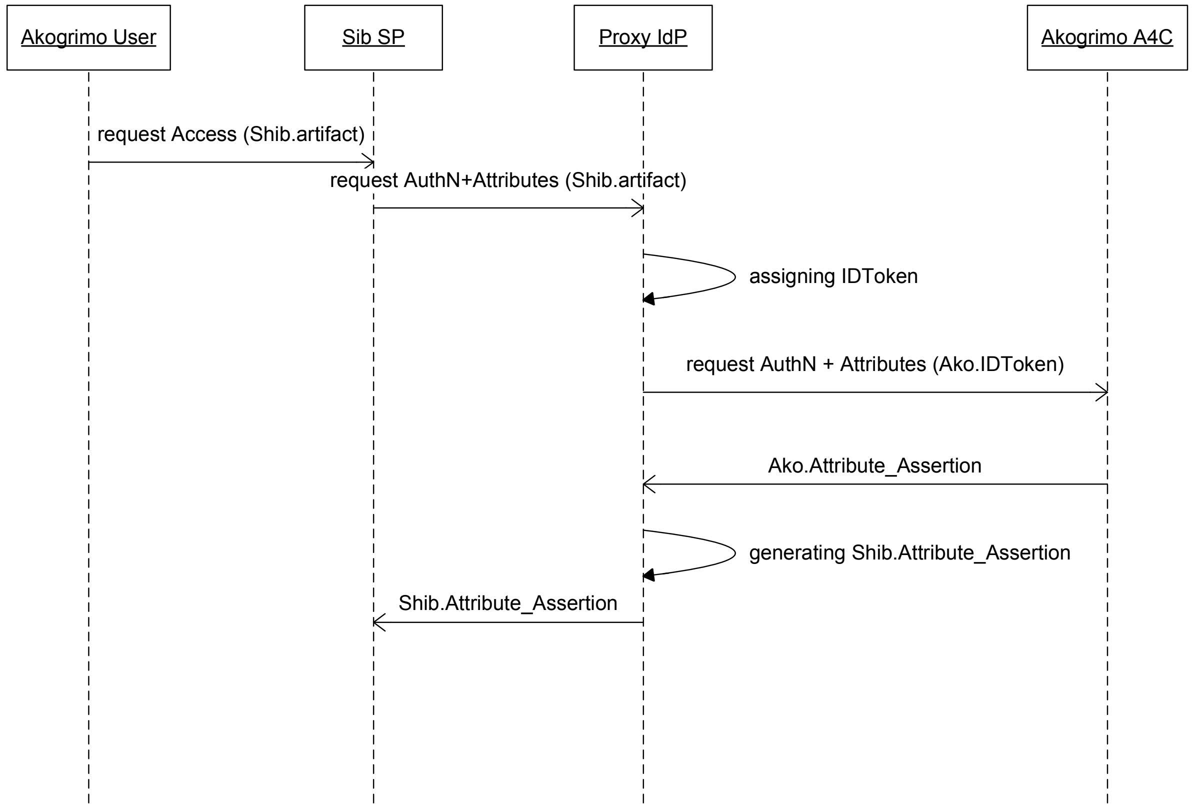

Akogrimo User
Sib SP
Proxy IdP
Akogrimo A4C
request Access (Shib.artifact)
request AuthN+Attributes (Shib.artifact)
assigning IDToken
request AuthN + Attributes (Ako.IDToken)
Ako.Attribute_Assertion
generating Shib.Attribute_Assertion
Shib.Attribute_Assertion

Figure 31 Message Flow for Akogrimo user accessing a Shibboleth resource with focus on attribute transmission. The bridging element acts as Shibboleth Identity Provider and Attribute Authority.

For the incoming bridge, a proxy A4C Server has to be established. The Shibboleth user visits a website provided at the bridge from where the Proxy A4C Server is contacted and the user's Identity Token is received for download. After storing this token on his device and starting the Akogrimo middleware, the Shibboleth user is able to use all Akogrimo SPs. If in some cases more attributes from the user are needed than provided by his Attribute Authority (AA), he has to enter them into a form on the website. The proxy A4C server stores only those required attributes, the user's IdP is not aware of. All other attributes are provided by the user's attribute authority in his home federation. Figure 32 shows in brief the message flow for that constellation, when a Shibboleth user wants to access an Akogrimo resource; the schema is an analogue to that used in the other direction.

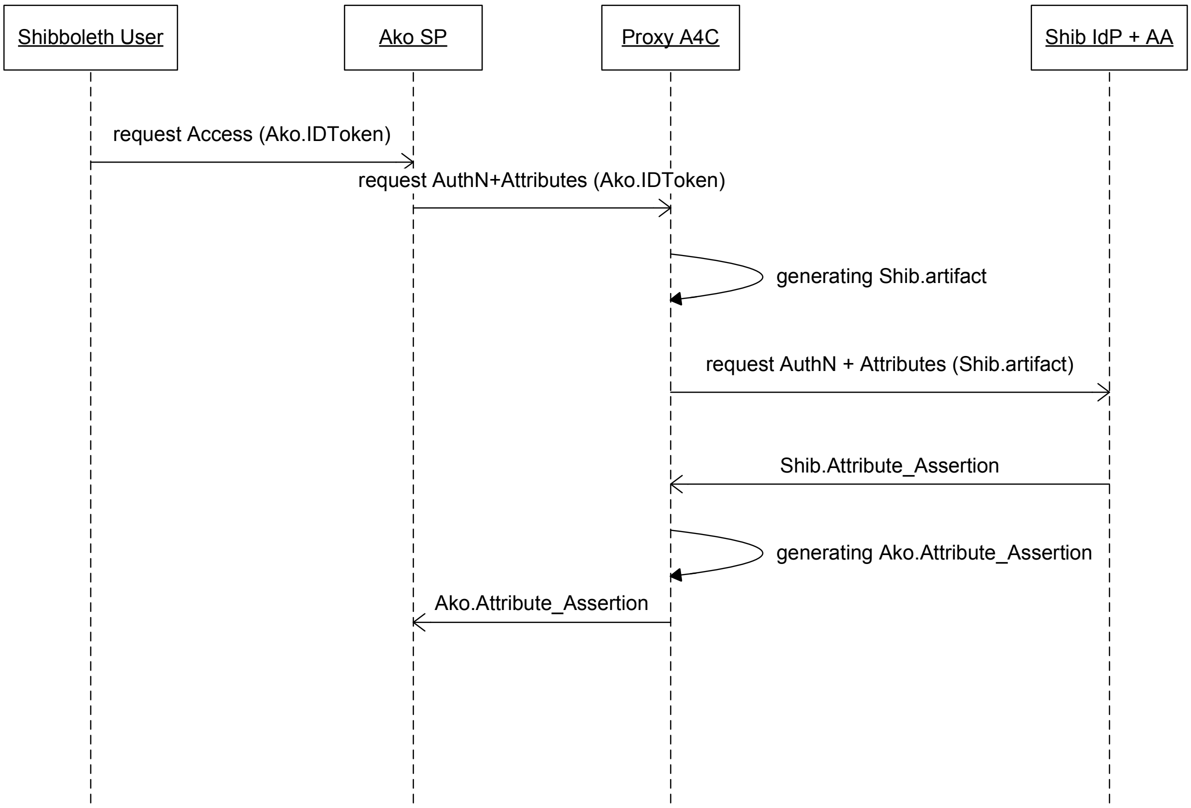

Shibboleth User
Ako SP
Proxy A4C
Shib IdP + AA
request Access (Ako.IDToken)
request AuthN+Attributes (Ako.IDToken)
generating Shib.artifact
request AuthN + Attributes (Shib.artifact)
Shib.Attribute_Assertion
generating Ako.Attribute_Assertion
Ako.Attribute_Assertion

Figure 32 Message Flow for Shibboleth user accessing an Akogrimo resource with focus on attribute transmission. The bridging element acts as Akogrimo A4C Server.

5. Context Manager

This section describes the Context Manager (CM), part of the Akogrimo Architecture. The main purpose of CM is to manage context for Akogrimo entities. In the current implementation an entity is a person, but in a more general setting it could also be a place, a device, etc. In general, context of an entity is any information that can be used to characterize the situation of the entity [Dey99].

5.1. Introduction

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. The section also contains an implementation status overview to indicate what is implemented and what is conceptually designed.

5.1.1. Requirements

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 19) or in phase 2 (beyond PM 19). Compliance is stated as C (Concept, the topic has been investigated and a solution proposed), I (Implementation, the proposed solution has been implemented), PC (partial compliance), or N (No compliance)

Table 30 Context Manager – Fulfilment of functional requirements

Req. nr	Description	Phase 1[1]	Phase 2[2]
F 1	**Gather context (end user information)**	**C/I**[3]	**C/I**
F 1.1	Presence	C/I	
F 1.2	Position	C/I	
F 1.2.1	Position – RFID based	C/I	
F 1.2.2	Position – based on e.g. GPS or WLAN		C/I
F 1.2.3	Position – mapping from coordinates to area/building plan		C/I
F 1.3	Terminal capabilities for users terminal	C/I	
F 1.4	Discover services in user's proximity	C/I	
F 1.4.1	Terminal-based discovery (discovery performed by user's terminal)		C/I
F 1.4.2	Centralised, directory-based discovery (match users position	C/I	

[1] Within Project Month 19

[2] Beyond Project Month 19

[3] C: Concept, I: Implementation, P: Partial compliance, N: No compliance

Req. nr	Description	Phase 1[1]	Phase 2[2]
	with services for that position)		
F 2	**Distribute context (to context consumers)**	**C/I**	
F 2.1	Offer subscription for user context	C/I	
F 2.1.1	Possibility to limit context scope (specify specific parts of context to subscribe to, e.g. only location)	C	C
F 2.2	Query (single response) for user context	C/I	C/I (P[4])
F 2.2.1	Possibility to limit context scope (specify specific parts of context to subscribe to, e.g. only location)	C	C
F 2.3	Ontology-based queries		P[5]
F 2.4	WS-based interface for context consumers	C/I	
F 2.4.1	Mechanism for returning context: Regular WS with callback	C/I	
F 2.4.2	Mechanism for returning context: WS Base Notification or WS Pubscribe		C/I

Table 31 Context Manager – Non-functional requirements

Req. nr	Description	Phase 1	Phase 2
N 1	Availability		N
N 1.1	Process surveillance, automatic startup		N
N 2	Performance		N
N 3	Scalability		N
N 4	Security		C/I
N 4.1	Admission control: Authenticate context request		C
N 4.2	Admission control: Authorize context request		C/I

[4] Compliance only for WS-based interface, not for WS-N interface

[5] Parts of Concept has been elaborated

5.1.2. Implementation status

In this section we specify the implementation status of the Context Manager. Table 32 shows the implementation of context related aspects.

Table 32 Context Implementation status

Topic	Concept	Implementation
Context delivery (WS, WS-Notify)	ok	ok
Context sources: SIP Presence SLP (Device overview) RFID (location) BT (BT Devices around user. GPS Pos)	ok ok ok ok	ok ok ok deprecated
Location handling Map pos -> area plan	ok	deprecated
Security (Access to context)	Ok	partly
Ontology support	Partly	-

Some of the requirements mentioned in Table 30 for phase 2 are not fully. GPS positioning implementation was stopped for Linux support as it was not needed, Terminal-based Discovery has not been needed, limitation of context scope has not been needed and ontology-based queries were deemed to be a too big redesign for the remaining resources available.

Some work has been done to test the non-functional requirements, and is an ongoing work.

5.2. Interfaces

The interfaces listed in Table 33 are shown in the Context Engine design (Figure 44, Section 5.4.1.).

Table 33 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	SOAP
SIP PA (WP 4.2)	CM (WP4.2)	- Obtain user presence	SIP SIMPLE

Server: Component (Layer) [Label]	Client: Component (Layer)	Purpose	Protocol
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
CM (WP4.2) [E-CM-3)	CM (WP4.2)	- Obtain Bluetooth devices near user - Obtain GPS position from GPS Bluetooth device	Java API

The external interfaces the Context Manager has are listed in Table 33, it describes interfaces both where CM acts as a client and where it acts as a server. 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

Basic flows are included in Figure 33. See also Figure 43 for how it is used. The interface is described more in section 3.4.4.

5.2.2. Context Manager – Context consumer

The Context Manager provides a Web Service interface to context consumers. Results are returned over a so called call-back interface, i.e. the Context Consumer offers a Web Service interface that the context manager may call to return results. An example of Basic flows are shown in Figure 33, methods are shown in Table 34 and Table 35. The web service that the context manager calls to return results needs the method(s) mentioned in Table 35.

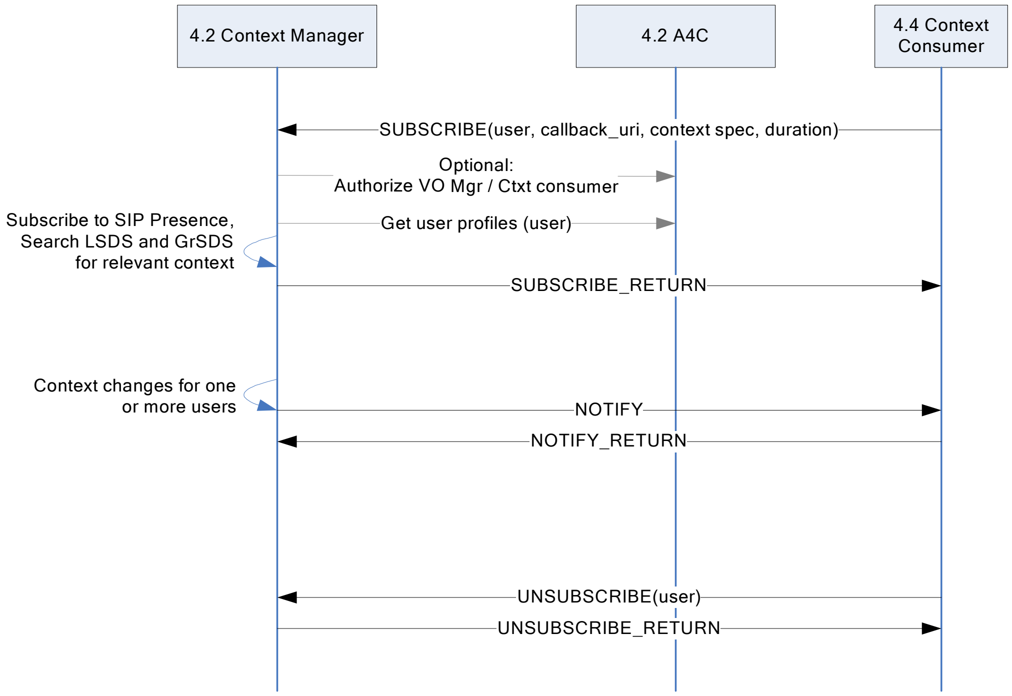

4.2 Context Manager
4.2 A4C
4.4 Context Consumer
SUBSCRIBE(user, callback_uri, context spec, duration)
Optional:
Authorize VO Mgr / Ctxt consumer
Get user profiles (user)
Subscribe to SIP Presence, Search LSDS and GrSDS for relevant context
SUBSCRIBE_RETURN
Context changes for one or more users
NOTIFY
NOTIFY_RETURN
UNSUBSCRIBE(user)
UNSUBSCRIBE_RETURN

Figure 33 Context Manager – Context consumer interface

Table 34 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
				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

Server: Component (Layer)	Client: Component (Layer)	Protocol	Method	Parameters		
				Name	Type	Explanation
				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
CM	Context Consumer (WP4.4)	SOAP	GETUSEREPR	User	AkogrimoID	UserID for the one you want EndpoitReference for (EPR)
				EPR	Return parameter EnpointReferenceType	Respose to GETUSEREPR

Table 35 Methods offered by the Context Consumer (callback interface)

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

Since D4.2.2 we have also included WS-Notification based access to Akogrimo context information. It is implemented using Globus Toolkit 4.0.2 Java WS Core. The WS-Notifications family of specifications provides a set of standard interfaces for realising notification patterns with Web Services, the interfaces remain the same compared to previously. For the client it works in the same way, but with the benefit that once changes happen to context (which is a resource the client is subscribing to), a notification will automatically be sent to the subscribing entity. A resource is addressed by an EndPointReference (EPR). That is accessed by a new Web Service method as shown in Figure 34.

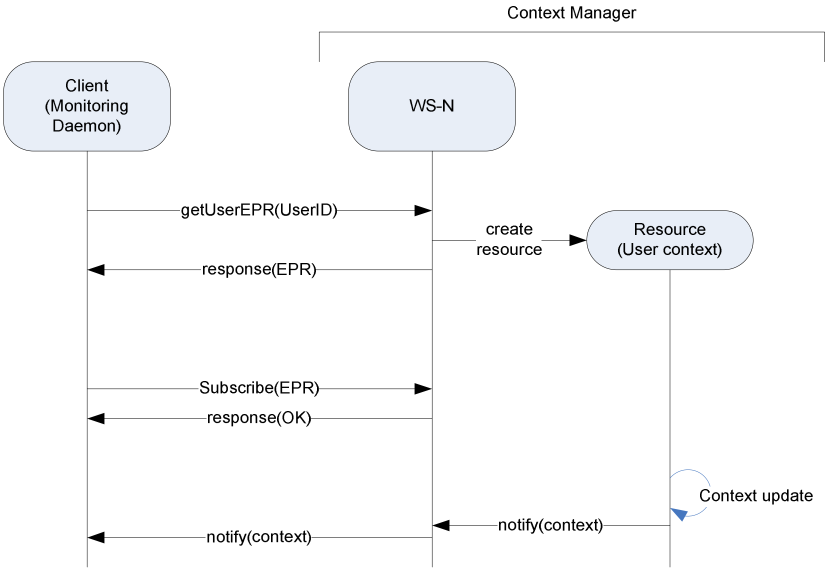


Figure 34 Basic functionality of WS-Notification

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 D4.2.2. Basic flows are shown in Figure 35.

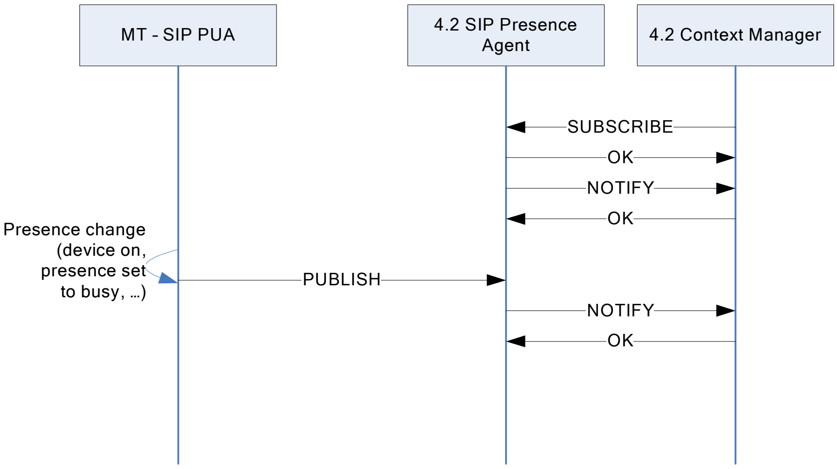


Figure 35 Context Manager – SIP Presence Agent interface

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, design and basic flows are described in 6.3.

5.2.5. Context Manager – RFID SW

The method offered by the Context Manager is shown in Table 36. Please refer to Section 5.4.5 for basic flows and application logic.

Table 36 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

5.2.6. CM – Bluetooth Context Harvester interface

The Bluetooth context Web Service running on the Context manager receives a list of discovered services and devices. If a GPS device is found a position is included. The akogrimoID is coupled with the Bluetooth MAC address for user device listings.

Server: Component (Layer)	Client: Component (Layer)	Purpose	Protocol
BT CA	CM	- Obtain Bluetooth devices near user - Obtain GPS position from GPS Bluetooth device	Java API

5.3. Concepts

In this section we describe some concepts that serve as basis for design and implementation of the context manager. This includes location handling, ontology and privacy/access control. The latter explains in part of how security is addressed in Akogrimo.

5.3.1. Location handling

In this section we will specifically be dealing with location mapping and the Akogrimo Location Model. Location mapping is the process of transferring one kind of location information to another. A specific need is the mapping of location from coordinates to area(s). The Akogrimo Location Model will be the definition of all available areas.

Location mapping is a special case of context mapping, where data is mapped from one representation to another. As another example, consider the time of day "14:00 pm" mapped to "afternoon". The data is more or less the same after the mapping, but it may have different significance for an application.

5.3.2. Location Mapping example

In Akogrimo scenarios there is a need for mapping the location coordinates to area descriptions. The area descriptions may be in the form of "Meeting Room, 2nd floor, Building A, Will Smith Street 5, Akogrimopolis, Akogrimoland", where we can abstract the level of the area if we don't need to know what room the user is in, but rather what part of the city he/she is in. For an ambulance driver in a medical emergency it will be sufficient at first to know the building a critical patient is in. When the ambulance gets closer to the precise location, it may be necessary to inform the driver what room the patient is in. The information of what building may be used to retrieve a building overview. That overview together with a room description tells the medical personnel where to locate the patient. All this may be derived from the raw data of a set of coordinates.

5.3.3. Akogrimo Location Model

To make demos more realistic we have looked into using a location model to represent coordinates. The model shows a fictional city called Akogrimopolis, divided in different areas. Area types are listed in Table 37.

Table 37 List of Area types

Large regions	Small regions	Building level	General
Continent Country Region County	Municipality Town TownArea	Building Floor Room	GeneralPolygon

We do not assume any specific restrictions for the layout of areas. Each area may contain one or more of the sub areas. Moreover, areas may be completely or partially overlapping. The GeneralPolygon is to identify areas that do not fit into any of the other categories (it can be an area of any size and form, most likely used for furniture). The corners for each area have defined coordinates. For a given coordinate we can then check what areas of different types the coordinate belongs to.

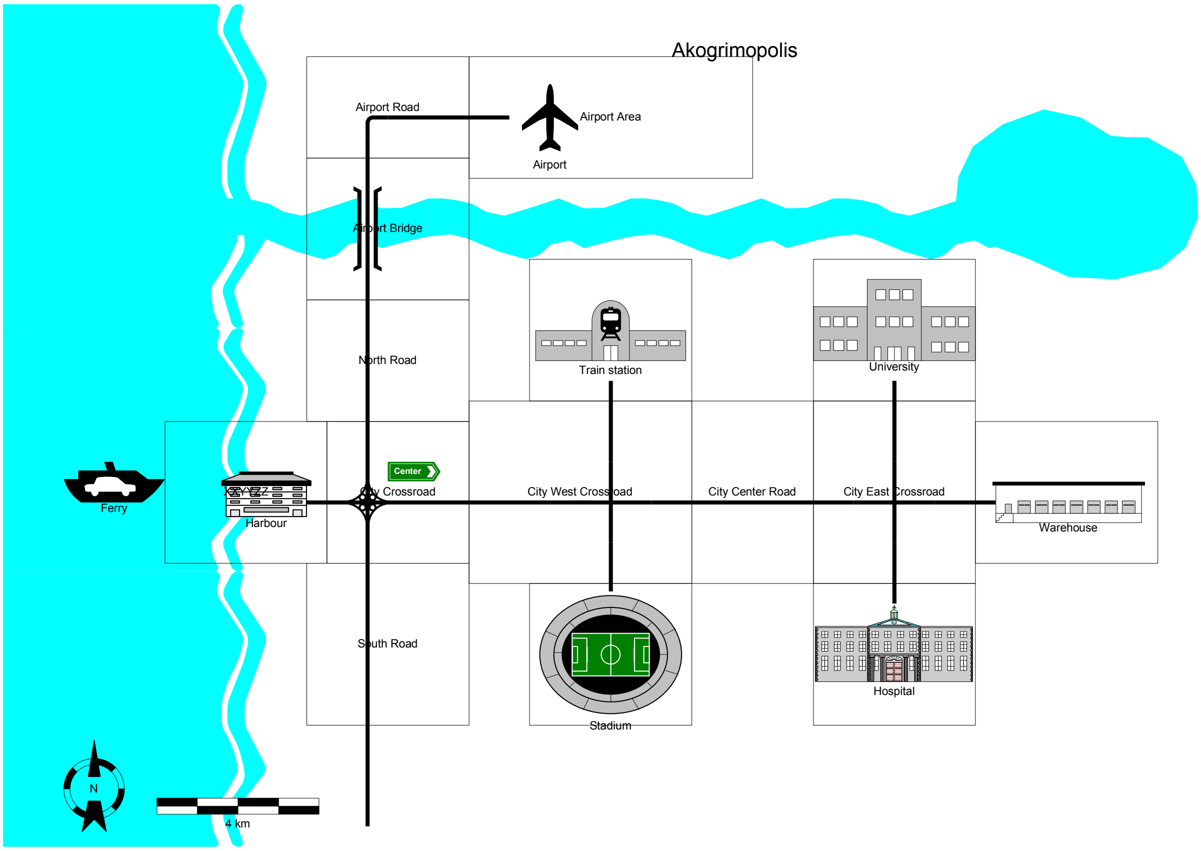


Figure 36 Akogrimopolis areas

Figure 36 shows how the fictive city of Akogrimopolis may be divided into different TownAreas.

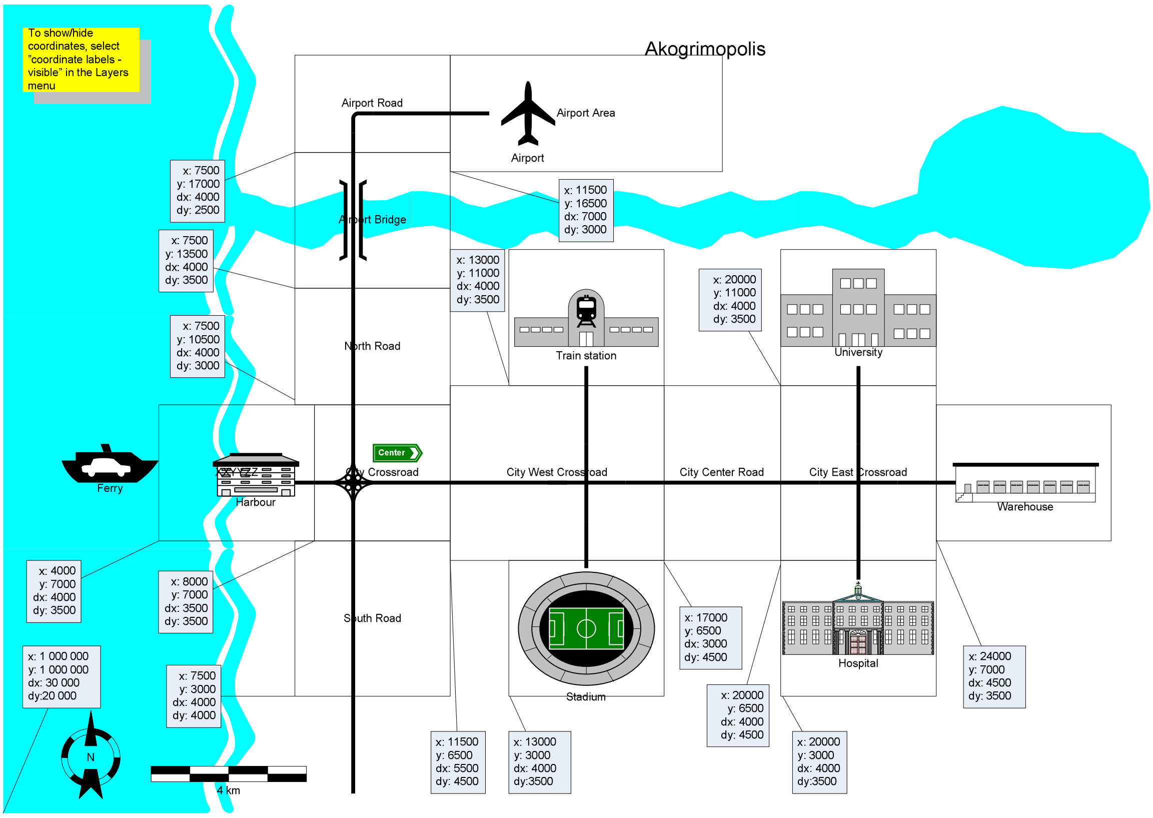


Figure 37 Akogrimopolis areas labelled

Figure 37 shows the same areas as in Figure 36, but with coordinates for the south west corner and values for how far the area stretches in north and east.

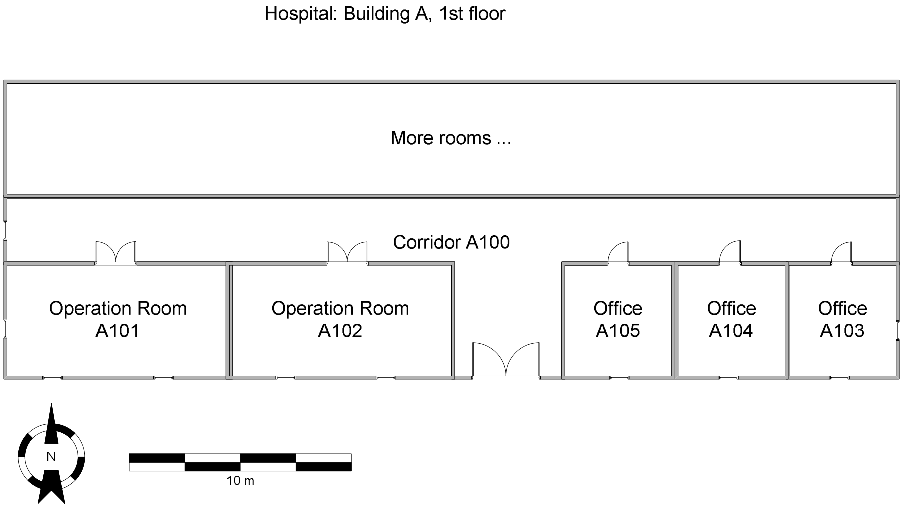


Figure 38 Hospital, Building A, 1st floor

Figure 38 shows a floor in a building belonging to the TownArea of the hospital of Akogrimopolis. All areas are listed in Table 38.

Table 38 Area name and type

Area name	Area type	Included in
Akogrimopolis	Town	-
Airport Area	TownArea	Akogrimopolis
Train Station	TownArea	Akogrimopolis
University	TownArea	Akogrimopolis
Warehouse	TownArea	Akogrimopolis
Hospital	TownArea	Akogrimopolis
Staium	TownArea	Akogrimopolis
Harbour	TownArea	Akogrimopolis
Airport Road	TownArea	Akogrimopolis
Airport Bridge	TownArea	Akogrimopolis
North Road	TownArea	Akogrimopolis
Crossroad	TownArea	Akogrimopolis
South Road	TownArea	Akogrimopolis
City West Crossroad	TownArea	Akogrimopolis
City Center Road	TownArea	Akogrimopolis
City East Crossroad	TownArea	Akogrimopolis
Hospital Building A	Building	Hospital
Hospital Building A 1st floor	Floor	Hospital Building A
A100	Room	Hospital Building A 1st floor
A101	Room	Hospital Building A 1st floor
A102	Room	Hospital Building A 1st floor
A103	Room	Hospital Building A 1st floor
A104	Room	Hospital Building A 1st floor
A105	Room	Hospital Building A 1st floor

5.3.3.1. CM – Location Model support

The location model is being defined in MySQL database that has spatial extensions. MySQL supports a subset of the geometry types defined by the Open Geospatial Consortium (OGC) in OpenGIS® Simple Features Specifications For SQL [OpenGIS]. Areas are defined with coordinates as shown in Figure 37. Given the location coordinates of a user, it is possible to query the database to retrieve the area(s) the person is within.

5.3.4. Semantics/Ontologies

Context information may be described in many different ways, e.g. depending on choice of terms, refinement and presentation of information. The following examples will illustrate this (here, "requestor" and "context consumer" are used interchangeably):

- Terms: When dealing with the property of presence, several expressions may be used. These are some of the thesaurus for the word presence: attendance, existence, and availability. In order for a requestor to identify that presence is the relevant piece of context information, there are mainly two alternatives:
    - The requestor must know exactly the terms used by the Context Manager, e.g. precisely which term is used for "presence"
    - The Context Manager can perform reasoning over the terms used by the requestor, and associate e.g. a request for "availability" (or similar) with "presence", which happens to be part of the provided context information. This will however require the Context Manager to understand and deal with thesaurus, and is considered as being too ambitious for the current project scope.
- Refinement and presentation of information: Location information may be presented at several levels: lower level such as coordinates (lat/lon), or more high level and user-friendly presentation such as area type (country, town,…), or descriptive (on a beach, in the office, …). The requestor need to indicate what representation of location information is needed

Semantics and ontologies may be used in different "steps". Each step provides more functionality, but will also add complexity and require more effort. This is depicted in Figure 39.

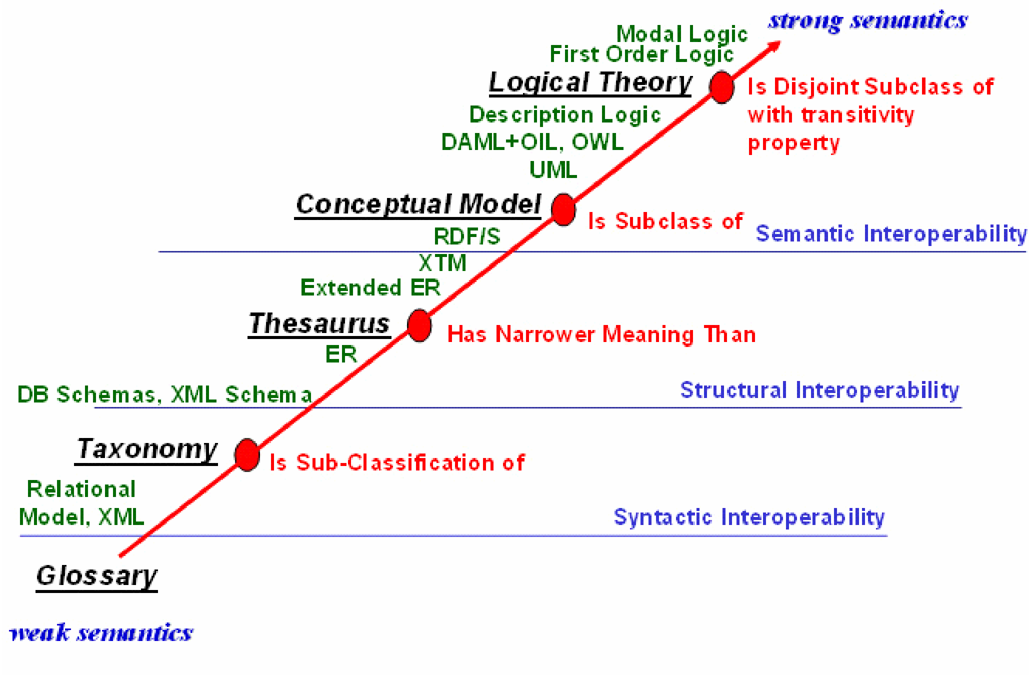


Figure 39 Ontology – functionality, complexity, and OWL technology

In context management for Akogrimo, the use of ontologies has immediate value for the following reasons:

1. It may be used to inform the context consumer of the possible selection of context information, i.e. form a common taxonomy to describe context.
2. If semantic reasoning is employed by CM, it may be able to handle requests that are not following the common context description.

In a first approach we shall address item 1 above. Item 2 will require more time and effort, and might be addressed at later stages.

5.3.4.1. A context ontology for Akogrimo

There are many ways to model context information, and several of these are available on the Internet [Wang04, Gu04, SIMLIE]. One might argue that reuse of one or several of these would be beneficial. However, after review of a selection of existing ontologies, as well as discussions with researchers in the field, we conclude that it is advantageous to establish a context ontology for Akogrimo from scratch. This is due to the fact that every project has its special requirements and properties, and adaptation of existing results often requires more effort than gain. The context ontology, which is presented in Figure 40, is to some extent inspired by Mostéfaoui et al [Mos03] and the SIMILE Location ontology [SIMLIE].

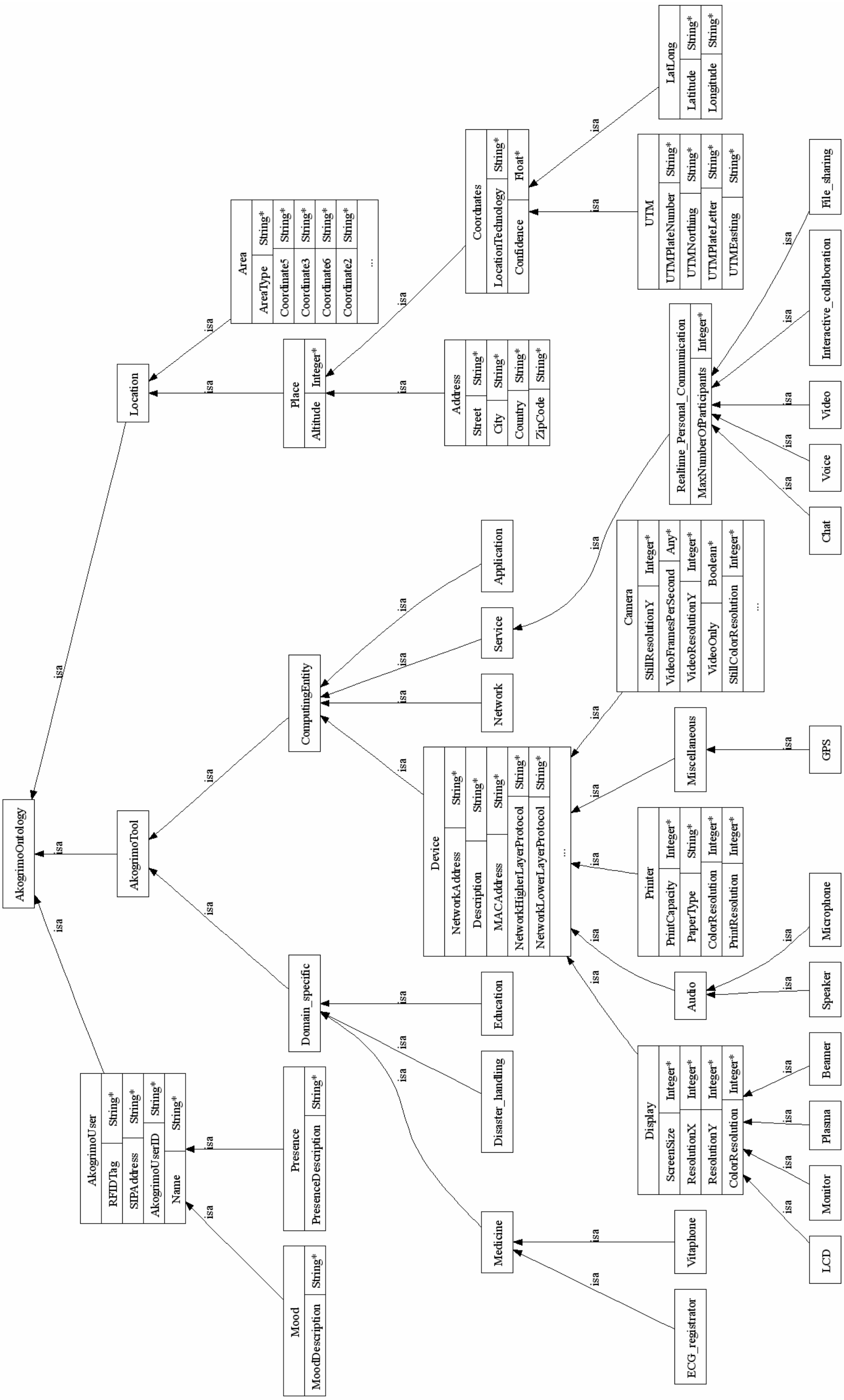


Figure 40 Context ontology for Akogrimo

5.3.5. Privacy and access control

This section describes privacy-related problems in context management, a general framework for addressing these problems, and a system design that can face some of the Akogrimo specific problems. Note that the design/realisation is fairly limited compared to the problem description. This is due to time and resource constraints in the project.

User Context is an important enabler for pervasive services in general, and in particular for the grid services developed within Akogrimo. However, as user context carries sensitive information it may violate privacy rights if abused. This section describes a proposal for privacy handling related to context management in Akogrimo. Basic context handling is as described in D4.2.2 [D4.2.2], and is depicted in Figure 41.

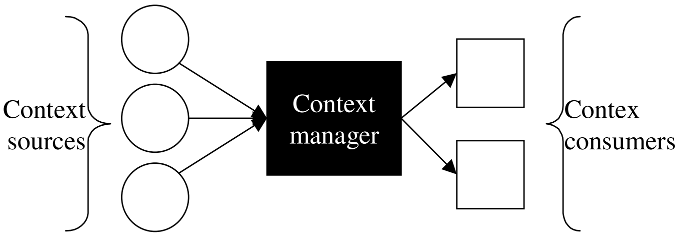


Figure 41 Basic infrastructure for context management

5.3.5.1. Problem statement

The use of context is a double-edged sword: On the one hand, large volumes of reliable user information will improve the impact of context-aware services and hence be beneficial for users. On the other hand, users will refrain from spreading sensitive information that may be used for surveillance purposes (references to recent cases: surveillance for anti-terror purposes in USA, GB and some other EU countries, US phone service: Parents watch teenager location based on GPS+mobile phone, EU requires operators to store users activities based on electronic communications).

IM clients such as Skype, Microsoft Messenger, Jabber, and ICQ feature presence management. Regarding privacy these are usually based on a simple principle: The user allows or denies other users to see his/her presence. In systems handling rich presence/context(reference), the trend is the same: Allow others to see all or nothing.

We believe there is need for privacy management that opens for differentiated access to view user context. This differentiation could be based on

- Who the requestor is: A user would probably opt to give family members access to more information than distant colleagues at work.
- Level of detail: It would be convenient to provide trusted parties with more detailed information than peripheral parties. E.g. presence and location information can be provided in a precise or inaccurate manner.
- Situation of the related user: A user could be interested in restricting context distribution depending on the current situation, i.e. his/her current context. Assume e.g. a taxi driver who has to display his location at the taxi control centre while at work. When he/she is off work, location should no longer be shown at the control centre.

In the same manner as for distribution of context information, one could be interested in restricting collection of user data. Referring to Figure 41, even if an individual is assured that collected data is not forwarded without permission, the level of privacy is increased if it is collected only when necessary.

5.3.5.2. General design

Structure

The main elements in the described privacy handling are

Table 39 Main elements in privacy handling

Context consumer	A user or entity that queries context information about other entities
Context provider	An entity that knows a given context and communicates this information to a management system
Context owner	The user or entity that has legal rights to the given user context□
User context	Information describing the situation of a user
Context source	The physical or logical source of information (sensors, IM clients, …)
Privacy administrator	Third party acting on behalf of the context owner to manage access to context

Figure 42 gives an overview of elements and associations. Note that "Provide Restriction" and "Request Restriction" are association classes.

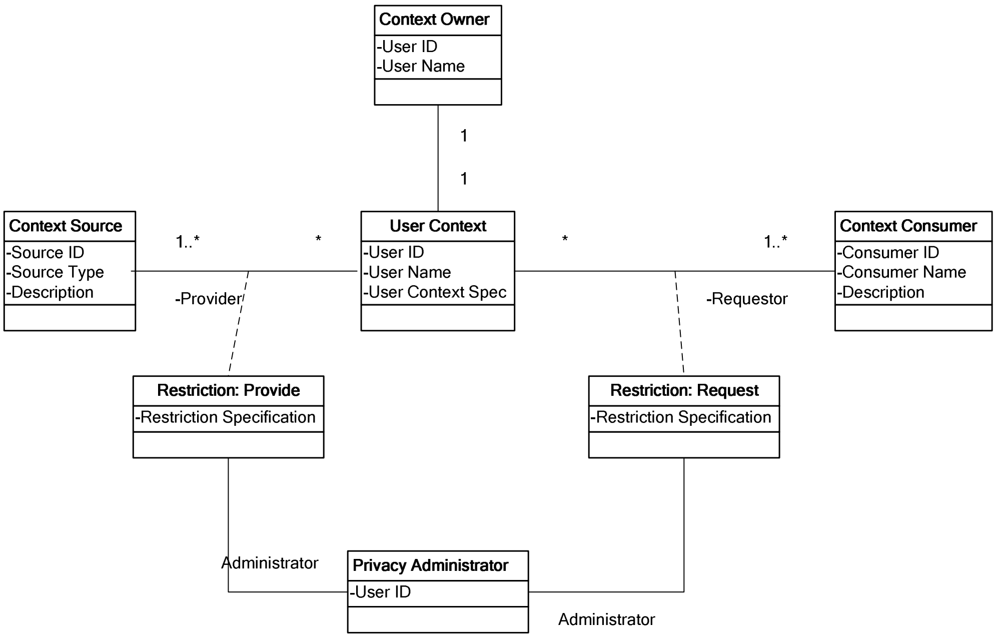


Figure 42 Main elements in privacy handling

Functionality

1. Specify privacy policy. The Privacy Administrator (or the context owner him/herself) specifies a set of restrictions for provisioning and request of context information.
2. Enforce privacy policy. Under operation, restrictions are consulted when context is provided or requested for.

5.3.5.3. Design applied in Akogrimo

In Akogrimo, only a subset of the structure and functionality described in Section 5.3.5.2 is applied. In particular:

- Only the relation context consumer – context owner is addressed (i.e. not context owner – context source)

- The privacy policy is limited to indicating which context consumer may request context information for which context owner.
- Specification of privacy policy is performed offline. A4C stores privacy restrictions and thus it contains a registry indicating which context consumer (identified as application@domain) may request context information for what user (identified as user@domain).
- Enforcement of privacy policy is carried out by CM and A4C
  - Takes place for every context request CM receives from context consumer – MD (WP4.4 Montoring Daemon)
  - CM authenticates MD
  - CM queries A4C to authorize the context request
- Authentication of context consumer based on use of certificates. Management of certificates is described in Section 3.4.4

Basic flow in privacy handling is shown in Figure 43. Note that if authentication or authorization of context manager fails, SUBSCRIBE_RETURN(NOK) will be sent to the Context Consumer.

Authentication is carried out based on public keys, and works as follows: The context consumer will send SUBSCRIBE with the following parameters:

1. *certificate*
2. *applicationID, userID, callback_uri, context spec, duration*
3. *K^-(H(applicationID, userID, callback_uri, context spec, duration))*[6].

The certificate will contain the public key K^+. CM will extract parameters in item 2 above and apply the Hash function H, then compare the result to K^+(parameter 3 in above list). Thus CM can verify the validity of the context consumer's identity and the integrity of the message in parameter 2.

Authorization is carried out as follows: During offline setup, the Privacy Administrator (sometimes the user him/herself) sets permissions for access to context. This permission is stored in A4C as request restrictions. When CM has authenticated a context consumer, it will ask A4C to verify that this specific consumer (applicationID = application@domain) can request context for a specific user (userID = user@domain). If the outcome is positive, CM will proceed to ask A4C for user profiles.

[6] Here K^-(msg) means "message *msg* encrypted with the private key K^-", and H(msg) means "Hash function applied to message *msg*".

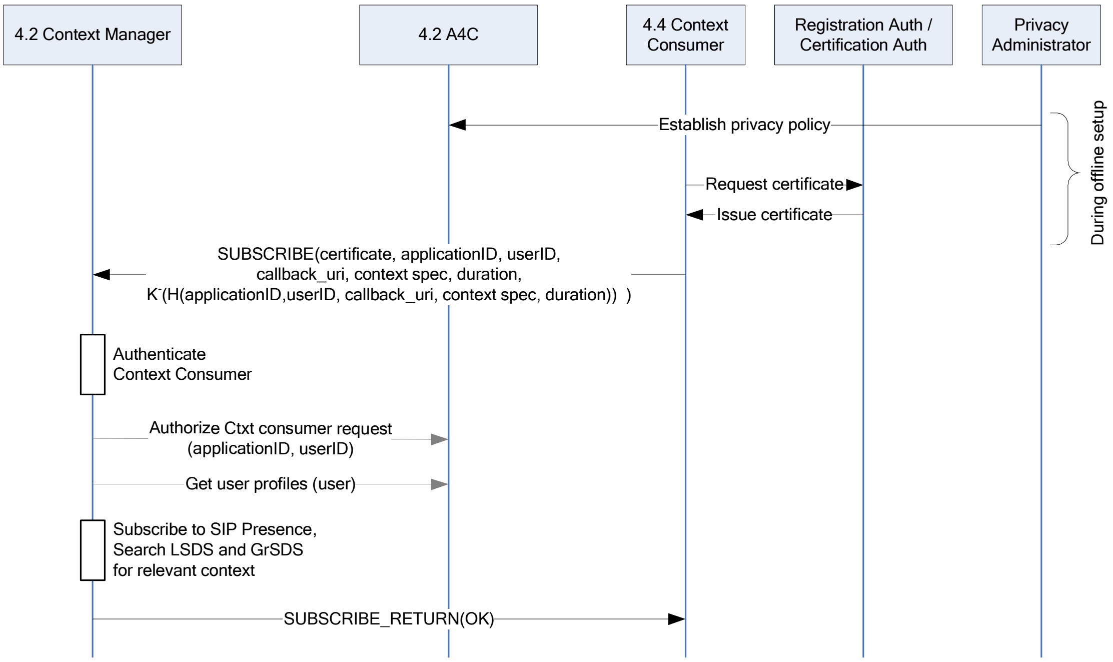

4.2 Context Manager
4.2 A4C
4.4 Context Consumer
Registration Auth / Certification Auth
Privacy Administrator
Establish privacy policy
Request certificate
Issue certificate
During offline setup
SUBSCRIBE(certificate, applicationID, userID, callback_uri, context spec, duration, K^-(H(applicationID,userID, callback_uri, context spec, duration)))
Authenticate Context Consumer
Authorize Ctxt consumer request (applicationID, userID)
Get user profiles (user)
Subscribe to SIP Presence, Search LSDS and GrSDS for relevant context
SUBSCRIBE_RETURN(OK)

Figure 43 Privacy handling – basic flow.

5.4. Design & Implementation

This section describes the various parts of the Context Manager. The core of the Context Manager includes the engine, databases and gateways. Applications that are developed to provide context are Bluetooth Context Harvester and RFID software. Context Viewer is an application that subscribes to context in order to view context information, usable for various tests.

- Context Engine
- Databases
  - Context
  - Location Model
- Context Gateways
  - SIP
  - RFID
  - SLP
- Bluetooth Context Aquirer
- RFID software
- Context Viewer

5.4.1. Context Engine

Context engine is the main component in the Context Manager. Its main functional parts are shown in Figure 44. 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 described here represents a simple approach to context handling; when context for a user is requested, the context engine will gather all available context information for that user.

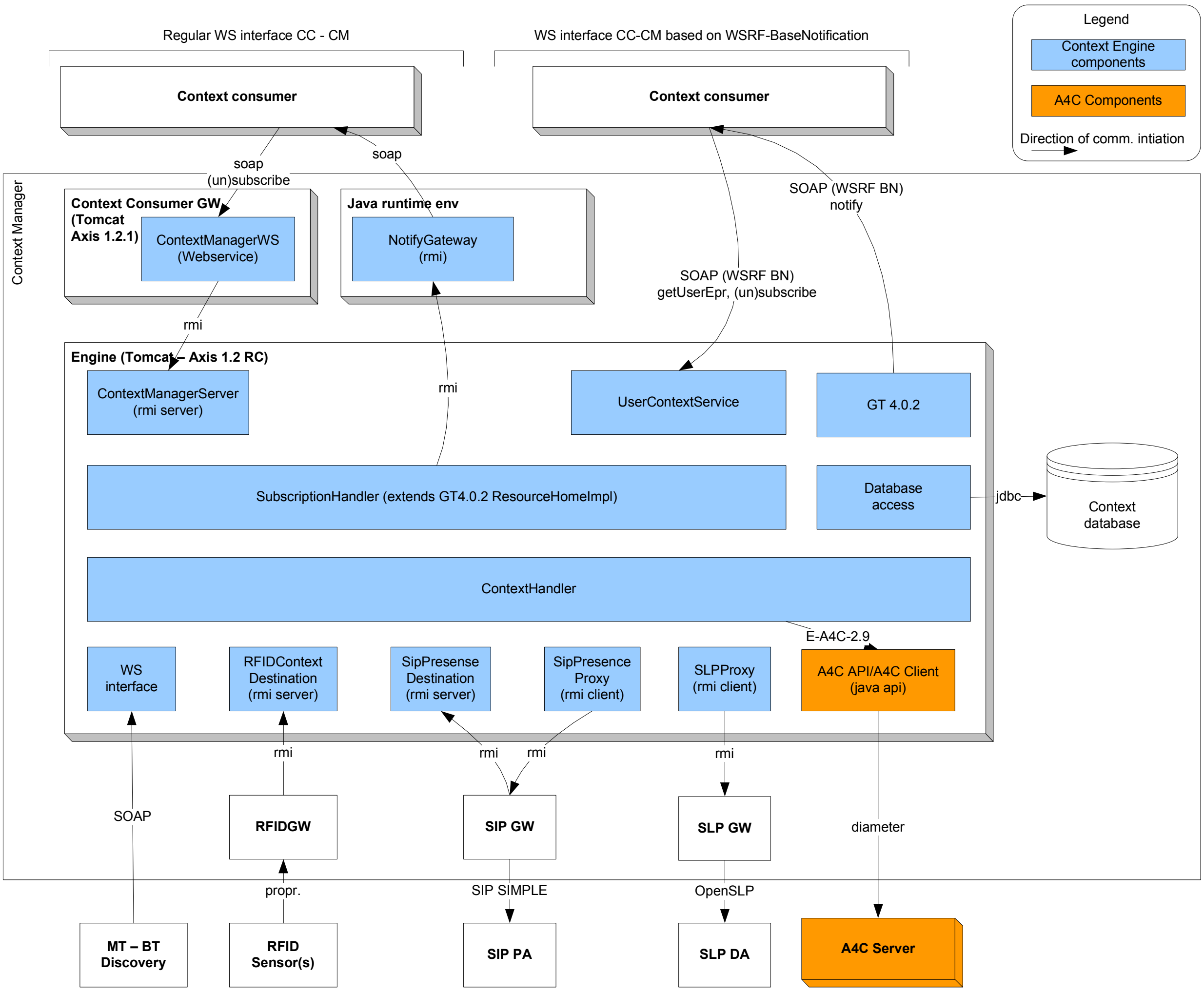

Regular WS interface CC - CM
WS interface CC-CM based on WSRF-BaseNotification
Legend
Context Engine components
A4C Components
Direction of comm. intiation
Context consumer
Context consumer
soap
(un)subscribe
soap
SOAP (WSRF BN)
notify
Context Manager
Context Consumer GW (Tomcat Axis 1.2.1)
ContextManagerWS (Webservice)
Java runtime env
NotifyGateway (rmi)
SOAP (WSRF BN)
getUserEpr, (un)subscribe
rmi
Engine (Tomcat – Axis 1.2 RC)
ContextManagerServer (rmi server)
rmi
UserContextService
GT 4.0.2
SubscriptionHandler (extends GT4.0.2 ResourceHomeImpl)
Database access
jdbc
Context database
ContextHandler
E-A4C-2.9
WS interface
RFIDContext Destination (rmi server)
SipPresense Destination (rmi server)
SipPresence Proxy (rmi client)
SLPProxy (rmi client)
A4C API/A4C Client (java api)
rmi
rmi
rmi
rmi
SOAP
RFIDGW
SIP GW
SLP GW
diameter
propr.
SIP SIMPLE
OpenSLP
MT – BT Discovery
RFID Sensor(s)
SIP PA
SLP DA
A4C Server

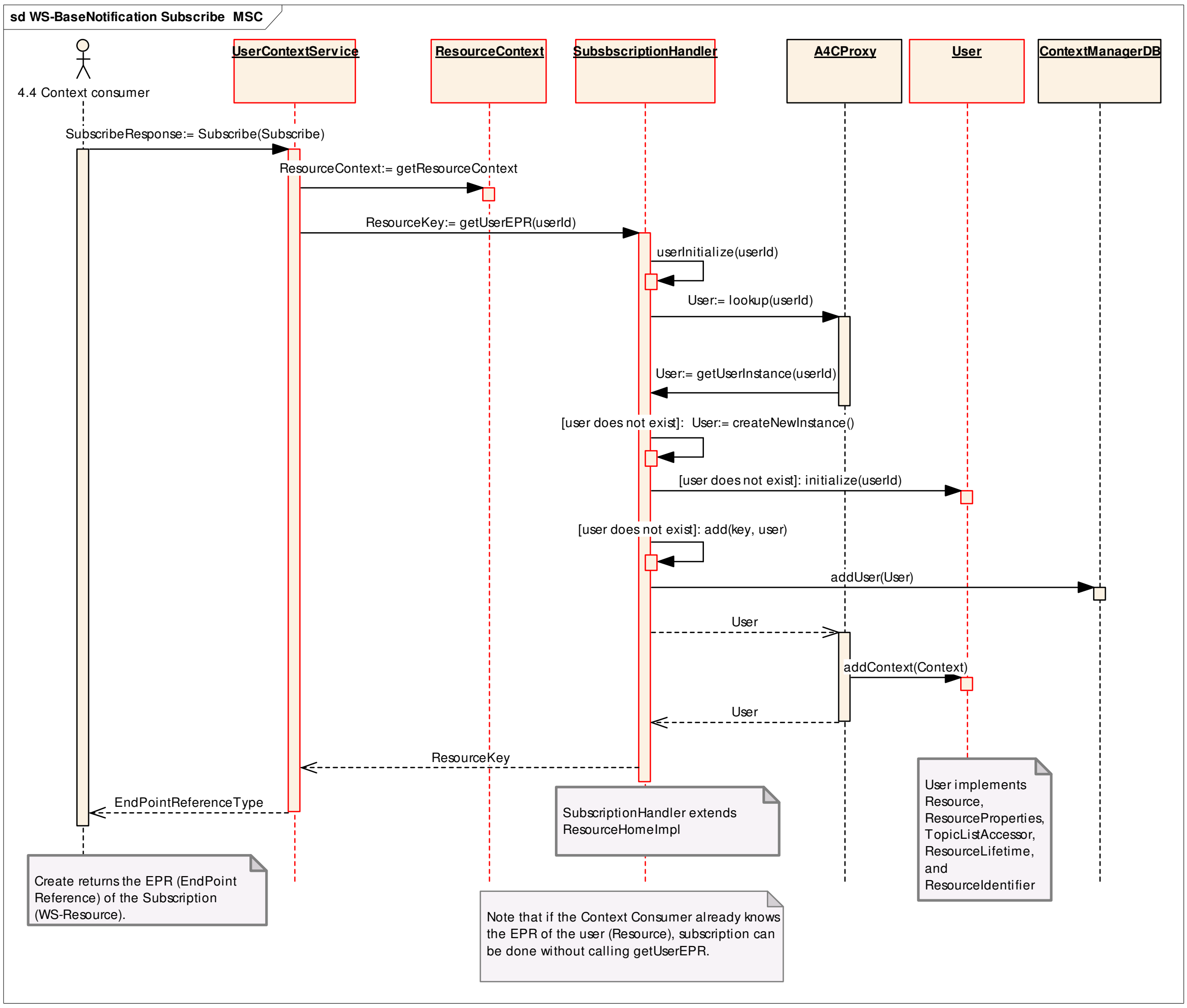
Figure 44 Context Engine - design

Subscriptions from context consumers

When a subscription is received, the following is performed (see Figure 45 for the WS BaseNotification method and Figure 46 for the regular WS Callback method):

- 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

The context consumer has to know the EPR of the Subscription for the WS BaseNotification method. A function to get the EPR of an Akogrimo user is shown in Figure 47.


sd WS-BaseNotification Subscribe MSC
4.4 Context consumer
UserContextService
ResourceContext
SubsbscriptionHandler
A4CProxy
User
ContextManagerDB
SubscribeResponse:= Subscribe(Subscribe)
ResourceContext:= getResourceContext
ResourceKey:= getUserEPR(userId)
userInitialize(userId)
User:= lookup(userId)
User:= getUserInstance(userId)
[user does not exist]: User:= createNewInstance()
[user does not exist]: initialize(userId)
[user does not exist]: add(key, user)
addUser(User)
User
addContext(Context)
User
ResourceKey
EndPointReferenceType

Create returns the EPR (EndPoint Reference) of the Subscription (WS-Resource).

SubscriptionHandler extends ResourceHomeImpl

Note that if the Context Consumer already knows the EPR of the user (Resource), subscription can be done without calling getUserEPR.

User implements Resource, ResourceProperties, TopicListAccessor, ResourceLifetime, and ResourceIdentifier

Figure 45 WS-Notification Subscribe sequence diagram

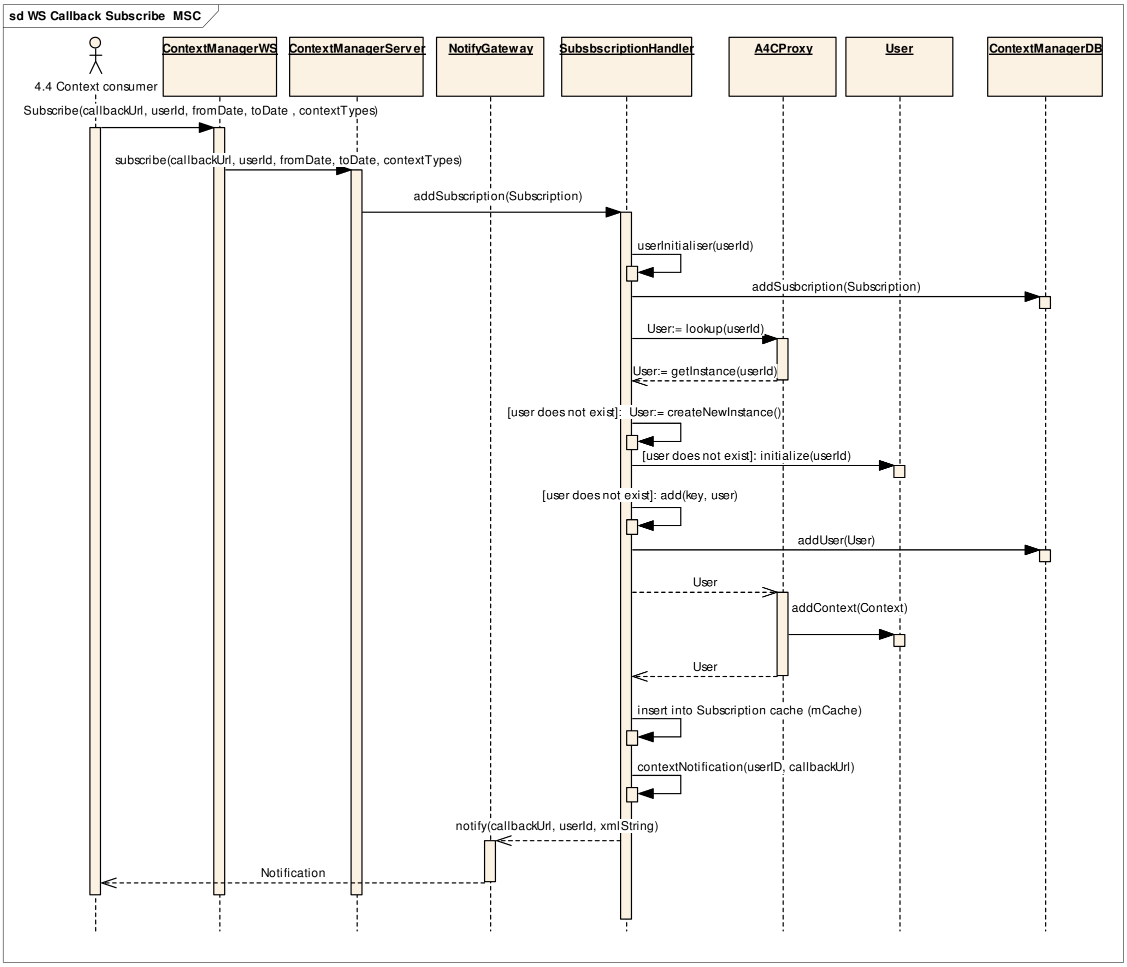

sd WS Callback Subscribe MSC
4.4 Context consumer
ContextManagerWS
ContextManagerServer
NotifyGateway
SubsbscriptionHandler
A4CProxy
User
ContextManagerDB
Subscribe(callbackUrl, userId, fromDate, toDate , contextTypes)
subscribe(callbackUrl, userId, fromDate, toDate, contextTypes)
addSubscription(Subscription)
userInitialiser(userId)
addSusbcription(Subscription)
User:= lookup(userId)
User:= getInstance(userId)
[user does not exist]: User:= createNewInstance()
[user does not exist]: initialize(userId)
[user does not exist]: add(key, user)
addUser(User)
User
addContext(Context)
User
insert into Subscription cache (mCache)
contextNotification(userID, callbackUrl)
notify(callbackUrl, userId, xmlString)
Notification

Figure 46 WS Callback subscribe sequence diagram

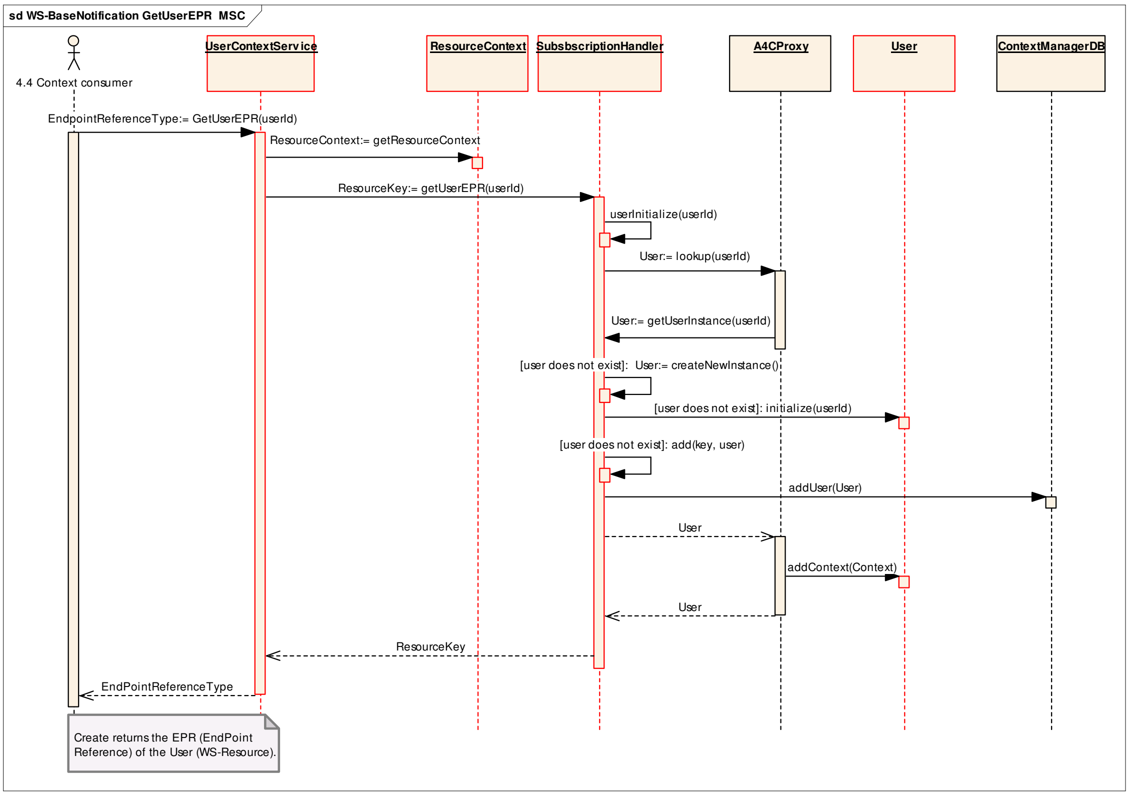

sd WS-BaseNotification GetUserEPR MSC
4.4 Context consumer
UserContextService
ResourceContext
SubsbscriptionHandler
A4CProxy
User
ContextManagerDB
EndpointReferenceType:= GetUserEPR(userId)
ResourceContext:= getResourceContext
ResourceKey:= getUserEPR(userId)
userInitialize(userId)
User:= lookup(userId)
User:= getUserInstance(userId)
[user does not exist]: User:= createNewInstance()
[user does not exist]: initialize(userId)
[user does not exist]: add(key, user)
addUser(User)
User
addContext(Context)
User
ResourceKey
EndPointReferenceType
Create returns the EPR (EndPoint Reference) of the User (WS-Resource).

Figure 47 WS-Notifcation get EPR for User sequence diagram

Receiving context

Reception of context updates triggers the following behaviour (see Figure 48 and Figure 49):

- 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 has requested 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.
- Notifications are sent to subscribers on both WS callback and WS-Notification interfaces.

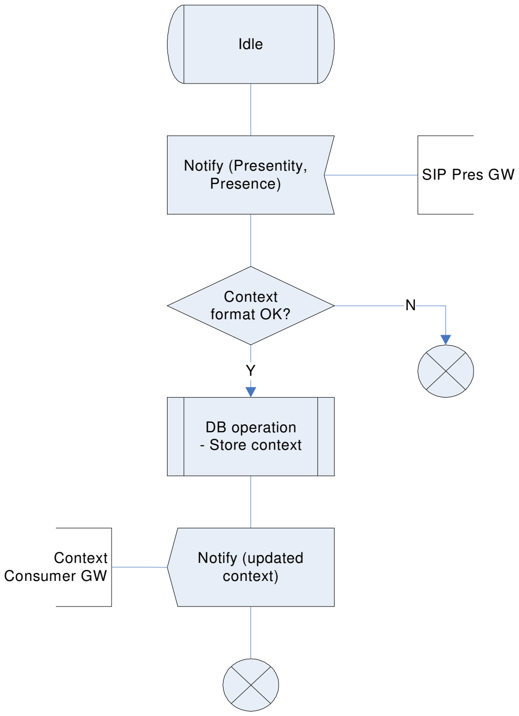
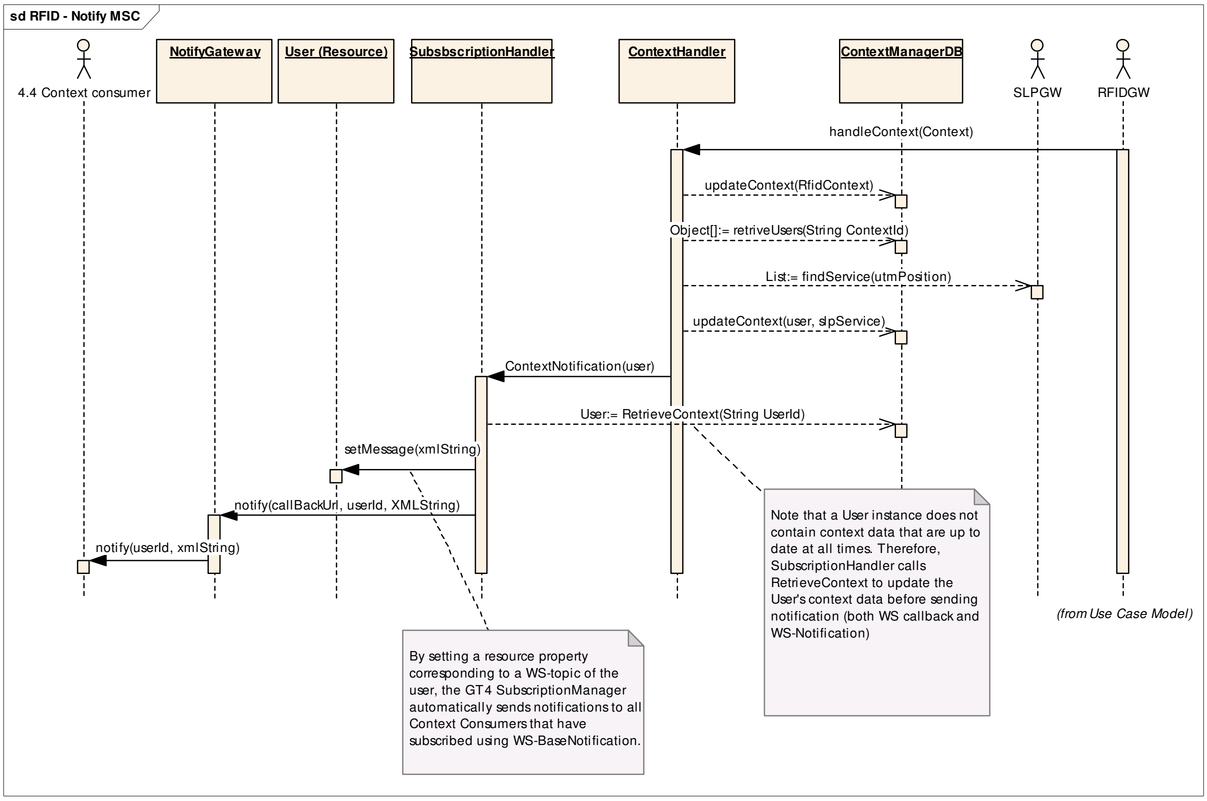


Figure 48 Updated SIP presence


sd RFID - Notify MSC
4.4 Context consumer
NotifyGateway
User (Resource)
SubsbscriptionHandler
ContextHandler
ContextManagerDB
SLPGW
RFIDGW
handleContext(Context)
updateContext(RfidContext)
Object[]:= retriveUsers(String ContextId)
List:= findService(utmPosition)
updateContext(user, slpService)
ContextNotification(user)
User:= RetrieveContext(String UserId)
setMessage(xmlString)
notify(callBackUrl, userId, XMLString)
notify(userId, xmlString)
(from Use Case Model)

Note that a User instance does not contain context data that are up to date at all times. Therefore, SubscriptionHandler calls RetrieveContext to update the User's context data before sending notification (both WS callback and WS-Notification)

By setting a resource property corresponding to a WS-topic of the user, the GT4 SubscriptionManager automatically sends notifications to all Context Consumers that have subscribed using WS-BaseNotification.

Figure 49 Notification triggered by RFID update

The context engine is implemented in Java. The main components are shown in Figure 44. 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

5.4.2. Databases

This section shows the design of the databases used in the Context Manager.

5.4.2.1. Context Database

User context consist of

- Presence information in SIP SIMPLE format
- User location from both RFID technology or Bluetooth GPS device
- Device availability found with SLP and UAProf

The database is shown in Figure 50. It is centered around context where context for each user can be of different types. The most important attributes and fields are described in Table 40,

Table 41, Table 42 and Table 43. The last tables are emphasized as they are new in the design.

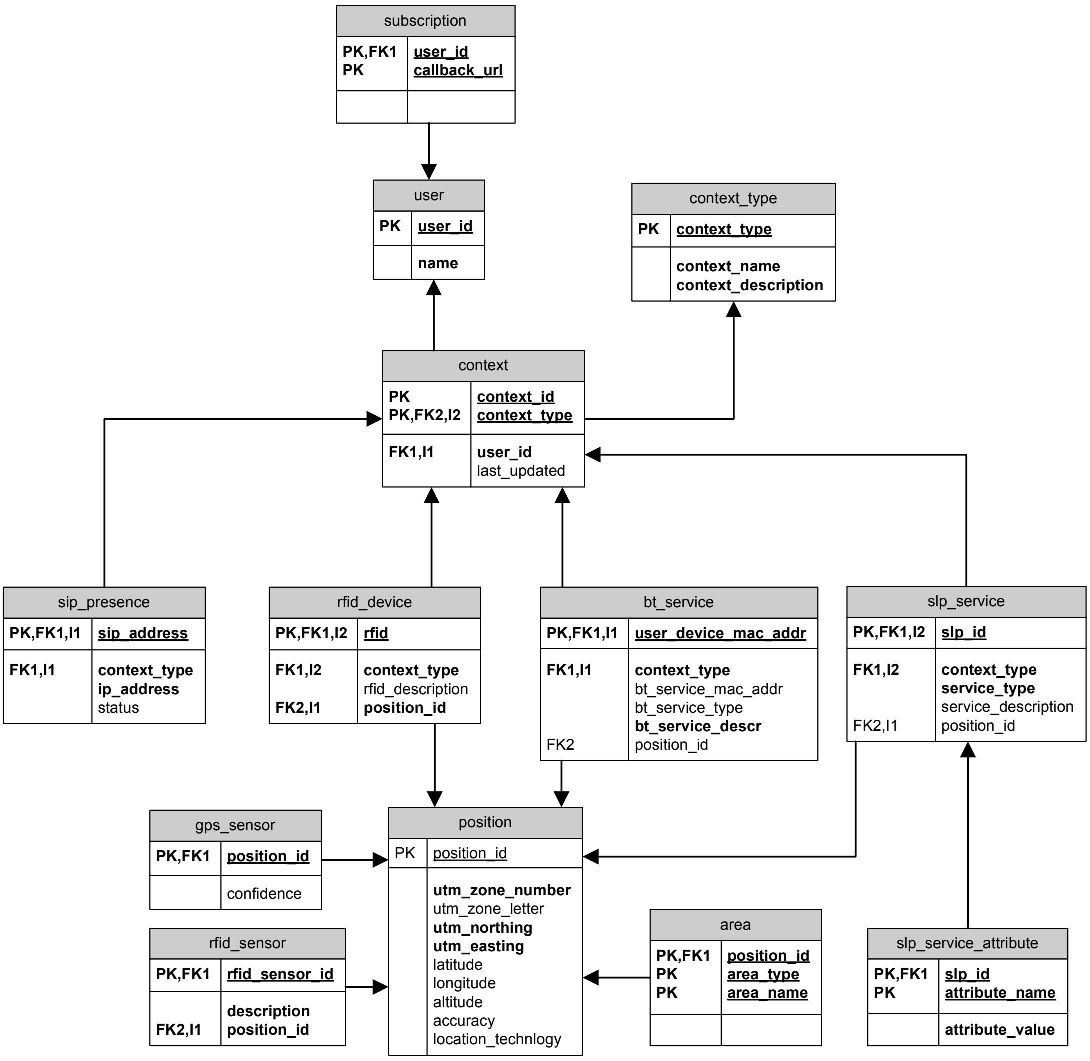


Figure 50 Context Database

Table 40 Main tables and fields in the Context Database

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
	context_type	from table context_type
context_type	context_type	1 = SIP Presence 2 = RFID 3 = SLP 4 = BT service
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.
	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>	Coordinates and spatial location, Table 44
rfid_sensor		Describes location of RFID sensors

Table 41 Bluetooth Harvester related fields

Field	Description
bt_id_mac	MAC address of the device that performs BlueTooth service discovery. This needs to be the user's personal device
bt_service_mac_addr	MAC address of the machine hosting a BT service
bt_service_description	Service description, e.g. "Headset", "SerialPort", and "Fax"
context_type	Bluetooth is type 4
bt_service_uuid	Universal(ly) Unique Identifier for the BT service. UUID is standardized: "Headset" = 0x1108, "SerialPort" = 0x1101, and "Fax" = 0x1111
position_id	Relates to a position if one of the BT services reports position from GPS

Table 42 Position with extended location data from Location Model Database

Field	Description
position_id	Relates to a position in table "position"
area_type	Type of area as defined in Table 44
area_name	Name of the area, see Table 44

Table 43 Bluetooth GPS fields

Field	Description
position_id	Relates to a position in table "position"
confidence	Used to indicate poor satellite connection

5.4.2.2. Location Model database

The location model database is a single lookup table which can be individually maintained parallel to the Context Database. It is not necessarily usable only for the Context Manager, but may have future usage for other location oriented applications.

Table 44 Fields in Location Model Database

Field	Description
area_id	Unique id for an area
area_name	The name of the area (not unique)
area_level	The level of the area: Continent Country Region County Municipality Town TownArea Building Floor Room GeneralPolygon
area_borders	GEOMETRY object defining the borders of the area

It is also necessary to mention that a procedure has been designed to automatically update locations with the new information in the Context Database. This is done in the database, and not in the application.

5.4.3. Context Gateways

This section describes the various context gateways that serve as entry points for context data for the Context Engine.

5.4.3.1. SLP

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), that 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.

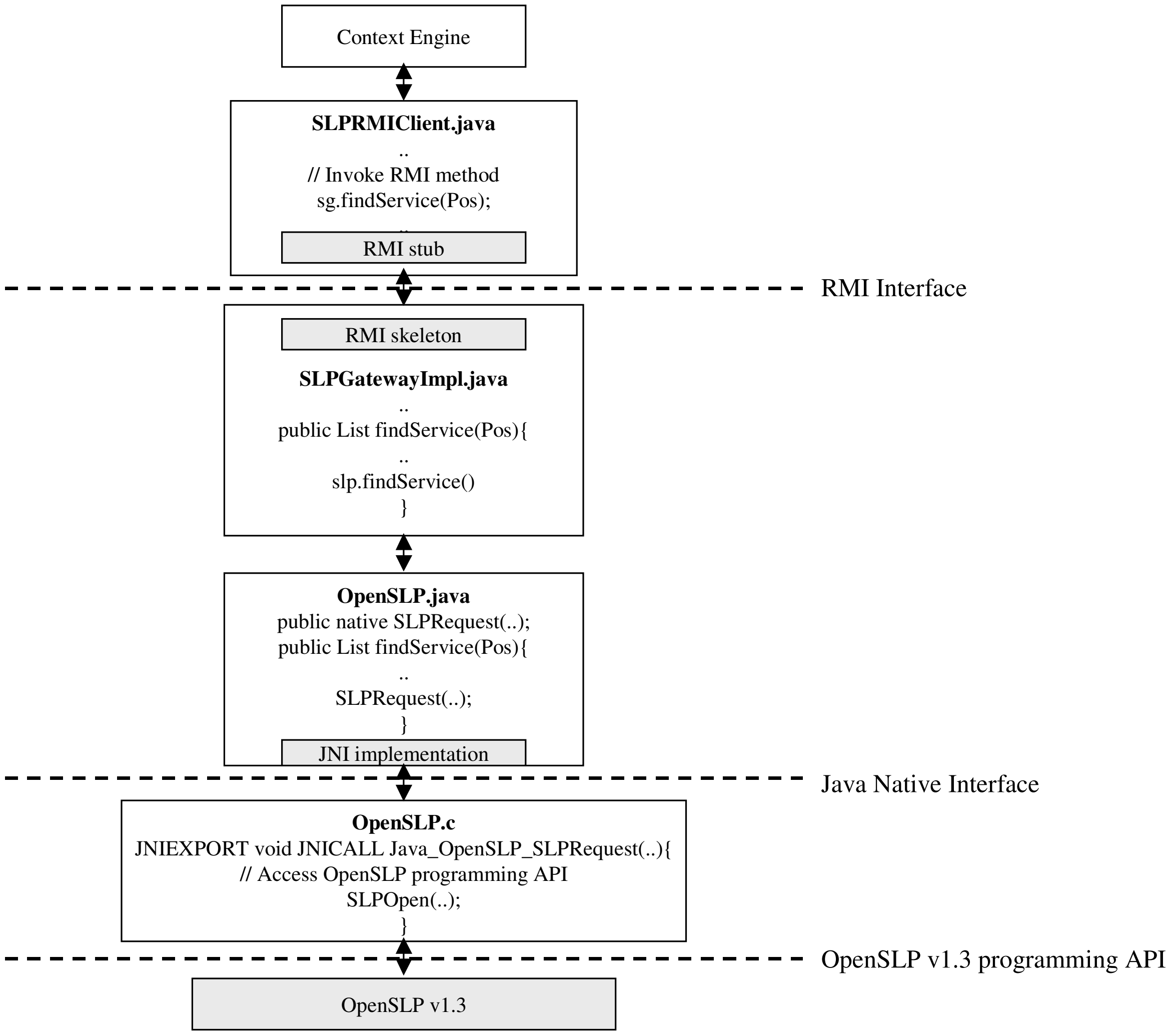


Figure 51 SLP Gateway design

The design of the SLPGateway is illustrated in Figure 51. 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 52 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.

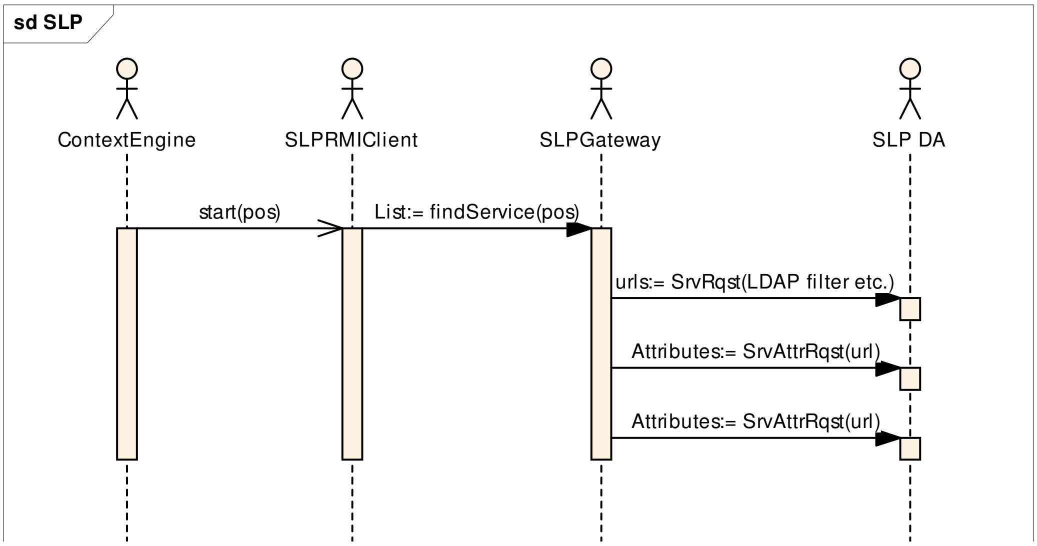

sd SLP
ContextEngine
SLPRMIClient
SLPGateway
SLP DA
start(pos)
List:= findService(pos)
urls:= SrvRqst(LDAP filter etc.)
Attributes:= SrvAttrRqst(url)
Attributes:= SrvAttrRqst(url)

Figure 52 Context Manager – SLP Directory Agent interface

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 result 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.

5.4.3.2. RFID

The RFID Gateway forwards the RFID readings via RMI interfaces. In addition to receive and transmit tags and locations it also provides logging. See section 5.4.5 for more information about RFID.

5.4.3.3. SIP

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 54 and Figure 55.

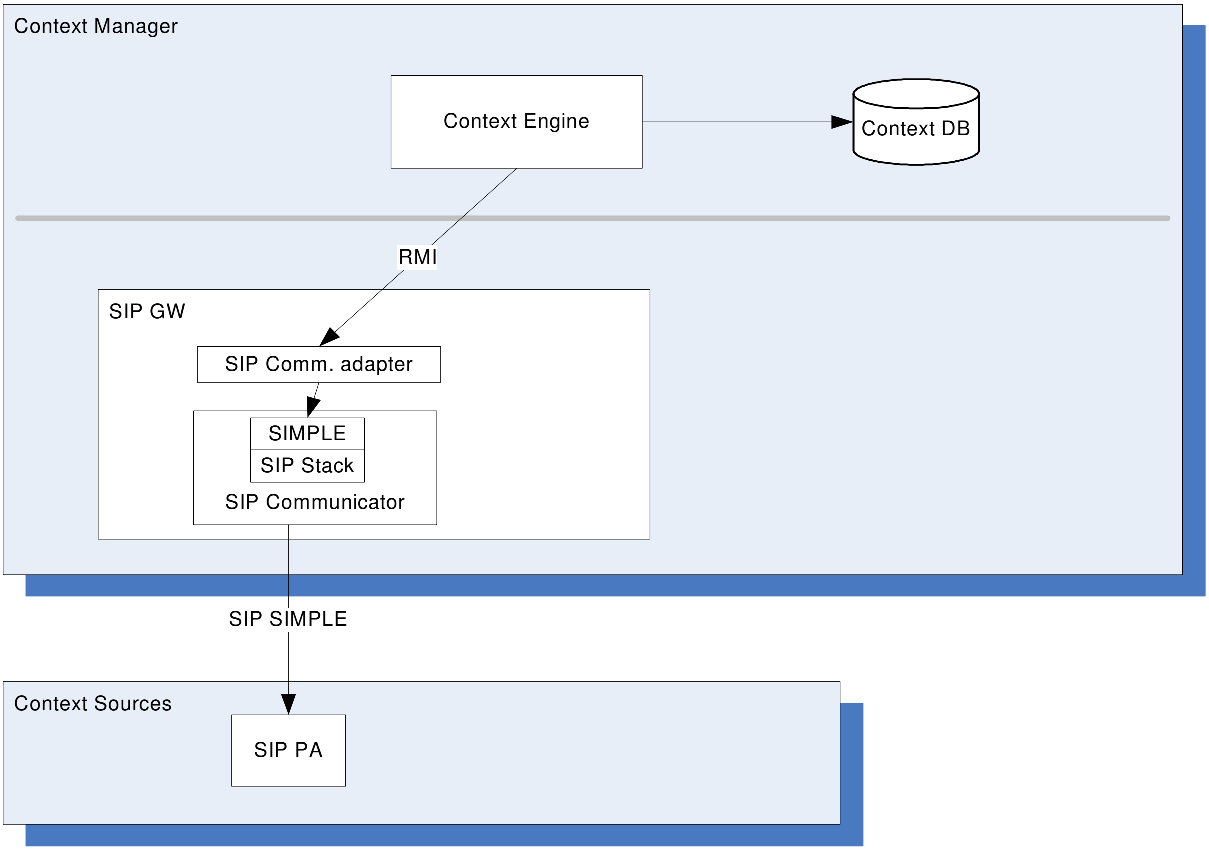


Figure 53 Structure of SIP Presence GW

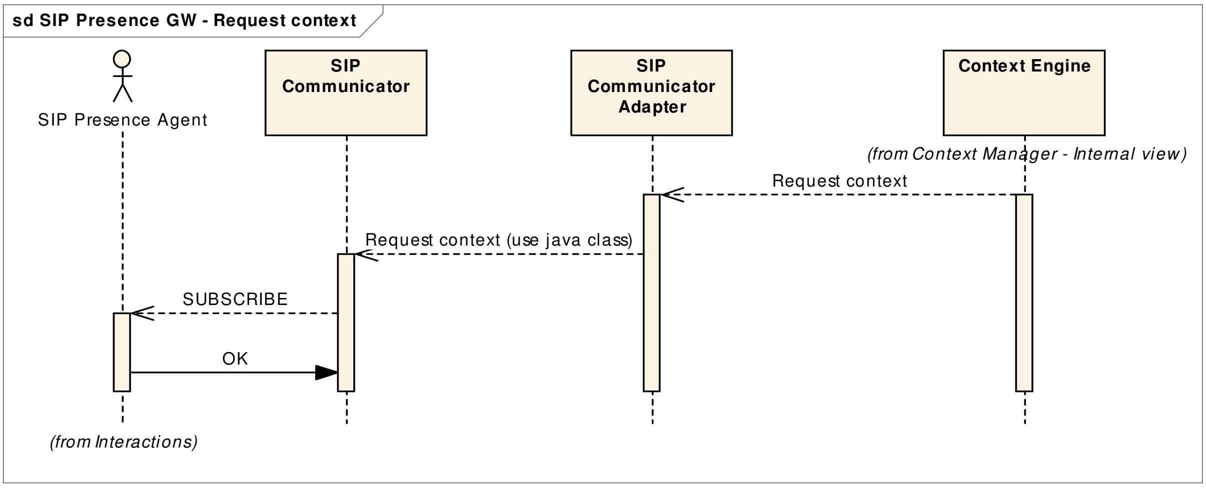


Figure 54 Basic flow – requesting context through SIP Presence GW

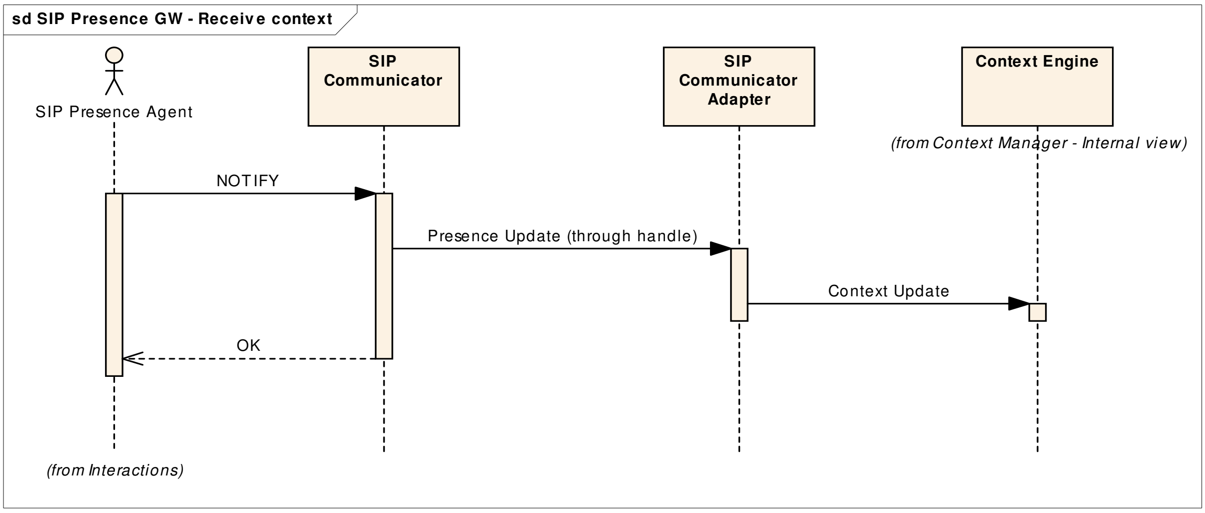


Figure 55 Basic flow – context received from SIP PA

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).

5.4.3.4. 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 44.

- 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 44**):

- 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.

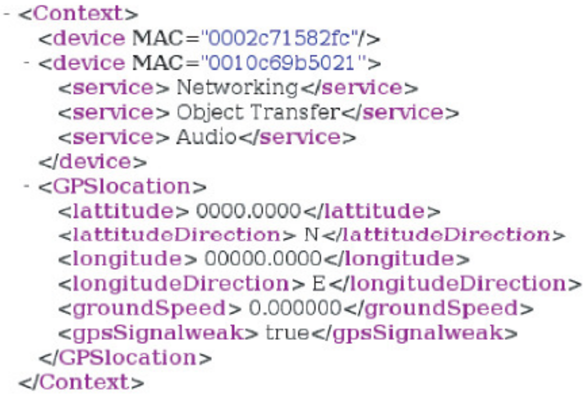
The Context consumer GW is implemented in java. The main components are the following:

- 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.

5.4.3.5. *Context Harvester Web Service at Context Manager*

This web services at the Context Manager will receive context updates from the Context Harvester, see in Figure 58.

Context Manager (i.e. the Web service at CM) will receive SOAP messages with a body containing a list of discovered services. This list may, if a GPS device was found, contain a position specification on latitude/longitude-format. An example is shown in Figure 56. Location values are set to zero here because the GPS device had no satellite contact since it was switched on. This is also indicated with the *gpsSignalweak*-variable set to value *true.*



```
- <Context>
    <device MAC="0002c71582fc"/>
  - <device MAC="0010c69b5021">
      <service> Networking</service>
      <service> Object Transfer</service>
      <service> Audio</service>
    </device>
  - <GPSlocation>
      <lattitude> 0000.0000</lattitude>
      <lattitudeDirection> N</lattitudeDirection>
      <longitude> 00000.0000</longitude>
      <longitudeDirection> E</longitudeDirection>
      <groundSpeed> 0.000000</groundSpeed>
      <gpsSignalweak> true</gpsSignalweak>
    </GPSlocation>
  </Context>
```

Figure 56 Computational and location context - example

In order to relate this information to the correct user, CM uses a link between *user ID* and *MAC address* for the device used during Bluetooth service discovery. This association user ID – MAC address is obtained from A4C along with information about SIP address and RFID tag.

5.4.4. BT Context Harvester

A Personal Area Network (PAN) may be described as a netwoek consisting of the connection of personal devices, allowing information exchange over short ranges. This term is used on several of today's short range wireless technologies like Bluetooth (BT), ZigBee, Infrared etc. In this section we look in to how Bluetooth, as a PAN technology, may be used to enrich the context information related to a person.

In Akogrimo so far, SLP in combination with user location has been used to establish computational context. Here we evaluate the capability of Bluetooth as a replacement or supplement of this method. The novelty lies in the use of radio communication to discover enabled devices in a persons immediate surrounding. The use of BlueTooth is intriguing because

- Bluetooth has become a fundamental standard for short-range communication, and many devices (laptops, mobile phones, PDAs) are shipped with BT;
- BT has mechanisms for service discovery and may easily collect information about the surrounding BT-enabled devices;
- The usual range is 10 meters allowing discovery of devices in immediate proximity;
- We assume that the BT radio on a users personal device is used, hence the range of discovery will follow the person for whom we wish to gather computational context;
- Some GPS modules are equipped with Bluetooth for communication with e.g. PDAs and mobile phones. This opens for retrieving GPS-based location information.

5.4.4.1. Design

A conceptual overview of the system is given in Figure 57. The remainder of this section gives a brief description of design and implementation. For details, please refer to [Dav06].

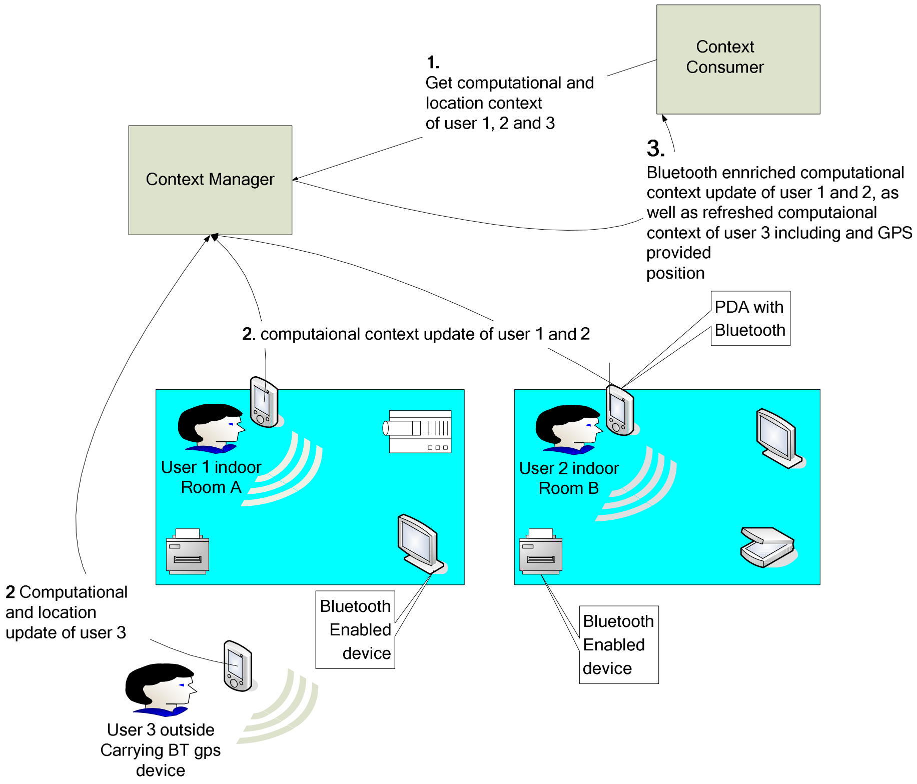

Context Consumer
1. Get computational and location context of user 1, 2 and 3
Context Manager
3. Bluetooth ennriched computational context update of user 1 and 2, as well as refreshed computaional context of user 3 including and GPS provided position
2. computaional context update of user 1 and 2
PDA with Bluetooth
User 1 indoor Room A
User 2 indoor Room B
2 Computational and location update of user 3
Bluetooth Enabled device
Bluetooth Enabled device
User 3 outside Carrying BT gps device

Figure 57 Using Bluetooth for acquisition of computational context

Mainly two components are required to realise the suggested system:

- A Context Harvester at the user's Mobile Terminal (described in this section)
- A Web Service interface at the Context Manager for receiving context updates (described in 5.4.3.5)

Context Harvester (CH) will interact with the Java API to the Bluetooth Software. CH runs cycles of device discovery. This process returns information of nearby Bluetooth devices of the terminal. A webservice client is trigged by these discovery processes, enabling uploading of context changes to the Context Manager, see Figure 58.

The cycles of device discovery is done every 5 second. If new devices are discovered, these devices will be further investigated by performing service discovery. Any relevant device/service information is stored as an array, acting as an out buffer for potential Web Service method invocations. For every third cycle a check is done to see whether the device context profile of the terminal has changed since the last WS-update. If there has been a change, due to new or no longer available Bluetooth devices, this check will trigger a WS call, again causing a CM update. Before the context is updated against the Web Service at CM, up to date location information (provided by a GPS Bluetooth service) is included in the update message. If no changes appear, no update will be made and CH continues its device and service discovery. Figure 59 gives a brief overview of that while interactions and messages are shown in Figure 60.

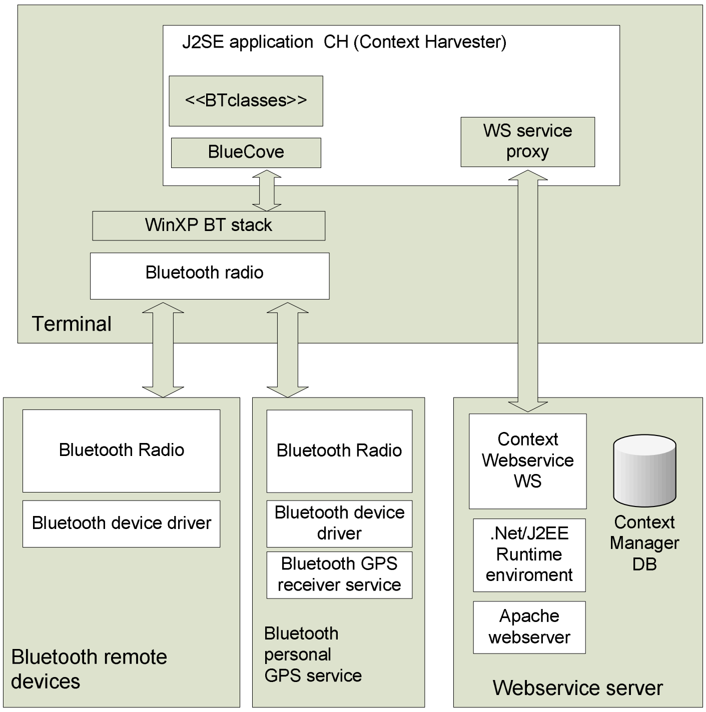


Figure 58 Context Harvester – Modules and external parts

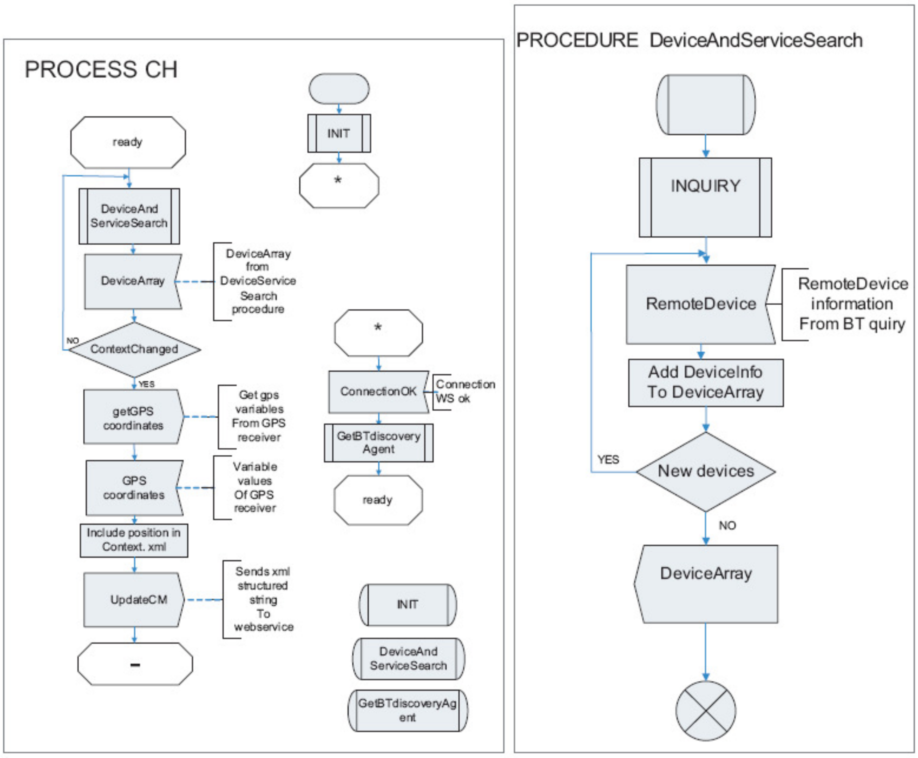

PROCESS CH
ready
DeviceAnd ServiceSearch
DeviceArray
DeviceArray from DeviceService Search procedure
NO
ContextChanged
YES
getGPS coordinates
Get gps variables From GPS receiver
GPS coordinates
Variable values Of GPS receiver
Include position in Context. xml
UpdateCM
Sends xml structured string To webservice
-
INIT
*
*
ConnectionOK
Connection WS ok
GetBTdiscovery Agent
ready
INIT
DeviceAnd ServiceSearch
GetBTdiscoveryAg ent
PROCEDURE DeviceAndServiceSearch
INQUIRY
RemoteDevice
RemoteDevice information From BT quiry
Add DeviceInfo To DeviceArray
YES
New devices
NO
DeviceArray

Figure 59 Basic functionality of Context Harvester (CH)

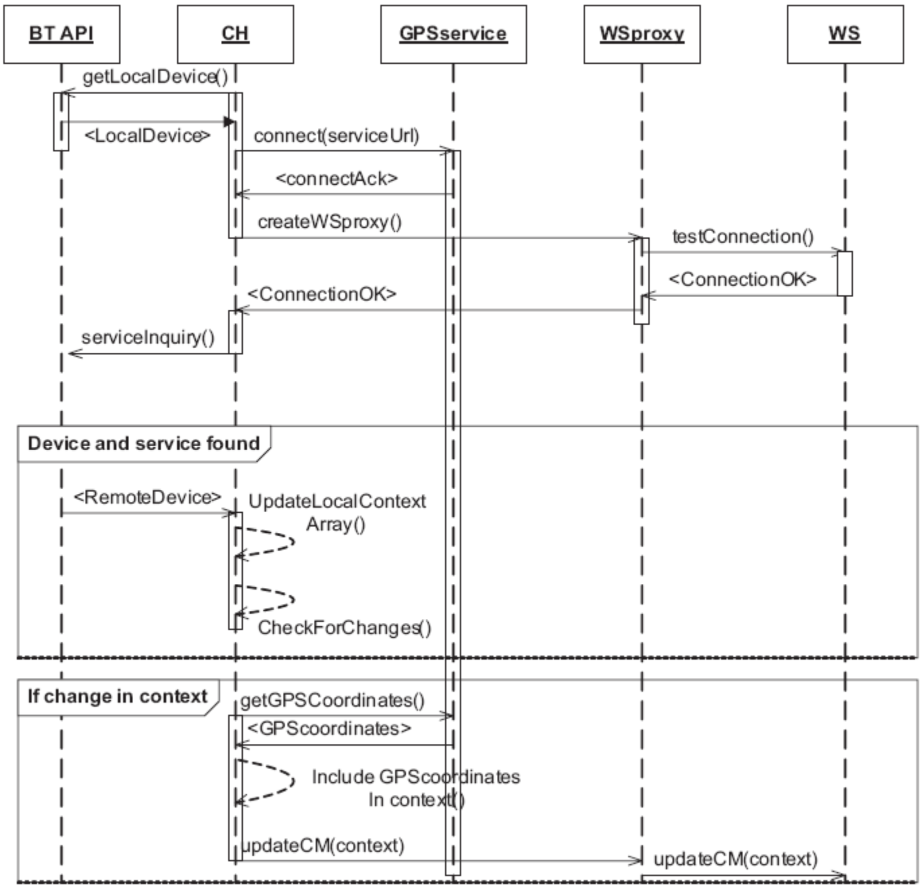


Figure 60 Message sequence chart for Context Harvester (CH)

5.4.4.2. Implementation

The system has been implemented and it is working, based on the Bluetooth API for Windows XP.

5.4.5. RFID software

One way to get location in Akogrimo is by RFID readings. Different RFID sensors can be placed around specific locations and connected to a RFID Server. The server knows the location of each of the different attached sensors.

5.4.5.1. Design

When a RFID tag is read by a sensor the RFID Server sends a message to the RFID Gateway of the Context Manager. RFID Servers are 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 a position object containing longitude, latitude, altitude, horizontal accuracy and vertical accuracy as well as UTM coordinates.

An overview of the RFID gateway and RFID Reader is given in Figure 61. 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.

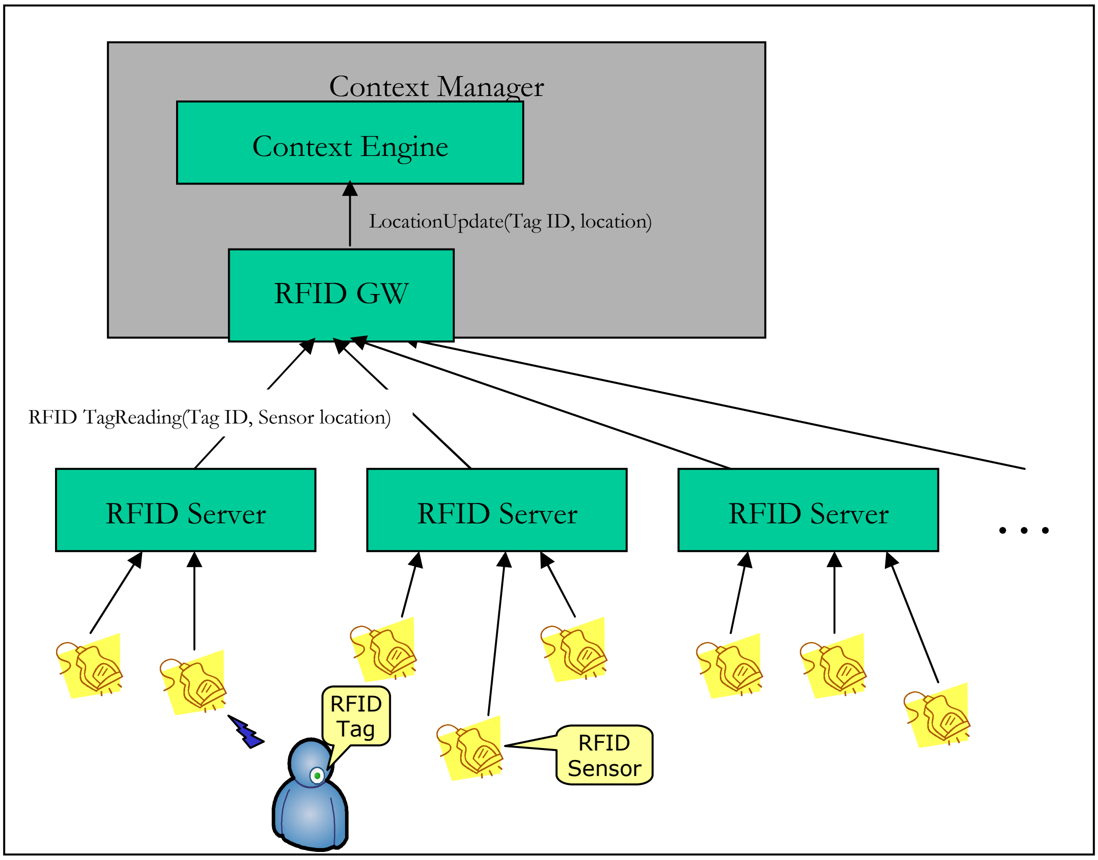


Figure 61 Basic elements for provisioning of RFID position

Figure 62 shows how the different components could interact when a tag is read at a RFID sensor. The Server 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 Server 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.

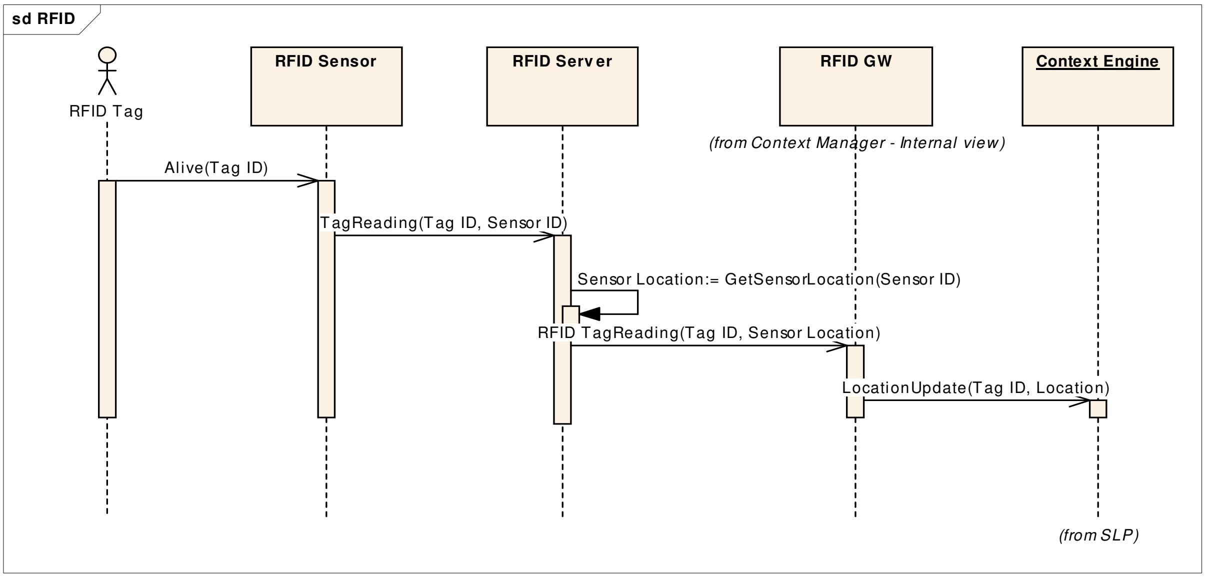

sd RFID
RFID Tag
RFID Sensor
RFID Server
RFID GW
Context Engine
(from Context Manager - Internal view)
Alive(Tag ID)
TagReading(Tag ID, Sensor ID)
Sensor Location:= GetSensorLocation(Sensor ID)
RFID TagReading(Tag ID, Sensor Location)
LocationUpdate(Tag ID, Location)
(from SLP)

Figure 62 MSC for provisioning of RFID position

5.4.5.2. Implementation

An outline of the RFID system is shown in Figure 61. 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. Libraries used are JPSCS (0.8.0) from IBM for both platforms and PCSC (PC Smart Cards) from [MUSCLE] for Linux platforms.
  - RFID Server software: Java-based, Telenor proprietary software for handling signalling conversion between RFID sensors and RFID GW.

5.4.5.3. TagReader

A simplified version of the RFID Server has been implemented in order to support local RFID tag readings for other services locally. Instead of reporting a position and RFID reading to the Context Manager it stores the RFID tag in a local file accessible by the local system and services.

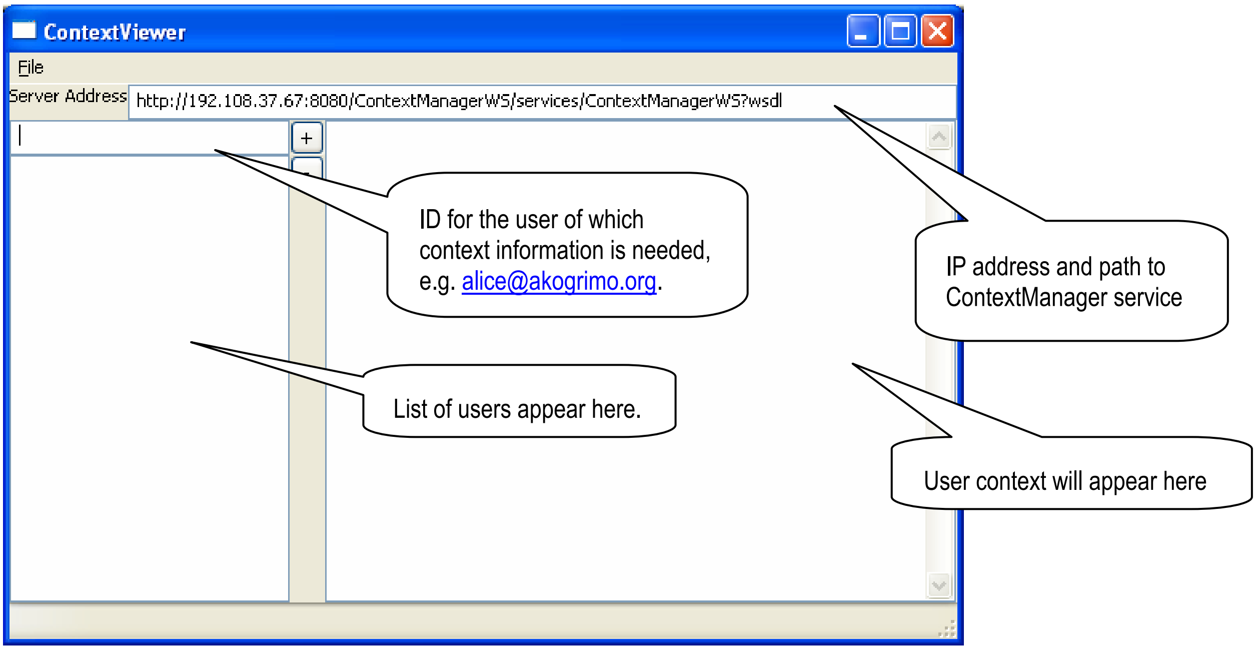
5.4.6. Context Viewer

The Context Viewer is an application that retrieves and displays context information from the Context Manager, see Figure 63.

The application subscribes to an akogrimo id in the Context Manager. Context Viewer uses the CM-interface based on WS, see Table 34 in Section 5.2.2. The user may provide the akogrimo id using a graphical interface. The application works as a Context Client, it subscribes by a call that includes the akogrimo id and the IP of the client for a return address. Upon receipt of a subscription the Context Manager notifies the client with current context data. If the akogrimo id does not exist no context will be shown. If subscription succeeds, the Context Viewer will be notified of any context.

The context can be viewed as the client wants to. The different entities of context can be parsed in the context xml and be dealt with separately. A location can be parsed and presented by a clickable button that opens up a map service like Google Maps to show the location. Other parses can make graphical icons that are clickable, a monitor can be shown, and when selecting that monitor show the properties for that monitor.

Currently the Context Viewer only views the context as textual data. The context is shown as a complete xml received from the Context Manager. It is implemented in Python [Python].



Server Address http://192.108.37.67:8080/ContextManagerWS/services/ContextManagerWS?wsdl

ContextViewer
File
+
ID for the user of which context information is needed, e.g. alice@akogrimo.org.
IP address and path to ContextManager service
List of users appear here.
User context will appear here

Figure 63 Context Viewer interface

5.5. Testing

Product testing is an important part of the overall task of Quality Assurance. Although Akogrimo does not aim to produce industry level products for a business or consumer market, thorough testing is necessary to ensure more efficient SW development, smoother integration of components, and overall reduction of risks. This section gives a brief description of test tools, testing environments, and test cases for Context Manager and related software.

5.5.1. Tools

By test tools we mean components and software that are designed with the purpose of testing the Context Manager. The tools will not be used normally when running the Context Manager service. This may include interface simulators and load generators. The following test tools have been provided for the context manager:

- **Context Viewer (CV)** is an independent software component that acts as a context consumer. CV will contact an instance of CM to subscribe to context for a specific user. CV is GUI-based and the user's context will be displayed in a window. It may hold several subscriptions at the same time. CV contacts CM over a WS-based interface, hence will only test this part of the CM - Context Consumer interface (i.e. not WS-N). However, the most essential test purpose of CV is to verify the content of a user's context information.
- **GetUserEPR /SubscribeToUser** are two command-line based applications that together act as a context consumer. They are used for testing the WS-Notification interface to Context Manager. GetUserEPR will retrieve the endpoint reference (EPR) for a specific user, while SubscribeToUser will subscribe to notification updates for the same user. User context will be printed to default system output, usually the terminal where the scripts are executed. The most essential test purpose of this tool is to verify the content of a user's context information.
- **A4C Simulation Mode** is useful when running CM in an environment where the A4C Server cannot be reached. When in A4C Simulation mode, the CM simulates information from the A4C Server, the data is retrieved from the local machine. Simulation mode is activated/deactivated in the CM properties file.

- **Location Simulator** is a command-line replacement for the regular RFID Server. It will also replace use of RFID Smart Cards (SC) and SC readers, and hence is useful when such equipment is unavailable. End user data (RFID Tags) and RFID SC locations may be easily configured; hence Location Simulator offers a fully functional replacement of the regular RFID Server.

5.5.2. Test environment

Suitable test environments are necessary for appropriate testing throughout the realization process. For testing CM the following environments have been used:

- **Programmer's PC**: CM is fully functional on a Linux machine, and nearly fully functional on a Windows machine. The following components will not work in Windows:
    - A4C interface. This may be compensated by using the A4C Simulation Mode
    - SLP GW. This GW requires libraries that only work on Linux. As a compensation, the GW may be run on Linux. Communication between the GW and CM is over RMI
- **Telenor test lab**: Augustus, a Linux PC with Ubuntu, is available in Telenor's lab for testing CM. It has the same configuration as machines in the Akogrimo test lab, and hence provides a good test environment. Augustus also hosts processes that CM communicates with, such as a SIP UA and a SLP DA. Due to firewall settings according to the Test Lab's security profile, it is not possible for all other Akogrimo partners to connect to Augustus.
- **Akogrimo test lab**. Akogrimo has two test labs; one in Madrid (hosted by partner UPM) and one in Stuttgart (hosted by partner USTUTT). The Stuttgart lab has been fully configured for running the entire Akogrimo architecture, and hence offers an complete environment for verification of the various components.

5.5.3. Test data

This section suggests data sets that are sufficient for evaluating the test cases in Section 5.5.4.

Users and user profile are listed in Table 45. The column "MD Authorization" indicates if Monitoring Daemon is authorized to view the context of a user.

Table 45 Users and authorization

User	Profile	MD Authoriz.
Alice	<userID>alice@akogrimo.org</userID> <RfidTag>D2C7F651B28804004659255641102102</RfidTag> <SipUrl>alice@akogrimo.org</SipUrl>	1
Bob	<userID>bob@akogrimo.org</userID> <RfidTag>B278FA51618804004659255641102102</RfidTag> <SipUrl>bob@akogrimo.org</SipUrl>	1
Carol	<userID>carol@akogrimo.org</userID> <RfidTag>7280F751548804004659255641102102</RfidTag> <SipUrl>carol@akogrimo.org</SipUrl>	0
David	Null	1
Enya	Null	0

Relevant test locations are listed in Table 46. Locations must match in the settings for RFID Readers (file /etc/rfidserver/reader.positions.config) and in the location of SLP Services (file /etc/slp.reg). In addition it is possible to define locations as suggested in 5.3.2.

Table 46 Locations

Trondheim	Stuttgart Loc 1, Universitat Stuttgart, Allmandring 30, Basement, Room U1.005	Stuttgart Loc 2, Universitat Stuttgart, Allmandring 30, Science Truck
reader.1.zone=32	reader.2.zone=32	reader.3.zone=32
reader.1.zoneLetter=V	reader.2.zoneLetter=U	reader.3.zoneLetter=U
reader.1.northing=7035380	reader.2.northing=5398567.68	reader.3.northing=5398548
reader.1.easting=569839	reader.2.easting=506937.77	reader.3.easting=506949
reader.1.latitude=63.44	reader.2.latitude=48.74009	reader.3.latitude=48.7500
reader.1.longitude=10.4	reader.2.longitude=9.094363	reader.3.longitude=9.084363
reader.1.altitude=0.0	reader.2.altitude=449.0	reader.3.altitude=450.0
reader.1.accuracy=0.0	reader.2.accuracy=0.0	reader.3.accuracy=0.0

Registration of local services are handled by SLP, and must match with what is desired in the tests to be performed. Table 47 is an excerpt from /etc/slp.reg, where Stuttgart Location 1 has a display accessible over the VNC protocol, and Stuttgart Location 2 has a display accessible via the SIP protocol.

Table 47 Local services: Excerpt from /etc/slp.reg

```
##Register service
# Printer 2 Trondheim
service:printer:lpr://129.241.219.163,en,65535
#use default scopes
#[attrid"="val1<newline>]
owner=bob3
description=Printer2 Trondheim
utmNorthing=7035370
utmEasting=569839
utmLetter=V
utmZone=32

##Register service
# Display1 Stuttgart - Location 1
service:display:vnc://ksat97.ipv6.rus.uni-stuttgart.de,en,65535
#use default scopes
#[attrid"="val1<newline>]
owner=Brynjar
description=Display
utmNorthing=5398567
utmEasting=506937
utmLetter=U
utmZone=32
location=U1.005,lab
display=8
sipuri=display1_stuttgart@akogrimo.org

##Register service
# Beamer Stuttgart - Location 2
service:display:sip://carol@ksat54.ipv6.rus.uni-stuttgart.de,en,65535
owner=EHealth_lab
description=Display
utmNorthing=5398548
utmEasting=506949
utmLetter=U
utmZone=32
location=science_truck
display=50
sipuri=carol@akogrimo.org
```

5.5.4. Test cases

Ideally, tests should be performed at several levels: during implementation, integration of components, value chain tests for the entire platform, FAT (Factory Acceptance Test) and SAT

(System Acceptance Test). The test cases described in this section aim at providing a passable set of value chain tests for CM, i.e. are not exhaustive and not for all the above mentioned levels.

An example of test cases and results are described in Table 48. In the reminder of this section we shall only describe test cases, the date, result and test status has been omitted.

Table 48 Sample test table

Test case #	Test description	Expected result	Test run date	Observed result	Test F/P[7]
A4C 1	MD request Alice's context	Context returned (Profile exists, MD is authorized)	11th Oct 2006	As expected	P

5.5.4.1. A4C tests

Test Data: Table 45 Users and authorization

Test environment: USTUTT Lab. CM should contact A4C Server. See test results in CM logs.

Test scenario: Request from Context viewer, CM performs A4C lookup (context consumer authorization, get user profile), observe that context is / is not returned to context viewer.

Table 49 A4C test cases

Test case	Test description	Expected result
A4C 1	MD request Alice's context	Context returned (Profile exists, MD is authorized)
A4C 2	MD request Bob's context	Context returned (Profile exists, MD is authorized)
A4C 3	MD request Carol's context	Context not returned (Profile exists, MD is not authorized)
A4C 4	MD request David's context	Context not returned (Profile dos not exist, MD is authorized)
A4C 5	MD request Enya's context	Context not returned (Profile does not exists, MD is not authorized)

5.5.4.2. Location (RFID) and local services (SLP) tests

The objective of this test is to verify that users are located in the right places, and that the correct local services are detected.

Test data: All data mentioned in Section 5.5.3.

Test environment: USTUTT Lab. RFID readers are connected to Windows machines. Observe results in Context Viewer (CV).

Test Scenario: Bob, Alice and Carol move between USTUTT Location 1 and 2. Start location is Trondheim.

Table 50 Location test cases

Test case	Test description	Expected result
RFID 1	Presumption: No location registered for Alice	
	1. CV[8] subscribes	Gets default context from Context DB
	2. Alice registers at USTUTT location 1 (lab)	Corresponding context is updated with CV: Location, two SLP devices (display 1 and 2)
	3. Alice registers at USTUTT location 2 (truck)	Corresponding context is updated with CV: Location, one SLP device (Beamer)
RFID 2	Presumption: No location registered for Bob	
	1. CV	Gets default context from Context DB

[7] F: Fail, P: Pass. Alternatives include NA: Not Applicable, Po: Postponed

[8] Context Viewer

Test case	Test description	Expected result
	subscribes	
	2. Alice registers at USTUTT location 1 (lab)	Corresponding context is updated with CV: Location, two SLP devices (display 1 and 2)
	3. Alice registers at USTUTT location 2 (truck)	Corresponding context is updated with CV: Location, one SLP device (Beamer)

5.5.4.3. Presence (SIP Presence) tests

The objective of this test is to verify that when presence is set by a user in the SIP UA, the same presence information reaches a context consumer.

Test data: Table 45 Users and authorization

Test environment: USTUTT Lab. Observe results in Context Viewer (CV).

Test Scenario: Bob, Alice and Carol change between two presence settings. Initial state is "Not registered" (i.e. the SIP UA has not been started).

Table 51 Location test cases

Test case	Test description	Expected result
Presence 1	Presumption: Alice's SIP UA has not been started	
	1. CV[9] subscribes	Gets default context from Context DB
	2. Alice starts her SIP UA, and registers with SIP server at USTUTT lab.	Alice's new presence is updated with CV
	3. Alice changes presence	Alice's new presence is updated with CV
Presence 2	Repeat Case Presence 1 for user Bob	As above
Presence 3	Repeat Case Presence 1 for user Carol	As above

[9] Context Viewer

6. Service Discovery

Due to the importance of this subject, in this second circle of the project the Service Discovery (SD) mechanism will be created to obtain a much more flexible and powerful architecture. In order to achieve this, a quite profound redesign is needed.

The Service discovery mechanisms at network level are divided in four parts: Network Discovery (ND), Device Discovery (DD), Local Service Discovery (LSDS) and Grid Service Discovery (GrSDS), as represented in Figure 64.

The challenge of providing and using mobility and location specific information is a central part in the project, however, practice shows that most of the times location changes are not transparent for the users. This is because these changes often require some kind of re-configuration at different levels. This reconfiguration should not place a burden to end users, and therefore this process needs to be partially automated. SD tries to overcome and solve some of these issues by providing an automatic look up of resources at different layers. These are the mentioned Network Discovery, Device Discovery, Local Service Discovery and Grid Service Discovery.

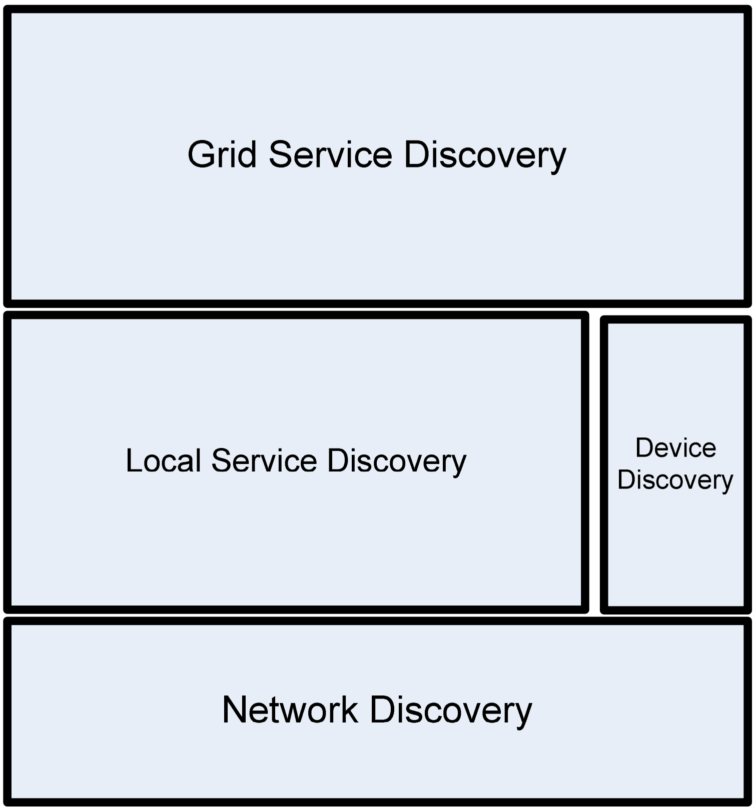


Figure 64 Service Discovery

This document mainly focuses on Local Service Discovery mechanisms, which arises once a device is totally functional in the network. This layer deals with general types of services such as printers, beamers, webcams, music sharing, video streaming, locally available Web Services, local UDDI servers, file sharing, contacts sharing, bookmark sharing, file servers, games, pop3, cvs, http, ssh, ftp servers, collaborative working tools, proxies, time servers and so on.

6.1. Network Discovery

This layer of SD understands the network as the initial point other Akogrimo services can be discovered and used. The process is divided in two stages: network discovery and network configuration.

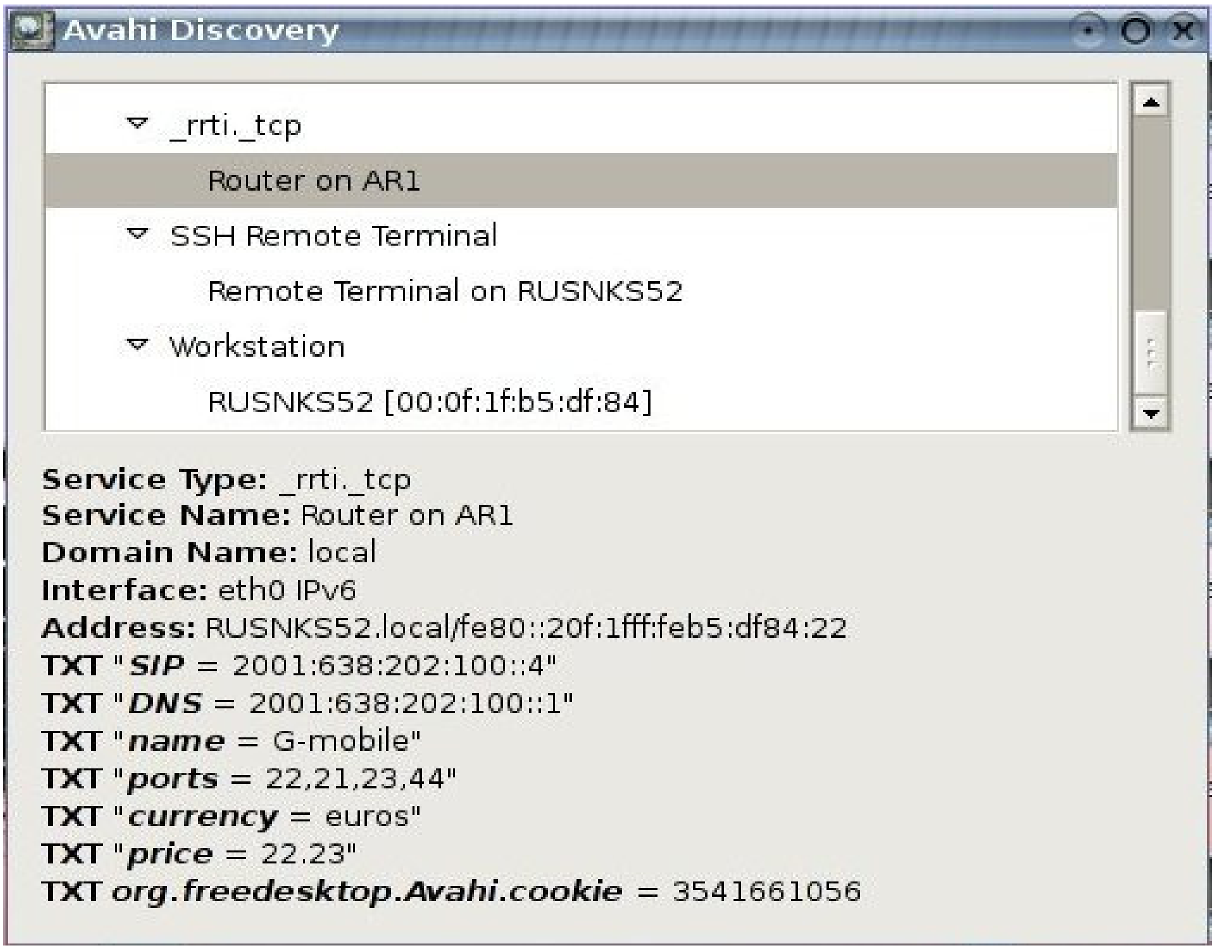
The first stage focuses on network discovery and happens before network configuration at IP level. Depending on the concrete situation, maybe ten different networks can be accessed through the different interfaces at that precise moment. Network selection takes place according to information such as bandwidth, cost of use, signal quality or usable ports for that network.

Secondly, the network needs to be configured for the specific location. This implies: the establishment of an IP address, routing configuration, local DNS to use, network security configuration, discovery of local authentication authorities (for authentication to take place at IP level), local SIP Proxies, etc.

In order to solve the problem of network discovery the first important thing to think about would be at which level the information necessary for a user's bootstrap can be transmitted to him. It is clear the OSI Link Layer information can not be used as we strive to provide an IP based media access independent architecture, therefore it needs to contain information from upper OSI layers. The most reasonable possibilities would be:

- Router Advertisements: this information could be embedded in router advertisements sent by access routers. The drawback would be that RAs are sent quite often and most of the times this information would be send with no reason wasting valuable resources.
- DHCP: The dynamic host configuration protocol was firstly designed to provide the host with an IP and some other information such as DNS server. This could be a reasonable solution.
- Zeroconf: Taking into account that in IPv6 the address is already provided by the access routers by means of RAs and that with DHCP the achievement of this information is triggered by the client and not the network, Zeroconf can provide extra features such as the retransmission to the user of changes occurred in the network or e.g. advertise that the AR is not usable any more before it shuts down so that the client knows what is going on.

An example of the user of Zeroconf for this is shown in the following figure, where a router AR1 is discovered with information about available ports, cost, name and the location of the SIP proxy and DNS. In addition to this it would be important to also transmit the location of the Authentication authority that needs to be contacted to enter the network.



Service Type: _rrti._tcp
Service Name: Router on AR1
Domain Name: local
Interface: eth0 IPv6
Address: RUSNKS52.local/fe80::20f:1fff:feb5:df84:22
TXT "***SIP*** = 2001:638:202:100::4"
TXT "***DNS*** = 2001:638:202:100::1"
TXT "***name*** = G-mobile"
TXT "***ports*** = 22,21,23,44"
TXT "***currency*** = euros"
TXT "***price*** = 22.23"
TXT ***org.freedesktop.Avahi.cookie*** = 3541661056

6.2. Device Discovery

Perpendicular to the rest of types of service discovery, device discovery focuses on aggregating information about the possible devices that are around the user's environment and are not necessarily part of the Akogrimo infrastructure. Examples of these devices could be Bluetooth phones, or beamers, or even people with which the user interacts directly. The rational for supporting non-Akogrimo devices would be the fact that for example, the Workflow would be aware and therefore allow the user to use a non-Akogrimo beamer to visualize a certain document as long as the policies related to this act allow the use of a public-non-related beamer, instead of making the user move all the way to the other side of the building where an Akogrimo beamer is located. This kind of discovery would integrate different types of mechanism, especially ad-hoc, such as Bluetooth or UPnP.

6.3. Local Service Discovery

The experience gained on the usage of the Session Initiation Protocol (SIP) as an efficient context propagation source and the study of classical service discovery mechanisms reveal that these notification capabilities can also be used to implement a powerful service discovery mechanism. This might sound like a forced approach at a first glance; however, it is needed to keep in mind that the underlying procedures are very akin to each other. SIP was primarily designed to negotiate any type of sessions between two or more participants. In this case the participants happen to be a client that looks for a certain service and the server that provides it. The mechanisms by which these two peers interrelate after they have been put in contact are independent from this system. Distinctive features of this system not present in other SD mechanisms would include the ability to make fine-tuned subscriptions to service changes or subscription to services even before they are available. In addition other interesting capabilities could be its reduced chattiness, since users receive information just about services they are subscribed to; bandwidth efficiency, considering that partial publications and notifications could be used to only convey changes in the description of the services instead of complete documents; and an xml-enhanced service description, which opens the door for the inclusion of semantic description.

6.3.1. Description of the architecture

From a conceptual point of view, service discovery architecture usually comprises three main actors: a **Service Provider**, or the entity which offers the service, a **Service Requestor** (the inquiring entity) and a **Repository** or **Directory Agent**, which facilitates a requestor to select the service provider that better matches its requirements.

The SD mechanism proposed in this document also matches this approach. The definition of the main entities involved and its mapping with the usual SD actors is the following:

- **Service Publication Agent (SPA)**: it is a SIP User Agent supporting SIP PUBLISH which will be in charge of making public the service capabilities. It plays the role of the service provider in a classic SD architecture.
- **Service Discovery Subscriber (SDS)**: this is the consumer of the service discovery information (i.e. the service requestor in the SD architecture); it could be an end user terminal, some network entity or another service controlling a SIP UA with SIP SUBSCRIBE and SIP NOTIFY support.
- **Service Discovery Agent (SDA)**: it is the Directory Agent in the conceptual SD architecture. This entity is primarily responsible for storing, aggregating and maintaining service information from different SPAs and the corresponding subscriptions from SDSs. Additionally, and due to the SIP mechanisms it is based on, it is also capable to inform the subscribers if any change on the services capabilities they are interested in occurs. Note that this is only a functional description: depending on the concrete implementation, and for scalability purposes, the functionality of the SDA can be distributed among several machines, instead of having a centralized server

This role mapping is depicted in Figure 65.

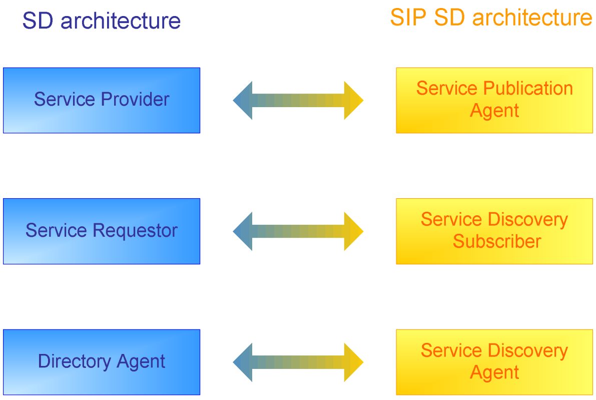


Figure 65 SIP Service Discovery role mapping

6.3.2. Functional Description

The definition of a Service Discovery framework includes the description of the formal mechanisms to enable the requestors to find required service providers. These mechanisms rely on two basic functionalities:

- **Service publication**: this is the way in which one service provider registers and publishes its services in the Directory Agent to be discovered by Service Requestors.
- **Services search**: this mechanism defines how a requestor can make a request to the repository, indicating the services features it is interested in, and how the reply is delivered.

The proposed SIP SD mechanism (which is described in Figure 66) makes use of SIP PUBLISH as a publication mechanism, while a specific-event based SIP SUBSCRIBE/NOTIFY framework is used to implement queries and services searches.

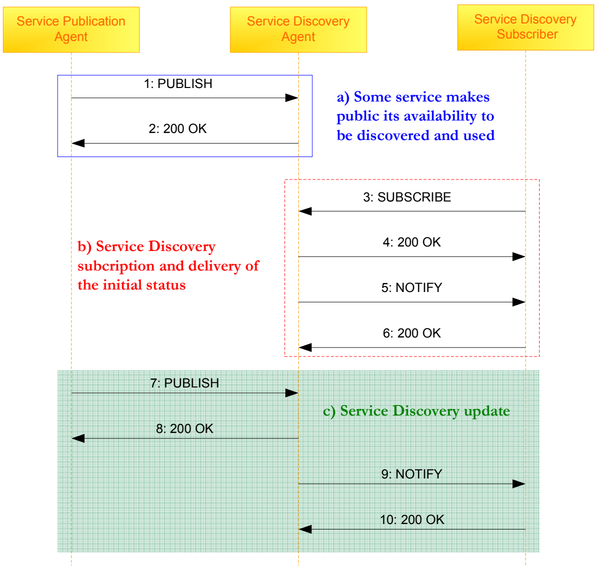

Service Publication Agent
Service Discovery Agent
Service Discovery Subscriber
1: PUBLISH
2: 200 OK
a) Some service makes public its availability to be discovered and used
3: SUBSCRIBE
4: 200 OK
b) Service Discovery subcription and delivery of the initial status
5: NOTIFY
6: 200 OK
7: PUBLISH
c) Service Discovery update
8: 200 OK
9: NOTIFY
10: 200 OK

Figure 66 SIP Service Discovery data flow

When a service wants to make public some information related to itself, it uses its associated SPA to send a SIP PUBLISH request containing all relevant resource information, which will be included in a xml document named "**Resource Description Document" (RDD)** containing some basic information like resource description, owner, location, IP address and how to access it. This interaction is shown in the first box in the figure above.

The SIP URI to be used in the PUBLISH request will be shared by all entities or individual resources associated to the service (e.g. all available beamers will register itself as beamer-service@domain. This URI should be standardized to name services much in the way of other SD protocols – like SLP or Zeroconf – do it). The PUBLISH message is sent to the SDA which will store, maintain and aggregate the information provided by different SPAs corresponding to the same service (e.g. the complete description of the "beamer service" will consist on the composition of the individual information provided by each registered beamer, that is, the aggregation of each individual RDD). This aggregated document that the SDA handles is the "**Service Description Document**" **(SDD)**.

When an entity (end users or other services) wants to find a service with specific characteristics, it can subscribe to this information via SIP SUBSCRIBE (the dash-lined box in Figure 66). The SIP SUBSCRIBE message can contain a body, the "**Service Subscription Document**" **(SSD)** which will be used to implement searches, or restrict the scope of the information the SDS wants to receive (e.g. only a list of the available beamers located in a certain building must be provided). This kind of filtering mechanism is envisaged in the general SIP-Specific Event Notification Request for Comment (RFC 3265, section 4.4.3) and will be part of the event package specification associated to the new kind of event it is described here. The format of the SSD also still needs to de defined.

Immediately after a valid subscription is created, the requested information is sent via a SIP NOTIFY message. This information, named the "**filtered Service Description Document**" **(fSDD),** will typically consist on a subset of the SDD, containing only the RDD of the resources matching the query. If the subscription document were empty, the whole SDD will be delivered.

Apart from services publications and searches, the usage of this mechanism provides an additional functionality, as depicted in the dark box of Figure 66). Using SIP SD, any change on the services properties or availability could be also notified during the period of time in which the subscription is valid (e.g. if there is a valid subscription to the "beamer service" in certain area, and then a new beamer appears, the corresponding subscriber will receive an update of the RSDD containing the new beamer information; in a similar way, subscribers will be notified if some resource is no longer available). This new functionality, combined for instance with the advanced capabilities that NGN can provide, would allow very sophisticated versions of the SD mechanisms.

6.3.3. Implementation

The logical entities that constitute the SIP SD architecture, i.e. SPA, SDS and SDA, use the already present signalling capabilities. Therefore, it is needed to build them on top of previous software pieces, making use of the available interfaces. Besides, it is also needed to carefully define the interfaces of those entities that will interact with application software, as is the case for the SPA and the SDS.

6.3.3.1. Service Publication Agent

As stated before, the SPA is the logical entity in charge of publishing service information. It has been designed as a library that offers a signalling API. This API allows for other software elements and applications to handle the signalling needed for service information publication.

In a way similar to that used for the presence infrastructure elements, the SPA is built on top of the SIP Event Publication Agent (EPA) (see section 2.2.2).

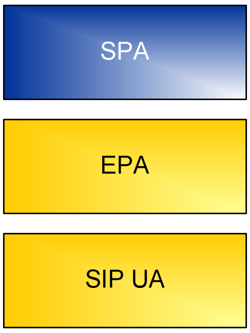


Figure 67 SPA architecture

The SIP EPA is a generic event publication agent. Its functionality can be used to publish any kind of information related with any kind of event. This approach provides a flexible architecture in which adding support for new event packages only requires the creation of new classes that make use of the signalling services provided by the EPA. The EPA uses the signalling handling capabilities of the SIP UA to send and receive the required messages. Thanks to the functionality offered by the SIP EPA, the SPA only needs to take care of SIP SD-specific features.

Since the SPA is a library targeted to allow whichever service to publish its SIP SD information, its JAVA API includes methods to manage the needed signalling: initial publication, publication update, publication refresh and automatic publication expiration handling.

6.3.3.2. Service Discovery Subscriber

The SDS is in charge of handling subscription to service information obtained through the SIP Service Discovery infrastructure. It also handles the notifications sent by the SDA when changes in service information occur. The implementative approach used to create the SDS is the same than that of the SPA. The SDS is also a library that offers the needed functionality as a JAVA API.

The SDS relays on the functionality offered by the SIP Event Subscription Agent (ESA).

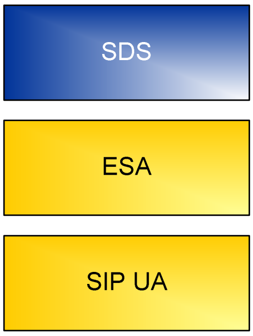


Figure 68 SDS architecture

In a way similar to that of the EPA to event information publication, the ESA provides the methods needed to create generic subscriptions in an event server. It also handles the notification messages that such a server would send when processing the subscription. Thanks to this approach, the SDS just needs to handle SIP SD-specific issues.

The SDS is presented as a JAVA library. It can be used by any software entity willing to obtain service information from the SIP SD infrastructure. Its API allows for the creation and management (i.e. refreshment, update and expiration) of multiple simultaneous subscriptions, as well as the associated notifications.

6.3.3.3. Service Discovery Agent

The Service Discovery Agent (SDA) is responsible for storing, aggregating and maintaining service information from different SPAs and the corresponding subscriptions from SDSs., as mentioned before. It also informs the subscribers if any change on the services capabilities they are interested in takes place.

From an implementation perspective, it basically consists of a Java servlet implementing the required functionality. This servlet is deployed in a services container (or Application Server, following the IMS - IP Multimedia Subsystem - terminology) based on a modified Tomcat that comprises a JAIN SIP-based SIP UA, being fully compliant with the SIP servlet API specification [JSR116].

The main internal architecture of the application server is represented in the figure below:

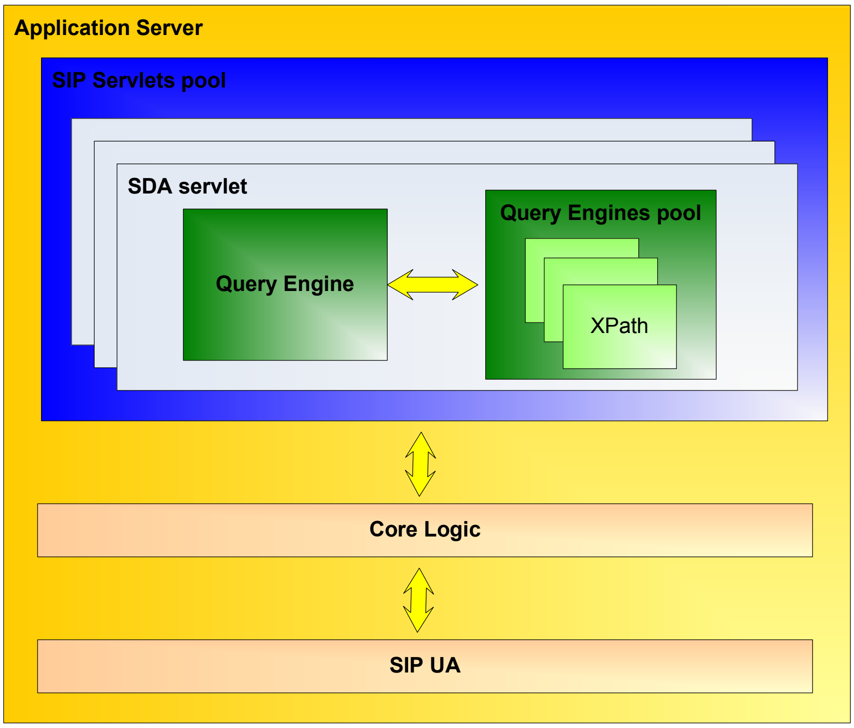


Figure 69 Application server and SDA internal structure

The underlying SIP UA is responsible for capturing the incoming SIP messages. The application server decides what to do with each SIP request following the instructions of the $CATALINA_HOME/conf/sip.xml file, which indicates the SIP servlet to be invoked. If the SDA servlet is responsible for the request handling (this will occur for PUBLISH and SUBSCRIBE requests whose 'Event' header field indicates "sipsd"), the SDA processing takes place.

The SDA servlet maintains a list of RDDs containing the related information of each resource. The most interesting process takes place when a SIP SUBSCRIBE request arrives, and it is depicted in the following figure:

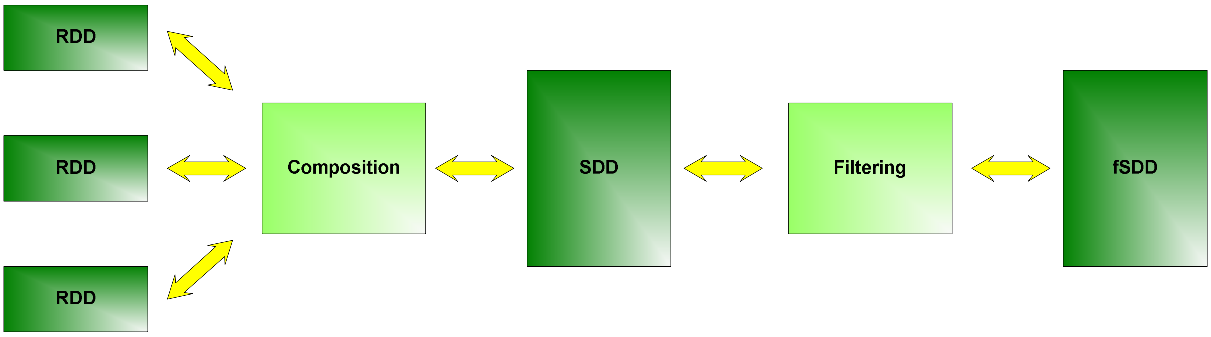


Figure 70 SD subscription processing

First, the SDA servlet examines the RequestURI in order to know to which resources the subscriber is interested in (beamers, printers, cameras), and builds a raw or unfiltered list of all related resources, the SDD; then, and using the information contained in the subscription (e.g. a 'Content-Type' header field with a value "application/sd-xpath+xml" indicates that the SDS contains an XPath query), it decides which Query Engine should be applied to perform the searching, or filtering, of the raw document. The output of this processing is the fSDD document to be included as the body of a SIP NOTIFY to be sent to the subscriber.

The SDA has been designed to support multiple and different search mechanisms. This functionality has been implemented by using java pluggins which are loaded and indexed automatically during the application servlet startup. A SD query engine pluggin consists on a java jar file located at $CATALINA_HOME\ webapps\panther-examples\WEB-INF\lib\qe including a manifest file containing a section like this:

Name: QueryEnginePlugin

QueryEngineType: related content type

QueryEngineClass: class implementing org.akogrimo.sip.sd.sda.qe.common.IQueryEngine interface

The related content type is a string value that indicates to which kind of subscriptions this query engine should be applied (e.g. application/sd-xpath+xml for xpath queries).

IQueryEngine implementation performs the query itself. The definition of this interface is described in the table below:

Table 52 IQueryEngine interface definition

Method	**Description**
String match(String strServiceDoc, String strFilterDoc)	Filters the specified doc by using the filter passed as parameter @param strServiceDoc @param strFilterDoc Returns a String object containing the filtered document
boolean validateFilter(String strFilterDoc)	Return if the filter passed as parameter is valid @param strFilterDoc Returns true if the filter passed as parameter is valid

This approach allows developers to easily develop a new type of filter mechanisms, since they only are restricted to these few rules; then, they only have to put the jar file on the corresponding folder and restart Tomcat. The SDA servlet automatically loads the new library and registers the new search mechanisms, which is ready to be used. For the Akogrimo demonstrator, an XPath query engine has been developed in this way.

Finally, the SDA servlet can be configured by modifying the file $CATALINA_HOME\webapps\panther-examples\WEB-INF\web.xml Configurable parameters are described in the table below:

Table 53 SDA servlet parameters

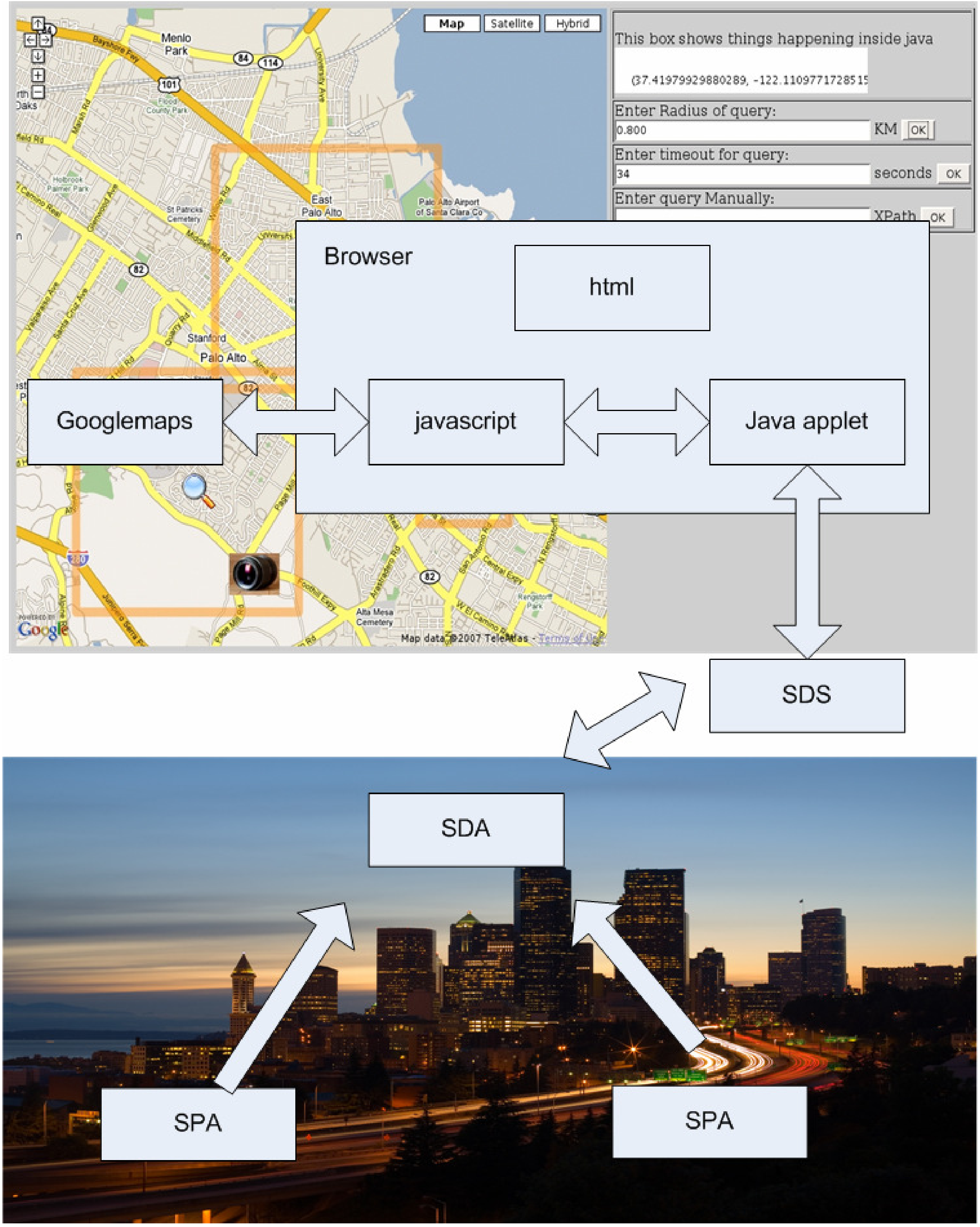
Parameter	**Purpose**
minExpires	Minimum allowed expiration time for publications and subscriptions [10]
maxExpires	Maximum allowed expiration time for publications and subscriptions [10]

[10] Inherited from Events servlet. The SDA servlet inherits most of its functionality from a generic Events servlet, since more of the processes that must take place when a PUBLISH or SUBSCRIBE arrives are independent from the type of event being handled.

Parameter	Purpose
defExpires	Default allowed expiration time for publications and subscriptions [10]
defaultEvent	Type of event being handled [10]
defaultContentType	Type of content type being handled [10]
logFile	Output log file
plugginDir	Absolute path to the folder from the query engine pluggins are loaded.

6.3.3.4. SIP Service Discovery test application

In order to demonstrate the capabilities of SIP SD in a way that is easy to see for an external expectator a GUI was developed that uses the underlying java classes that call the SIP SD methods. The proof of concept implementation used utilizes XML service descriptions and XPath in order to perform queries. The GUI uses a browser and googlemaps to perform queries in a certain area over a map. Whenever a camera is discovered a MM session can be triggered by clicking on the corresponding camera. A Java applet is used to handle this and the communication between the browser and the SDS implementation. Once a SDS subscribes to a certain area for a certain period it will receive information of all cameras (SPA) appearing, disappearing or moving in the corresponding area.



This box shows things happening inside java

(37.41979929880289, -122.110977172851

Enter Radius of query: 0.800 KM OK

Enter timeout for query: 34 seconds OK

Enter query Manually: XPath OK

Map data ©2007 TeleAtlas - Terms of Use

Figure 71 SIP Service Discovery test application

6.4. Grid Service Discovery

Grid Service Discovery has been worked on, but information about it can be found in D 4.4.3, Report on the implementation of the Application Support Services Layer [D4.4.3], delivered at the same time as this deliverable.

Annex A. References

[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 19.12.2006

[D3.1.1] J. Jähnert et al: "Overall Architecture Definition and Layer Integration". Akogrimo Deliverable D3.1.1, v1.0. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=36 URL last visited: 19.12.2006

[D3.1.2] J. Jähnert et al: "Detailed Overall Architecture". Akogrimo deliverable D3.1.2, v1.0. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=73 URL last visited: 19.12.2006

[D3.1.3] J. Jähnert et al: "Overal Architecture Definition and Layer Integration". Akogrimo deliverable D3.1.3. Will be available at http://www.akogrimo.org

[D4.1.1] N. Inácio et al: "Consolidated Network Layer Architecture". Akogrimo Deliverable D4.1.1 07.11.2005, v1.0. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=43 URL last visited 19.12.2006

[D4.1.3] N. Inácio et al: "Final Network Service Provisioning Concept". Akogrimo Deliverable D4.1.3. Scheduled for December 2006

[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 19.12.2006

[D4.2.2] Osland et al: "Final Integrated Services Design and Implementation Report." Akogrimo Deliverable D4.2.2. November 2005. URL: http://www.akogrimo.org/modules.php?name=UpDownload&req=getit&lid=42 URL last visited 19.12.2006

[D4.2.3] Solsvik et al: "Final Integrated Services Design and Implementation Report." Akogrimo Deliverable D4.2.3 December 2006. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=110 URL last visited 14.07.2007

[D4.4.1] G. Laria et al: "Architecture of the Application Support Service Layer". Akogrimo Deliverable D4.4.1, v1.0. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=38 URL last visited: 19.12.2006

[D4.4.2] G. Laria et al: "Prototype Implementation of the Grid Application Support Service layer". Akogrimo deliverable D4.4.2 v1.0. URL: http://www.mobilegrids.org/modules.php?name=UpDownload&req=getit&lid=74 URL last visited: 19.12.2006

[D4.4.3] G.Laria et al: “Report on the Implementation of the Application Support Servies Layer”. Akogrimo Deliverable D4.4.3. Scheduled for December 2006

[D5.1.2] Akogrimo: “Integrated Prototype”. Akogrimo Deliverable D5.1.2. URL: http://www.akogrimo.org/modules.php?name=UpDownload&req=getit&lid=80
Last visited: 19.12.2006

[Dav06] H. Davidsen: “Bluetooth in context acquisition”. Master thesis, Dept of Telematics, NTNU. July 17, 2006

[Dey99] A.K.Dey,G.D.Abowd,D.Salber; ”A Context-Based Infrastructure for Smart Environments”. In Proceedings of the 1st International Workshop on Managing Interactions in Smart Environments (MANSE '99), 1999, pp. 114-128; URL: www.cc.gatech.edu/fce/contexttoolkit/pubs/MANSE99.pdf; (online 19.12.2006)

[eduGAIN] D. Lopez et al: “DJ5.2.3,1: Best Practice Guide – AAI Cookbook – First Edititon”. 2006.
URL: http://www.geant2.net/upload/pdf/GN2-06-236v5-DJ5-2-3-1_Best_Practice_Guide-AAI_Cookbook_First_Edition.pdf
URL last visited November 29th, 2006

[EduPerson] Internet2: "EduPerson Object Class Specification (200604)”. 2006. Internet2 Middleware Architecture Committee for Education, Directory Working Group (MACE-Dir)
URL: http://www.educause.edu/eduperson/ URL last visited November 29th, 2006

[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-13.txt
Last visited 19.12.2006

[Gu04] Tao Gu, Xiao Hang Wang, Hung Keng Pung, Da Qing Zhang: “An Ontology-based Context Model in Intelligent Environments”, In Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference, San Diego, California, USA, January 2004

[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 19.12.2006

[JNI] Java Native Interface (JNI), http://java.sun.com/docs/books/tutorial/native1.1/
Last visited 19.12.2006

[JSR116] Java Specification Requests 116 “SIP Servlet API”, http://jcp.org/en/jsr/detail?id=116 Last visited 19.12.2006

[Mos03] S. Kouadri Mostéfaoui, G. Kouadri Mostéfaoui, “Towards A Contextualisation of Service Discovery and Composition for Pervasive Environments”, In Proc. of the Workshop on Web-services and Agent-based Engineering (WSABE), 2003.

[MUSCLE] MUSCLE Webpage: URL http://musclecard.com/ Last visited 19.12.2006

[MySQL] MySQL Database. URL: http://www.mysql.com Last visited 19.12.2006

[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 19.12.2006

[OpenGIS] Open GIS Consortium, Inc.: “OpenGIS® Simple Features Specification For SQL”. Revision 1.1. May 5, 1999. URL: http://www.opengis.org/docs/99-049.pdf Last visited 19.12.2006

[openslp] OpenSLP Webpage. URL www.openslp.org Last visited 19.12.2006

[Python] Python web page. URL: http://www.python.org/. URL last visited 07.12.2006

[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 19.12.2006

[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 19.12.2006

[Ros04-2] J. Rosenberg, “A Watcher Information Event Template-Package for the Session Initiation Protocol”, RFC 3857 Internet Engineering Task Force, August 2004. URL: http://www.ietf.org/rfc/rfc3857.txt Last visited 19.12.2006

[SAML] OASIS: “Security Assertion Markup Language, SAML”, October 2005. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security Last visited 19.12.2006

[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/%201371/oasis-sstc-saml-core-1.0.pdf Last visited 19.12.2006

[Shib] M. Erdos, S. Cantor: “Shibboleth-Architecture Draft v04”. 2001. URL: http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v04.pdf. URL last visited November 29th, 2006

[SIMLIE] SIMLIE - Semantic Interoperability of Metadata and Information in unLike Environments. SIMLIE Location Ontology: http://simile.mit.edu/2005/05/ontologies/location. URL last visited 19.12.2006

[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 19.12.2006

[Wang04] Xiao Hang Wang, Tao Gu, Da Qing Zhang, Hung Keng Pung: “Ontology Based Context Modeling and Reasoning using OWL“. Context Modeling and Reasoning Workshop at PerCom. 2004

[XML] Extensible Markup Language (XML) 1.0 (Third Edition). W3C Recommendation 04 February 2004. URL: http://www.w3.org/TR/REC-xml/

[xmlSchema] XML Schema Part 0: Primer Second Edition. W3C Recommendation 28 October 2004. URL: http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/ Last visited 19.12.2006

Annex B. SIP Presence Handling

B.1. Presence User Agent

B.1.1. Test cases

In order to check the Presence Agent functionality, we propose several test cases, basically oriented to verify the functional requirements. 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 state).	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 (empty/default if no publications from that presentity stored).

Test Case	Description	Result
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).

B.1.2. Installation, configuration and usage

SIP Server (including embedded SIP PA) is currently installed, configured and running on ksat115. The installation comprises two debian packages, the one corresponding to the standard SER (named ser) and the one that installs the PA library (libsipserver), the configuration file (ser.cfg) and the extended command utility (serctl). The Akogrimo forge server includes them and the corresponding readme file indicates which version of the files correspond each other.

By default, the installation configures the SIP Server (and so, the embedded Presence Agent) to show logs messages to the console from it were started. It can be also configured to redirect these logs to a file. Log level can be changed by editing file `/etc/ser/ser.cfg` (find the line `debug = 3` and type a higher level of detail; 3 should be enough).

SIP PA has their own debug system and traces level independent from the general log framework of the SIP Server. To modify the, modify the line

```
modparam("esc", "trace_level",   1)
```

and type the desired level of detail.. Current values should be enough, so in principle you should not to change this file at all.

In order to star the SIP server, type (from any directory):

```
sudo ser start
```

After starting, you should see general information (like aliases, the IP address and the port where the SIP Server is listening, current debug level…) and some particular information regarding esc module (Presence Agent) and a4c module (note that for the demo this module will not be used). If no error messages SIP Server is running properly. You should see some logs regarding registration, proxy transactions, presence publication and delivery of presence information.

To stop the SIP Server, type:

```
sudo pkill ser
```

There is a script utility useful to monitor how the component is running. From a console type:

```
sudo serctl [usage]
```

where [ ] indicates an optional command, and the serctl utility will show the usage menu. Some useful commands are:

```
sudo serctl ul show [AoR]
```

if you want to see the Location Service status (users currently registered). Type yes when requested. If you provide a SIP AoR, the information to be shown will be only related to the corresponding AoR.

```
sudo serclt moni
```

to monitor ser activity. If ser is running, you should see some information about the status of the SIP Server internal cycles (should be moving) and the requests and replies that the component has handled.

```
sudo serclt esc show [AoR]
```

to check the status of the publications and subscriptions. If you provide a SIP AoR, the information to be shown will be only related to the corresponding AoR.

Annex C. A4C

C.1. QoS Profile Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<xs:schema         xmlns:xs="http://www.w3.org/2001/XMLSchema"         elementFormDefault="qualified"
attributeFormDefault="unqualified">
  <xs:element name="ANUP">
    <xs:annotation> <xs:documentation>Akogrimo Network User Profile</xs:documentation>
    </xs:annotation>
    <xs:complexType>
      <xs:sequence>
        <xs:element name="UID" type="xs:string">alice@akogrimo.org
        </xs:element>
        <xs:element name="PRIO" type="xs:short">2
        </xs:element>
        <xs:element name="MAC" type="xs:string">00-02-44-8E-3D-84
        </xs:element>
        <xs:element name="DSCP" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="ID" type="xs:unsignedInt">1
              </xs:element>
              <xs:element name="Macro">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="MaxUpBW" type="xs:unsignedLong">256
                    </xs:element>
                    <xs:element name="MaxDownBW" type="xs:unsignedLong">512
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
              <xs:element name="Layout" maxOccurs="unbounded">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element name="Filter">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="FromAddr" type="xs:string">*
                          </xs:element>
                          <xs:element name="ToAddr" type="xs:string">*
                          </xs:element>
                          <xs:element name="FromPort" type="xs:unsignedInt">*
                          </xs:element>
                          <xs:element name="ToPort" type="xs:unsignedInt">*
                          </xs:element>
                          <xs:element name="Protocol" type="xs:string">UDP
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="QoSBundle" type="xs:unsignedLong">2
                    </xs:element>
                    <xs:element name="PeriodOfDay" maxOccurs="unbounded">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="Start" type="xs:string">*
                          </xs:element>
                          <xs:element name="End" type="xs:string">*
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```